

**EVALUASI KAPABILITAS DETEKSI  
ANTI-SPYWARE TERHADAP PACKER  
SPYWARE MODE STEALTH**

**Proposal Tugas Akhir**

Oleh

**Nathaniel Liady  
18222114**



**PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
Desember 2025**

## **LEMBAR PENGESAHAN**

### **EVALUASI KAPABILITAS DETEKSI ANTI-SPYWARE TERHADAP PACKER SPYWARE MODE STEALTH**

### **Proposal Tugas Akhir**

Oleh

**Nathaniel Liady  
18222114**

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 5 Desember 2025

Pembimbing

Prof. Dr. Ir. Suhardi, M.T.  
NIP. 196312111990011002

## DAFTAR ISI

<b>DAFTAR GAMBAR . . . . .</b>	<b>v</b>
<b>DAFTAR TABEL . . . . .</b>	<b>vi</b>
<b>DAFTAR KODE . . . . .</b>	<b>vii</b>
<b>I PENDAHULUAN . . . . .</b>	<b>1</b>
I.1 Latar Belakang . . . . .	1
I.2 Rumusan Masalah . . . . .	2
I.3 Tujuan . . . . .	2
I.4 Batasan Masalah . . . . .	3
I.5 Metodologi . . . . .	3
<b>II STUDI LITERATUR . . . . .</b>	<b>6</b>
II.1 <i>Information Gathering</i> . . . . .	6
II.2 <i>Social Engineering</i> . . . . .	7
II.3 <i>Cyber Security</i> . . . . .	8
II.4 <i>Anti-Spyware</i> . . . . .	9
II.4.1 <i>Endpoint Detection and Response (EDR)</i> . . . . .	10
II.4.2 <i>Signature-Based Anti-Virus</i> . . . . .	11
II.4.3 <i>Behavior-Based Anti-Virus</i> . . . . .	11
II.4.4 <i>Entropy-Based Anti-Virus</i> . . . . .	12
II.5 <i>Malware</i> . . . . .	12
II.5.1 <i>Fileless Malware</i> . . . . .	14
II.5.2 <i>Obfuscation</i> pada <i>Malware</i> . . . . .	14
II.6 <i>Spyware</i> . . . . .	15
II.6.1 <i>Spyware Mode Stealth</i> . . . . .	16
II.6.2 <i>Packer Spyware Mode Stealth</i> . . . . .	16
II.6.3 <i>Payload</i> pada <i>Packer Spyware</i> . . . . .	17
II.7 Studi Sebelumnya . . . . .	18
<b>III ANALISIS MASALAH . . . . .</b>	<b>20</b>
III.1 Analisis Kondisi Saat Ini . . . . .	20
III.1.1 Masalah Kerentanan dan Ketertinggalan <i>Anti-Malware</i> konvensional . .	20
III.1.2 Keterbatasan Deteksi Berbasis Tanda Tangan ( <i>Signature-Based</i> ) . . .	20
III.1.3 Gagal Menganalisis Code <i>Fileless Execution</i> . . . . .	21

III.1.4	Efektivitas Teknik Evasi Sederhana . . . . .	21
III.1.5	Kurangnya Deteksi Proaktif Terhadap Arsitektur Serangan . . . . .	21
III.1.6	Gap Analysis Celah Deteksi <i>Packer Spyware Mode Stealth</i> . . . . .	21
III.2	Analisis Kebutuhan . . . . .	22
III.2.1	Kebutuhan Fungsional . . . . .	22
III.2.2	Kebutuhan Nonfungsional . . . . .	23
III.3	Analisis Pemilihan Solusi . . . . .	24
III.3.1	Alternatif Solusi . . . . .	24
III.3.2	Rust . . . . .	24
III.3.3	C/C++ . . . . .	25
III.3.4	Python (PyInstaller) . . . . .	25
III.3.5	Analisis Penentuan Solusi . . . . .	25
III.3.6	Hasil Analisis Penentuan Solusi . . . . .	27
<b>IV</b>	<b>DESAIN KONSEP SOLUSI . . . . .</b>	<b>30</b>
IV.1	Diagram Konseptual . . . . .	30
IV.1.1	. . . . .	30
IV.1.2	. . . . .	31
IV.1.3	. . . . .	32
IV.2	Klarifikasi Tugas Desain . . . . .	32
IV.2.1	. . . . .	32
IV.2.2	. . . . .	34
IV.2.3	Spesifikasi Mesin Uji . . . . .	35
IV.3	Konseptualisasi Arsitektur Solusi . . . . .	35
IV.3.1	Komponen Arsitektur <i>Packer</i> . . . . .	35
IV.3.1.1	<i>Stub Packer (Loader)</i> . . . . .	36
IV.3.1.2	<i>Encrypted Lightweight Payload</i> . . . . .	36
IV.3.2	Alur <i>Packer</i> dengan artefak <i>Stealth</i> . . . . .	36
IV.4	Actual Support . . . . .	39
IV.4.1	Modul yang Diimplementasikan . . . . .	39
IV.5	Verifikasi Internal dan Rencana Evaluasi . . . . .	40
IV.5.0.1	Kriteria Keberhasilan (Internal Success Criteria) . . . . .	40
IV.5.0.2	Ruang Lingkup Evaluasi . . . . .	41
<b>V</b>	<b>RENCANA SELANJUTNYA . . . . .</b>	<b>42</b>
V.1	Rencana Implementasi . . . . .	42
V.1.1	Langkah-langkah Implementasi . . . . .	42
V.2	Rencana Evaluasi . . . . .	43

V.2.1	Metode Pengujian: Studi Kuasi-Eksperimental Komparatif . . . . .	43
V.2.2	Kriteria Keberhasilan . . . . .	44
V.2.3	Analisis Risiko . . . . .	44

## **DAFTAR GAMBAR**

I.1	<i>Design Research Methodology Framework</i>	4
IV.1	<i>Packer Konvensional</i>	31
IV.2	<i>Packer dengan artefak Stealth</i>	31
IV.3	<i>Intended Impact Model</i>	33
IV.4	<i>Packer dengan artefak Stealth</i>	37

## DAFTAR TABEL

II.1	Studi Sebelumnya ( <i>Packer Spyware</i> dan Mekanisme Evasi) . . . . .	19
III.1	Kebutuhan fungsional (Functional Requirements) . . . . .	22
III.2	Kebutuhan fungsional (Functional Requirements) . . . . .	23
III.3	Kebutuhan nonfungsional (Non-Functional Requirements) . . . . .	23
III.4	Alternatif Solusi . . . . .	24
III.5	Kriteria Desain Packer Spyware . . . . .	26
III.6	Bobot kriteria hasil metode EWM . . . . .	28
III.7	Hasil Perhitungan Bobot EWM dan Skor Solusi . . . . .	29
IV.1	Perbandingan <i>Packer</i> Konvensional dan <i>Packer</i> dengan Artefak <i>Stealth</i> . . . . .	32
IV.2	Key Factors prototipe <i>packer</i> . . . . .	33
IV.3	Spesifikasi Lingkungan Target untuk Pengujian . . . . .	35
V.1	Rencana Implementasi Prototipe Stealth . . . . .	43
V.2	Kriteria Keberhasilan dan Indikator Pengukuran . . . . .	44
V.3	Analisis Risiko Pengembangan dan Pengujian Prototipe Stealth . . . . .	45

## **DAFTAR KODE**

# BAB I

## PENDAHULUAN

### I.1 Latar Belakang

Ancaman siber saat ini telah mengalami evolusi signifikan, bergeser dari *malware* tradisional berbasis *file* menuju serangan yang lebih canggih dan tersembunyi, seperti *fileless malware* dan *spyware*. Evolusi ini menciptakan tantangan mendasar bagi sistem *anti-malware* konvensional, karena pertahanan yang mengandalkan deteksi berbasis tanda tangan (*signature-based*) menjadi tidak efektif. *Fileless malware* secara khusus memanfaatkan *utility* sistem operasi yang sah, seperti PowerShell dan Windows Management Instrumentation (WMI), untuk menjalankan kode berbahaya langsung di memori (*in-memory*) tanpa meninggalkan jejak *file* pada *disk*.

Kelemahan sistem pertahanan ini diperkuat oleh hasil studi empiris yang menguji kapabilitas deteksi produk keamanan komersial. Penelitian telah menunjukkan bahwa teknik *evasion* sederhana seperti enkripsi dan injeksi proses terbukti sangat efektif dalam menghindari deteksi. Bahkan, dalam sebuah studi di tahun 2023, sejumlah *anti-malware* yang diuji hanya mampu mendeteksi sebagian kecil dari variан *malware* yang disamarkan. Temuan ini menegaskan bahwa terdapat kerentanan substansial terhadap *malware* lama yang dimodifikasi dengan trik penyamaran baru, sementara *scanner anti-virus* juga sering gagal menganalisis kode yang dikemas.

Penelitian ini memfokuskan diri pada pengembangan dan pengujian modul *Packer Spyware Mode Stealth*. *Packer* ini dirancang untuk mencapai keberhasilan *Initial Access* dengan menerapkan teknik *obfuscation* tingkat tinggi yang langsung menyerang kelemahan inti *anti-malware*: *Signature-Evasion* dan *In-Memory Execution*. Karena *spyware* modern memanfaatkan teknik penghindaran analisis dan deteksi baik dalam bentuk statis maupun dinamis, penelitian ini berupaya menghasilkan artefak yang sulit teridentifikasi.

Dari berbagai temuan tersebut, terlihat jelas bahwa terdapat *gap* signifikan pada kemampuan deteksi *signature-based* dan *behavior-based* terhadap teknik *packing*, *obfuscation*, dan *fileless execution*. Gap inilah yang menjadi fokus utama penelitian ini, yang bertujuan untuk secara empiris menguji dan menganalisis kemampuan deteksi solusi keamanan yang tersedia di pasar terhadap *spyware* kustom.

## I.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah dalam penelitian ini adalah:

1. Bagaimana kapabilitas sistem *anti-spyware* dalam mendeteksi aktivitas tersembunyi dari perilaku *spyware mode stealth* pada lingkungan uji terkontrol?
2. Bagaimana pendekatan paling efektif untuk mengembangkan *packer spyware stealth* yang mampu menghindari deteksi *anti-malware*
3. Bagaimana teknik penyamaran (*obfuscation*) dan metode eksekusi berbasis memori (*in-memory execution*) mempengaruhi kemampuan deteksi produk *anti-spyware* saat ini?
4. Apa saja celah keamanan dan kelemahan spesifik yang ditemukan pada produk *anti-spyware* dan EDR ketika dihadapkan pada serangan *spyware* kustom yang dirancang untuk menghindari deteksi?

## I.3 Tujuan

Secara umum, tujuan dari pelaksanaan tugas akhir ini adalah untuk mengukur dan mengevaluasi efektivitas produk *anti-spyware* dan EDR komersial dalam mendeteksi *spyware mode stealth* yang dikembangkan dengan teknik-teknik penghindaran deteksi modern.

Secara spesifik, tujuan yang ingin dicapai adalah:

1. Menganalisis dan mengidentifikasi teknik-teknik evasi yang paling efektif digunakan oleh *spyware* untuk menghindari deteksi dari *anti-spyware* dan EDR.
2. mengembangkan modul *Packer Spyware Mode Stealth* dengan *lightweight payload* untuk *Initial Access*.
3. Melakukan pengujian komparatif prototipe *packer spyware mode stealth* pada sejumlah produk *anti-spyware* dan EDR komersial untuk mengevaluasi tingkat keberhasilan dan kegagalan deteksi.
4. Mendokumentasikan dan mempublikasikan celah keamanan yang ditemukan pada produk *anti-spyware*, serta menyusun rekomendasi mitigasi untuk pe-

ngembangan sistem pertahanan yang lebih adaptif dan tangguh. kriteria keberhasilan dari pelaksanaan tugas akhir ini adalah:

- Prototipe *spyware* berhasil dikembangkan dengan setidaknya dua teknik evasif (misalnya, enkripsi dan obfuscation) dan mampu menghindari deteksi oleh salah satu produk *anti-spyware* yang diuji.
- Hasil pengujian menunjukkan bahwa ada perbedaan signifikan dalam tingkat deteksi antara format skrip dan *anti-spyware* yang berbeda.

Kedua kriteria tersebut menjadi indikator utama untuk menilai efektivitas penelitian, serta menunjukkan sejauh mana solusi yang ditawarkan mampu mengungkap kelemahan sistem keamanan siber saat ini. Pencapaian terhadap kriteria ini diharapkan dapat menjadi dasar pertimbangan dalam peningkatan kapabilitas deteksi *anti-spyware* dan EDR di masa depan

#### I.4 Batasan Masalah

Batasan masalah dalam pelaksanaan tugas akhir adalah sebagai berikut:

1. Penelitian ini hanya berfokus pada pengujian kapabilitas deteksi *anti-spyware* terhadap aktivitas awal (*Initial Access*) yang dilakukan oleh *spyware mode stealth* pada sistem operasi *desktop*.
2. Evaluasi hanya mencakup perangkat anti *spyware* yang dipilih sesuai kriteria penelitian, tidak membandingkan seluruh produk anti malware yang ada.
3. Aktivitas *malware* yang diujikan dibatasi secara ketat pada tahap penyusupan, enkripsi *payload* dan *fileless execution*.
4. Tugas akhir ini dikerjakan secara kelompok dengan anggota penelitian sebagai berikut:
  - Nathaniel Liady
  - M. Kasyfil Aziz
  - Audra Zelvania Putri Harjanto
  - Khayla Belva Annandira

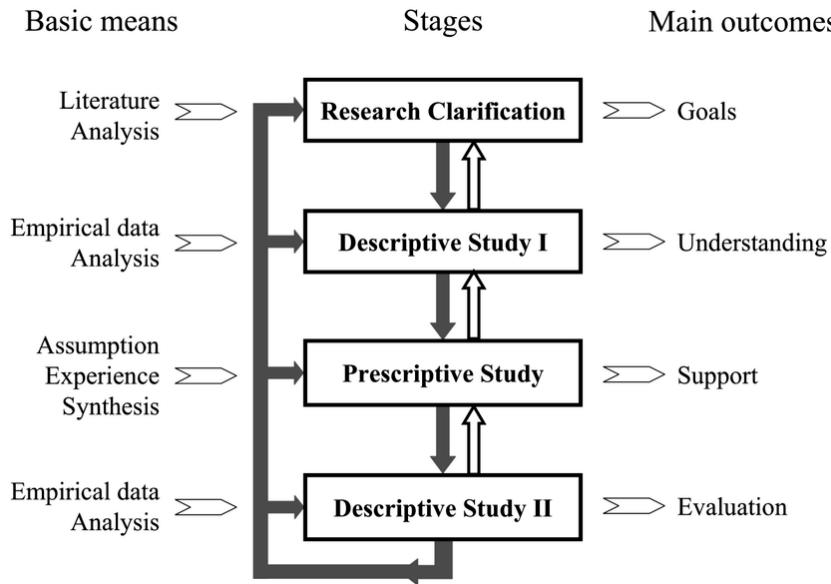
#### I.5 Metodologi

Metodologi penelitian yang digunakan adalah *Design Research Methodology* (DRM) dikenalkan oleh Blessing dan Chakrabarti (2009).<sup>1</sup> DRM dibuat dengan tujuan supaya riset dilakukan dengan lebih efektif dan efisien. Metodologi ini terdiri dari empat tahap utama yaitu Research Clarification (RC), Descriptive Study I (DS-I),

---

1. L. T. M. Blessing & A. Chakrabarti, Design Research Methodology, 2009.

Perspective Study (PS), dan Descriptive Study II (DS-II). Berikut ini adalah gambaran dari kerangka kerja DRM.



Gambar I.1 *Design Research Methodology Framework*

### 1. *Research Clarification (RC)*

Fase ini akan dimulai dengan identifikasi masalah utama: adanya celah yang signifikan antara teknik serangan siber modern, khususnya *spyware mode stealth*, dan kemampuan deteksi solusi keamanan yang ada. Pengumpulan data awal akan dilakukan melalui tinjauan literatur komprehensif, termasuk laporan industri, artikel akademis, dan berita, untuk memahami lanskap ancaman dan teknik evasif yang digunakan untuk menghindari deteksi *anti-spyware* dan EDR. Hasil dari fase ini adalah rumusan masalah yang jelas dan terperinci, yang akan menjadi landasan untuk seluruh penelitian.

### 2. *Descriptive Study I (DS-I)*

Pada fase ini, analisis mendalam terhadap masalah yang telah dirumuskan akan dilakukan dengan mengumpulkan data dan informasi yang relevan. Ini mencakup analisis rinci mengenai teknik-teknik evasi canggih, seperti penggunaan *PowerShell* dan obfuscation, untuk memahami bagaimana ancaman ini bekerja dan mengapa mereka sulit dideteksi. Selain itu, studi-studi terdahulu yang telah menguji bypass *antivirus* akan dikaji untuk mendapatkan wawasan mengenai metode dan potensi hasil. Sebagai bagian penting dari fase ini, alur kerja atau arsitektur teknis dari serangan *spyware* akan disusun, yang akan menjelaskan fase-fase seperti *Initial Access*, *Establish Foothold*, *Persistence*, *Data Collection*, dan *Exfiltration*. Diagram alur yang telah ada

akan menjadi representasi visual dari arsitektur ini.

### 3. *Perspective Study (PS)*

Fase ini merupakan inti dari DRM, di mana solusi untuk masalah yang telah didefinisikan akan dirancang dan dikembangkan. Artefak yang akan dibuat adalah prototipe *spyware mode stealth* dengan fungsi-fungsi untuk *initial access*. Desain prototipe akan mengintegrasikan teknik evasi yang telah diidentifikasi pada fase sebelumnya, seperti enkripsi *payload* dan penggunaan PowerShell untuk menjalankan kode langsung di memori. Berbagai modul, termasuk Packer untuk menyamarkan *payload*.

### 4. *Descriptive Study II (DS-II)*

Fase terakhir ini bertujuan untuk menguji dan mengevaluasi efektivitas solusi yang telah dikembangkan. Serangkaian pengujian eksperimental akan dilakukan dalam lingkungan virtual yang terisolasi untuk mengukur tingkat deteksi berbagai produk *anti-spyware* terhadap prototipe *spyware*. Hasil pengujian akan dianalisis untuk mengidentifikasi teknik evasi mana yang paling efektif dan mengapa produk keamanan tertentu gagal atau berhasil dalam mendeteksi ancaman. Analisis ini akan memvalidasi temuan awal dan memberikan kontribusi nyata pada pemahaman tentang kerentanan sistem keamanan modern, yang akan menjadi dasar untuk rekomendasi perbaikan dan penelitian lebih lanjut.

## BAB II

### STUDI LITERATUR

Bab ini membahas landasan teori dan kajian literatur yang relevan untuk mendukung penelitian ini. Studi literatur dilakukan untuk memahami konsep, teori, dan teknologi yang mendasari penelitian, termasuk *information gathering*, *social engineering*, *cyber security*, *Anti-Spyware*, *malware*, dan *spyware*. Selain itu, kajian terhadap penelitian terdahulu yang berkaitan juga dilakukan untuk mengidentifikasi solusi yang sudah ada, celah penelitian, serta pendekatan yang dapat diadopsi atau dikembangkan lebih lanjut. Hasil dari studi literatur ini akan menjadi dasar dalam merumuskan solusi desain yang diusulkan dalam penelitian ini.

#### II.1 *Information Gathering*

*Information gathering* merupakan tahap awal yang sangat penting dalam proses pengujian keamanan siber maupun kegiatan intelijen digital. Tahap ini bertujuan untuk mengumpulkan informasi sebanyak mungkin mengenai target, baik berupa sistem, jaringan, organisasi, maupun individu, dengan tujuan memahami permukaan serangan yang tersedia. Verma dkk. (2021) menjelaskan bahwa kegiatan *information gathering* dilakukan untuk memetakan aset dan mengidentifikasi potensi kerentanan sebelum serangan atau pengujian keamanan dilakukan. Secara umum, proses ini terbagi menjadi dua pendekatan utama, yaitu pasif dan aktif. Pengumpulan informasi secara pasif dilakukan tanpa melakukan interaksi langsung dengan sistem target, misalnya dengan memanfaatkan data publik dari mesin pencari, basis data domain, atau sumber intelijen terbuka (*Open Source Intelligence/OSINT*). Sebaliknya, pendekatan aktif melibatkan aktivitas langsung seperti *port scanning*, *service enumeration*, atau *banner grabbing* untuk memperoleh data teknis yang lebih spesifik, namun metode ini berisiko lebih tinggi untuk terdeteksi oleh sistem keamanan target.

Teknik *information gathering* secara tradisional terdiri atas tiga tahap utama, yaitu *footprinting*, *scanning*, dan *enumeration*. Pada tahap *footprinting*, peneliti mengumpulkan informasi dasar seperti alamat IP, nama domain, sistem operasi, serta teknologi yang digunakan. Tahap *scanning* bertujuan untuk menemukan *host* aktif dan *port* terbuka, sementara *enumeration* melibatkan eksplorasi lebih dalam terhadap layanan, pengguna, atau konfigurasi sistem yang dapat dimanfaatkan dalam tahap berikutnya. Seiring berkembangnya teknologi, berbagai alat bantu seperti Nmap, Wireshark, The Harvester, Netcraft, dan Metagoofil banyak digunakan untuk mendukung proses pengumpulan informasi ini. Studi Verma dkk. (2021) juga menyoroti pentingnya kombinasi antara OSINT dan pemindaian aktif untuk meningkatkan efektivitas deteksi potensi risiko, meskipun penggunaan metode ini harus dibatasi dalam ruang lingkup yang legal dan etis.

Selain aspek teknis, penelitian terbaru menekankan pentingnya aspek etika dan hukum dalam kegiatan *information gathering*. Pengumpulan data secara berlebihan atau tanpa izin dapat melanggar privasi individu maupun regulasi keamanan informasi. Oleh karena itu, para peneliti dianjurkan untuk melakukan kegiatan ini dalam lingkungan laboratorium yang terisolasi, dengan batasan yang jelas dan persetujuan dari pihak terkait. Di sisi lain, hasil pengumpulan informasi juga harus dikelola dengan prinsip *responsible disclosure*, yaitu melaporkan temuan yang berpotensi sensitif kepada pihak yang berwenang tanpa menyebarkannya secara publik. Dengan demikian, *information gathering* tidak hanya berfungsi sebagai tahapan teknis untuk mendukung pengujian keamanan, tetapi juga sebagai fondasi penting dalam membangun kesadaran, kebijakan, dan strategi pertahanan siber yang lebih komprehensif.

## II.2 *Social Engineering*

*Social engineering* adalah teknik manipulasi psikologis yang mengeksloitasi kelemahan manusia, bukan kerentanan teknis pada sistem keamanan. Dalam lanskap siber yang terus berkembang, *social engineering* menjadi salah satu ancaman paling signifikan dan efektif. Sejak tahun 2021, serangan *social engineering* telah meningkat baik dari segi volume maupun kecanggihan, dengan penjahat siber dan kelompok terorganisir mengeksloitasi bias kognitif dan emosi manusia untuk menipu. Ini menjadikan faktor manusia sebagai mata rantai terlemah dalam keamanan siber.

Penelitian terbaru mengidentifikasi berbagai metode serangan *social engineering*,

mulai dari yang sederhana hingga yang sangat canggih. *Phishing*, *spear phishing*, dan *whaling* adalah serangan yang menggunakan email atau pesan palsu yang disesuaikan untuk menipu individu atau target tingkat tinggi. Serangan *phishing* di media sosial juga dapat menjangkau audiens yang lebih luas daripada email konvensional. Selain itu, *pretexting* adalah ketika penyerang menciptakan skenario palsu untuk mendapatkan informasi sensitif atau akses, sering kali dengan menyamar sebagai rekan kerja atau figur otoritas. Ada juga metode *baiting* yang menggunakan umpan (seperti file berbahaya) atau manipulasi suara untuk memancing korban agar mengambil tindakan yang membahayakan. Penyerang juga dapat melakukan *impersonation*, yaitu menyamar sebagai individu yang dikenal atau dipercaya, sebuah taktik yang lebih mudah dilakukan di media sosial karena melimpahnya informasi korban. Untuk meningkatkan presisi dan skalabilitas, penjahat siber kini juga memanfaatkan teknologi canggih seperti kecerdasan buatan (AI) dan *deepfake* untuk membuat serangan *social engineering* lebih meyakinkan.

Berbagai studi telah mengidentifikasi beberapa faktor yang membuat individu rentan terhadap serangan *social engineering*. Kesadaran keamanan yang rendah adalah salah satu faktor utama. Penyerang juga mengeksplorasi emosi seperti ketakutan, urgensi, rasa ingin tahu, dan kepercayaan untuk mendorong korban membuat keputusan yang salah. Kepercayaan berlebihan pada figur otoritas juga membuat korban lebih mudah dimanipulasi. Serangan-serangan ini memiliki dampak yang signifikan, termasuk kerugian finansial, kerusakan reputasi, dan hilangnya data. Studi kasus skema penipuan keuangan yang menargetkan Google dan Facebook menunjukkan bahwa organisasi dengan sistem keamanan yang kuat pun tidak kebal terhadap *social engineering*.

### **II.3 Cyber Security**

*Cyber security* merupakan disiplin ilmu dan praktik yang berfokus pada perlindungan sistem komputer, jaringan, data, serta perangkat digital dari berbagai ancaman yang dapat mengganggu kerahasiaan, integritas, dan ketersediaan informasi. Menurut Ainslie dkk. (2023), keamanan siber tidak hanya menjadi isu teknis, melainkan juga tantangan strategis yang berpengaruh terhadap pengambilan keputusan di tingkat organisasi. Dalam konteks modern, setiap keputusan bisnis harus mempertimbangkan potensi risiko siber, karena ancaman digital kini dapat berdampak langsung pada keberlanjutan operasional dan reputasi perusahaan. Oleh karena itu, *cyber security* mencakup kombinasi aspek teknologi, manusia, dan kebijakan organisasi yang bekerja secara terpadu untuk mencegah, mendeteksi, dan merespons

insiden keamanan secara efektif.

Komponen utama dalam *cyber security* meliputi perlindungan data dan privasi, pengelolaan risiko, deteksi serta respons terhadap insiden, hingga penerapan *Cyber Threat Intelligence (CTI)* yang berfungsi untuk mengidentifikasi pola serangan dan memberikan wawasan bagi pengambilan keputusan keamanan Ainslie dkk. (2023). Selain itu, munculnya ancaman baru seperti fileless malware turut memperluas ruang lingkup keamanan siber. Berdasarkan penelitian Sudhakar dan Kumar (2020), *fileless malware* beroperasi langsung di memori tanpa menyimpan file berbahaya di sistem, sehingga sulit dideteksi oleh *antivirus* tradisional yang berbasis tanda tangan. Ancaman ini menunjukkan bahwa sistem pertahanan harus bergeser dari deteksi berbasis file menuju pendekatan berbasis perilaku dan analisis memori.

Dalam penerapannya, pendekatan holistik menjadi kunci keberhasilan manajemen keamanan siber. FLECO, sebuah kerangka kerja yang dikembangkan oleh Domínguez-Dorado dkk. (2024), menekankan pentingnya integrasi antara aspek teknologi, tata kelola, dan budaya organisasi untuk membangun sistem keamanan yang berkelanjutan. Pendekatan ini membantu organisasi dalam mengukur kesiapan keamanan siber dan memperkuat koordinasi lintas departemen agar setiap unit memahami tanggung jawabnya dalam menjaga keamanan digital. Dengan demikian, *cyber security* tidak hanya berfungsi untuk merespons ancaman yang terjadi, tetapi juga sebagai strategi proaktif yang melibatkan seluruh komponen organisasi dalam menciptakan ketahanan siber yang adaptif dan menyeluruh.

#### **II.4 Anti-Spyware**

Sistem *anti-spyware* modern menghadapi tantangan signifikan dalam lanskap keamanan siber yang terus berevolusi, beralih dari malware tradisional menuju serangan stealth yang canggih. Menurut Sudhakar dan Kumar (2020), fileless malware beroperasi langsung di memori, menjadikannya sulit dideteksi oleh antivirus tradisional yang berbasis tanda tangan. Spyware merupakan ancaman yang beroperasi tersembunyi, dirancang untuk memantau dan mengumpulkan informasi pengguna secara rahasia, dan dikategorikan sebagai ancaman cyber espionage. Ancaman ini terus berkembang: vektor serangan meluas hingga menyasar fitur modern seperti asisten suara smartphone dan spyware disebarluaskan melalui injeksi pada aplikasi palsu. Selanjutnya, Ainslie dkk. (2023) menekankan bahwa keamanan siber bukan hanya isu teknis, tetapi tantangan strategis yang memengaruhi pengambilan keputusan di tingkat organisasi, sehingga pentingnya *Cyber Threat Intelligence (CTI)* semakin

krusial.

Kelemahan deteksi pada sistem *anti-spyware* timbul dari metode evasi canggih yang memanfaatkan celah arsitektur keamanan. E. dkk. (2023) menjelaskan bahwa spyware menggunakan teknik *packing* dan *obfuscation* untuk mengubah tanda tangan digitalnya, secara efektif menghindari deteksi berbasis tanda tangan. Bahkan, Koutsokostas dan Patsakis (2021) menunjukkan bahwa malware dapat memanfaatkan *obfuscation bytecode* Python karena kegagalan alat keamanan dalam memprosesnya. Lebih lanjut, ancaman *fileless* mengandalkan skrip bawaan sistem operasi, terutama PowerShell, untuk evasi, karena eksekusi langsung di memori (*in-memory*) menghindari deteksi berbasis file tradisional. Kareem (2024) menggarisbawahi bahwa spyware tingkat lanjut menunjukkan kapabilitas *zero-click* dan memerlukan mekanisme deteksi yang jauh lebih kompleks.

Mengingat kompleksitas ancaman yang ada, strategi *anti-spyware* harus beralih dari deteksi pasif menuju pendekatan yang lebih proaktif dan holistik. Dalam konteks pengembangan alat offensive security, Kerkour (2021) memilih bahasa pemrograman modern seperti Rust sebagai pilihan unggulan karena unggul dalam menciptakan tool yang tangguh dan sulit dilacak. Koutsokostas dan Patsakis (2021) serta Elghaly dan M. (2024) sama-sama menekankan bahwa inti dari penguatan *anti-spyware* adalah adopsi pendekatan deteksi berbasis perilaku (*behavior-based*) untuk mengenali anomali aktivitas sistem, alih-alih hanya menandai *file*. Selain itu, Domínguez-Dorado dkk. (2024) menyoroti bahwa manajemen keamanan siber harus berfokus pada integrasi aspek teknologi, tata kelola, dan budaya organisasi, didukung oleh kerangka kerja holistik seperti FLECO.

#### **II.4.1 *Endpoint Detection and Response (EDR)***

*Endpoint Detection and Response (EDR)* merupakan solusi keamanan siber yang sangat penting dalam strategi pertahanan modern, dirancang untuk mengatasi kelembahan *anti-malware* konvensional yang terlalu mengandalkan deteksi berbasis tanda tangan (*signature-based*). EDR berfokus pada pendekatan deteksi berbasis perilaku (*behavior-based*) dan analisis aktivitas, menjadikannya garis pertahanan krusial terhadap ancaman *stealth* dan *fileless malware*. Kebutuhan akan EDR meningkat karena *fileless malware* beroperasi langsung di memori (*in-memory*), sering memanfaatkan *script* bawaan sistem operasi seperti PowerShell untuk menjalankan kode berbahaya, yang secara efektif menghindari deteksi berbasis *file*. E. dkk. (2023) menguatkan bahwa EDR harus mengatasi teknik *obfuscation* dan *packing* *spyware* yang efektif melawan mesin *anti-malware* lama. EDR menjadi vital da-

lam menghadapi ancaman canggih seperti *spyware* dengan kapabilitas *zero-click*, dan Ainslie dkk. (2023) menekankan bahwa EDR merupakan komponen kunci dalam pengambilan keputusan keamanan strategis. Oleh karena itu, EDR berfungsi untuk melengkapi *anti-malware* tradisional, menyediakan kemampuan analisis perilaku dan forensik, serta merupakan komponen inti dalam penguatan ketahanan siber organisasi secara keseluruhan.

#### **II.4.2 *Signature-Based Anti-Virus***

Deteksi berbasis tanda tangan (*signature-based*) merupakan metode dasar dan tradisional yang digunakan oleh sebagian besar produk *anti-virus* (*AV*) sejak awal kemunculannya. Metode ini bekerja dengan membandingkan *hash* kriptografi atau pola urutan byte unik (disebut sebagai tanda tangan atau *signature*) dari *file* yang *discan* dengan basis data ekstensif yang berisi *signature malware* yang sudah dikenal. Jika terjadi kecocokan (*match*), *file* tersebut diklasifikasikan sebagai *malware* dan akan dikarantina atau dihapus. Meskipun metode ini sangat cepat dan memiliki akurasi 100% dalam mendeteksi *malware* yang *signature*-nya telah terdaftar, kelemahan mendasarnya adalah sifatnya yang reaktif; *anti-virus* harus memiliki *signature* terlebih dahulu, yang berarti metode ini tidak efektif terhadap ancaman baru (*zero-day threats*). Lebih lanjut, *signature-based detection* mudah dihindari oleh *malware* modern melalui teknik penyamaran seperti enkripsi, *packing*, dan *obfuscation*, yang mengubah *signature file* tanpa mengubah fungsionalitas intinya. Kegagalan ini memaksa sistem keamanan untuk bergeser menuju pendekatan yang lebih proaktif, seperti analisis perilaku dan pembelajaran mesin.

#### **II.4.3 *Behavior-Based Anti-Virus***

Deteksi berbasis perilaku (*behavior-based*) adalah pendekatan yang lebih proaktif dan modern, dikembangkan untuk mengatasi kelemahan utama metode berbasis tanda tangan (*signature-based*). Metode ini tidak bergantung pada *signature* yang sudah dikenal, melainkan memantau dan menganalisis tindakan atau pola perilaku yang mencurigakan yang dilakukan oleh suatu program saat *runtime*. Program keamanan akan memonitor serangkaian aktivitas sistem, seperti upaya untuk memodifikasi *registry sistem*, mencoba mengakses dan mengenkripsi *file* sensitif, atau meluncurkan proses sistem yang sah (misalnya, PowerShell atau WMI) dengan parameter yang tidak biasa. Jika suatu program menunjukkan urutan tindakan yang menyerupai *malware* (seperti *fileless execution* atau *persistence*), program tersebut akan ditandai atau dihentikan. Meskipun deteksi berbasis perilaku efektif dalam mengidentifikasi *malware* baru dan *fileless malware*, metode ini juga me-

miliki tantangan. *Malware* canggih sering kali dirancang untuk meniru perilaku proses yang sah (*living-off-the-land*) atau menunda eksekusi berbahaya, sehingga *behavior-based detection* rentan terhadap *false positives* (program sah yang salah dideteksi) atau *evasion* yang kompleks. Oleh karena itu, pendekatan ini sering diintegrasikan dengan *machine learning* dan analisis *in-memory* untuk meningkatkan akurasi dan mengurangi *false positives*.

#### **II.4.4 Entropy-Based Anti-Virus**

*Entropy* adalah konsep yang digunakan dalam teori informasi untuk mengukur tingkat keacakan atau ketidakpastian data yang terkandung di dalam sebuah *file* atau segmen kode. Nilai *entropy* biasanya berkisar dari 0 hingga 8; di mana nilai yang mendekati 8 menunjukkan data yang sangat acak dan tidak terstruktur, mendekati *noise* murni. Nilai *entropy* yang tinggi ini merupakan indikator penting yang digunakan oleh *anti-spyware* dan alat analisis statis (*static analysis*) untuk menduga adanya teknik *evasion* yang canggih, terutama *packing* dan enkripsi. Ketika *payload malware* dienkripsi atau dikompresi oleh *packer*, data asli yang terstruktur diubah menjadi data yang tampak acak. Oleh karena itu, *anti-spyware* menggunakan ambang batas *entropy* yang tinggi (misalnya, di atas 7.0) sebagai bendera merah (*red flag*) untuk menandai *file* sebagai mencurigakan (*suspicious*), yang mengindikasikan bahwa sebagian besar isi file tersebut tidak dapat dibaca atau dianalisis secara statis. Meskipun *entropy* tidak dapat memastikan secara pasti bahwa *file* tersebut adalah *spyware*, ia sangat efektif dalam mengidentifikasi adanya upaya penyembunyian (*concealment*) yang merupakan karakteristik utama dari *Packer Spyware Mode Stealth*.

### **II.5 Malware**

*Malware* merupakan salah satu ancaman utama dalam dunia keamanan siber yang terus berevolusi dari generasi ke generasi. Secara umum, *malware* adalah perangkat lunak berbahaya yang dirancang untuk menyusup, merusak, atau mencuri data dari sistem komputer tanpa sepengetahuan pengguna. Sudhakar dan Kumar (2020) menjelaskan bahwa evolusi malware telah bergeser dari bentuk tradisional berbasis *file* menuju *fileless malware* yang beroperasi sepenuhnya di memori. Jenis *malware* ini tidak meninggalkan jejak *file* di sistem, sehingga sulit dideteksi oleh antivirus berbasis tanda tangan. *Fileless malware* sering memanfaatkan komponen sah dari sistem operasi, seperti *Windows Management Instrumentation (WMI)* dan *PowerShell*, untuk melancarkan serangan tanpa menulis *file* berbahaya ke *disk*. Teknik

ini memungkinkan pelaku untuk melakukan aksi seperti *reconnaissance*, pencurian data, dan persistensi tanpa terdeteksi oleh solusi keamanan konvensional.

E. dkk. (2023) menambahkan bahwa dalam “permainan kucing dan tikus” antara pembuat *malware* dan pembuat *antivirus*, berbagai teknik penghindaran deteksi terus berkembang. Teknik-teknik seperti *code obfuscation*, *polymorphism*, *packing*, dan *process injection* digunakan untuk mengubah struktur dan perilaku kode agar tidak mudah dikenali. Studi mereka menunjukkan bahwa dari 16 sampel *malware* yang diujikan dengan tujuh teknik penghindaran klasik, hanya sebagian kecil *antivirus* yang mampu mendeteksi lebih dari separuh variasi *malware* tersebut. Hal ini menegaskan bahwa bahkan *malware* “lama” dengan trik penyamaran baru masih mampu menembus sistem deteksi modern. Lebih jauh lagi, peneliti juga menemukan bahwa penggunaan model pembelajaran mesin (ML) untuk deteksi *malware* masih rentan terhadap serangan *adversarial*, di mana pembuat *malware* dapat menghasilkan varian baru yang tampak seperti program sah.

Koutsokostas dan Patsakis (2021) berfokus pada pengembangan *malware stealth* berbasis Python yang mampu menghindari deteksi tanpa menggunakan *obfuscation*. Mereka menemukan bahwa keterbatasan pada mesin deteksi statis, seperti VirusTotal dan sandbox analisis dinamis, dapat dimanfaatkan untuk menciptakan *malware* yang “bersih” dari hasil pemindaian puluhan antivirus. Studi tersebut mengungkapkan bahwa PyInstaller—alat populer untuk membungkus program Python menjadi *executable* dapat dimodifikasi agar menghasilkan *malware* yang tidak terdeteksi karena kelemahan inheren dalam cara antivirus memproses bytecode Python. Selain itu, mereka menemukan bahwa sandbox publik sering kali gagal mendeteksi *malware* yang menunda eksekusi, mendeteksi lingkungan virtual, atau memeriksa artefak sistem sebelum beraksi.

Berdasarkan literatur tersebut, tren utama dalam penelitian *malware* modern menyebutkan bahwa ancaman kini tidak hanya berasal dari varian baru, tetapi dari kemampuan *malware* untuk beradaptasi terhadap mekanisme pertahanan yang ada. Dengan kombinasi teknik *living-off-the-land*, penghindaran berbasis memori, dan eksplorasi terhadap celah dalam sistem analisis otomatis, *malware* modern semakin sulit diidentifikasi. Oleh karena itu, studi-studi ini menegaskan perlunya pendekatan deteksi berbasis perilaku dan kecerdasan buatan yang mampu mengenali pola aktivitas abnormal alih-alih bergantung semata pada tanda tangan statis. Dengan demikian, penelitian mengenai *malware* tidak hanya penting untuk memahami sifat serangan, tetapi juga menjadi dasar dalam merancang sistem pertahanan yang lebih adaptif

dan tangguh terhadap ancaman siber generasi baru.

### **II.5.1 *Fileless Malware***

*Fileless execution* adalah teknik serangan siber canggih di mana kode berbahaya dijalankan secara langsung di dalam memori sistem (RAM) tanpa perlu menulis atau menyimpan *file* yang dapat dideteksi ke *disk (hard drive)*. Metode ini sering dimanfaatkan oleh *fileless malware* atau *spyware* untuk menghindari deteksi berbasis tanda tangan (*signature-based*) dan forensik digital tradisional yang berfokus pada analisis *file system*. *Fileless execution* umumnya dicapai dengan mengeksplorasi alat (*utility*) atau fitur bawaan sistem operasi yang sah, seperti PowerShell, *Windows Management Instrumentation (WMI)*, atau dengan menyuntikkan kode langsung ke proses yang sudah ada dan terpercaya (*process injection*). Karena tidak ada *file* berbahaya yang disimpan, serangan ini sangat sulit dilacak dan dideteksi oleh *anti-virus* konvensional, memaksa sistem keamanan untuk beralih ke deteksi berbasis perilaku dan analisis memori.

### **II.5.2 *Obfuscation pada Malware***

*Obfuscation* adalah teknik penyembunyian (*concealment*) yang digunakan untuk memanipulasi kode *malware* atau *payload* agar menjadi sulit untuk dipahami, dinilai, atau dideteksi oleh alat keamanan statis maupun dinamis. Menurut E. dkk. (2023) *code obfuscation* adalah teknik klasik yang terbukti masih sangat efektif untuk menghindari deteksi *anti-virus* modern. Tujuan utamanya adalah untuk mengubah tanda tangan (*signature*) kode, *string*, dan alur program, sehingga *anti-virus* berbasis *signature* kesulitan mencocokkannya dengan basis data yang ada.

Dalam implementasinya, *obfuscation* sering dilakukan melalui berbagai metode. Misalnya, dalam pengembangan *malware* berbasis script seperti Python, *obfuscation* dapat terjadi ketika kode sumber dikemas menjadi bentuk *bytecode* terkompilasi menggunakan alat seperti PyInstaller (banyak *anti-virus* gagal menganalisis *bytecode* ini, sehingga menghasilkan *false negative*). Selain itu, *obfuscation* diterapkan pada *Dropper* untuk menyamaraskan skrip yang dieksekusi melalui *utility* sistem seperti PowerShell yang secara langsung menargetkan kelemahan *script monitoring* pada EDR dan AV. Keseksian *obfuscation* memaksa peneliti dan anti-malware untuk beralih dari analisis statis ke analisis perilaku (*behavior-based detection*) dan teknik *deobfuscation* yang mahal.

## II.6 *Spyware*

*Spyware* merupakan salah satu bentuk *malware* yang dirancang untuk memantau, mengumpulkan, dan mengirimkan informasi pengguna tanpa izin atau kesadaran mereka. Perangkat lunak ini biasanya berjalan secara tersembunyi di latar belakang sistem dan dapat merekam aktivitas pengguna, seperti penekanan tombol (*keylogging*), riwayat peramban, data *login*, serta *file* sensitif. Dalam literatur keamanan siber, *spyware* sering dikategorikan sebagai ancaman yang bersifat *stealth*, karena kemampuannya untuk beroperasi tanpa menimbulkan indikasi mencolok bagi pengguna maupun sistem keamanan. Menurut Koutsokostas dan Patsakis (2021), kemampuan *stealth* seperti ini muncul karena *malware* modern, termasuk *spyware*, memanfaatkan teknik penghindaran analisis dan deteksi baik dalam bentuk statis maupun dinamis. Mereka menunjukkan bahwa banyak *antivirus* gagal mengenali kode berbahaya yang dikemas menggunakan alat seperti PyInstaller karena keterbatasan dalam menganalisis *bytecode Python*. Hal ini menyebabkan sebagian besar *multi-engine scanner*, termasuk VirusTotal, dapat memberikan hasil “bersih” terhadap *file* yang sebenarnya mengandung komponen *spyware*.

E. dkk. (2023) dalam studi “*Bypassing Antivirus Detection: Old-school Malware, New Tricks*” menguatkan temuan tersebut dengan menyoroti bagaimana teknik klasik seperti *code obfuscation*, *packing*, dan *process injection* masih sangat efektif untuk menghindari deteksi *antivirus* modern. Mereka menguji berbagai varian *malware*, termasuk *spyware*, terhadap beberapa produk *antivirus* dan menemukan bahwa sebagian besar sistem deteksi hanya mampu mengenali sebagian kecil varian yang dimodifikasi. Temuan ini menunjukkan bahwa banyak mesin *antivirus* masih mengandalkan pencocokan tanda tangan (*signature-based detection*), yang tidak mampu mengidentifikasi pola perilaku baru dari *spyware* yang berevolusi. Selain itu, metode penghindaran berbasis lingkungan—seperti deteksi *sandbox* atau penundaan eksekusi (*delayed execution*) membuat *spyware* semakin sulit teridentifikasi melalui analisis dinamis tradisional.

Dari sisi karakteristik perilaku, *spyware* modern cenderung memanfaatkan teknik *living-off-the-land*, yaitu memanfaatkan fungsi atau layanan sah dari sistem operasi seperti PowerShell, WMI, atau API Windows untuk melaksanakan aksinya tanpa mengunduh *file* berbahaya tambahan. Pendekatan ini menjadikan *spyware* semakin sulit dilacak, karena aktivitasnya tampak seperti proses sistem yang normal. Koutsokostas dan Patsakis (2021) menegaskan bahwa pola serangan semacam ini tidak hanya menunjukkan kelemahan sistem deteksi *antivirus*, tetapi juga menyoroti per-

lunya pendekatan baru berbasis perilaku (*behavior-based detection*) yang mampu mengenali anomali aktivitas sistem, bukan sekadar menandai *file* berbahaya.

Secara keseluruhan, *spyware* merupakan evolusi dari malware tradisional menuju ancaman yang lebih canggih, tersembunyi, dan adaptif. Dengan kemampuan memanfaatkan celah pada sistem deteksi statis maupun dinamis, *spyware* modern menjadi tantangan utama dalam bidang keamanan siber. Oleh karena itu, penelitian dan pengembangan sistem pertahanan di masa depan perlu berfokus pada integrasi antara analisis perilaku, pembelajaran mesin, dan *threat intelligence* untuk mendeteksi aktivitas mencurigakan secara proaktif. Pendekatan ini diharapkan dapat menutup celah yang selama ini dimanfaatkan oleh *spyware* untuk beroperasi tanpa terdeteksi di berbagai *platform*, baik *desktop* maupun *mobile*.

### **II.6.1 *Spyware Mode Stealth***

*Spyware mode stealth* merujuk pada evolusi ancaman *spyware* yang secara khusus dirancang untuk beroperasi secara tersembunyi dan menghindari deteksi sistem keamanan siber. Ancaman ini dikategorikan dalam lingkup yang lebih luas yaitu *cyber espionage* dan merupakan evolusi dari malware tradisional.

Kemampuan *spyware* untuk beroperasi secara *stealth* sangat bergantung pada eksploitasi kelemahan dalam mekanisme deteksi. E. dkk. (2023) menjelaskan bahwa *spyware* secara aktif menggunakan teknik *packing* dan *obfuscation* untuk mengubah tanda tangan digitalnya, efektif menghindari deteksi berbasis tanda tangan. Sudhakar dan Kumar (2020) dan Elghaly dan M. (2024) sama-sama menyoroti bahwa ancaman *fileless* merupakan mekanisme *stealth* kunci, di mana *spyware* beroperasi langsung di memori, sering memanfaatkan PowerShell untuk menjalankan kode berbahaya, menghindari deteksi berbasis *file*. Koutsokostas dan Patsakis (2021) menambahkan bahwa kegagalan alat keamanan dalam memproses *bytecode* Python juga dieksplorasi untuk menyamarkan *script* berbahaya. EDR menghadapi tantangan besar karena harus memantau proses sistem yang sah ini untuk mengidentifikasi aktivitas mencurigakan.

### **II.6.2 *Packer Spyware Mode Stealth***

*Packer* merupakan alat atau teknik perangkat lunak yang berfungsi sebagai lapisan perlindungan awal dengan mengemas (*wrap*) *payload malware* agar menjadi sulit dideteksi dan dianalisis oleh sistem keamanan seperti *anti-virus (AV)* maupun EDR (*Endpoint Detection & Response*). Tujuan utama *packer* adalah mencapai *evasion*

dengan mengubah *signature file* sebelum eksekusi. *Packer* mencapai tujuan ini dengan melakukan beberapa proses, termasuk kompresi (*compression*) untuk mengurangi ukuran *file*, enkripsi untuk mengacak kode dan data, serta *obfuscation* untuk menyamarkan struktur kode agar tidak mudah dicocokkan dengan basis data *signature AV*. Keberhasilan *packer* dalam mengubah *signature file* secara efektif menjadikannya taktik utama untuk lolos pada tahap pemeriksaan statis (*pre-execution*).

Dalam konteks *spyware mode stealth*, *packer* sering diimplementasikan sebagai *Loader single-stage* yang bertanggung jawab untuk menjalankan *runtime unpacking*. Pada *runtime* di sistem target, sebuah stub kecil di dalam *file* yang sudah di-*pack* akan mendekode atau mendekripsi *payload* asli secara langsung di memori (RAM). Dengan melakukan proses *unpacking* dan eksekusi di memori tanpa menulis *payload* yang sudah didekripsi ke *disk*, *packer* mendukung teknik *fileless execution*. Hal ini secara signifikan meningkatkan stealth karena menghindari analisis *file system* dan *signature-based detection*. Secara keseluruhan, *Packer Spyware Mode Stealth* bertujuan mengeksplorasi celah *anti-malware* ganda: pertama, dengan memanipulasi *signature file*, dan kedua, dengan menjalankan kode berbahaya hanya di memori (*in-memory*) melalui alat sistem yang sah seperti PowerShell, menjadikannya sangat sulit dilacak oleh *anti-virus* konvensional.

### II.6.3 *Payload* pada *Packer Spyware*

*Payload* dalam konteks *malware* merujuk pada komponen inti yang membawa fungsi utama dari serangan. Secara umum, *payload* merupakan bagian yang mengeksekusi aksi berbahaya atau mencapai tujuan operasional setelah proses infiltrasi berhasil. Namun, pada kasus tertentu—seperti *packer-based spyware*—*payload* tidak selalu berupa komponen yang destruktif atau melakukan aksi spionase penuh.

Dalam penelitian ini, *payload* pada *packer spyware* berfungsi sebagai muatan awal (*initial-stage payload*) yang sangat minimal, yaitu hanya menjalankan instruksi untuk mengumpulkan informasi dasar sistem (*system information*) dan mengirimkannya ke server pengendali. *Payload* jenis ini sering disebut sebagai *lightweight reconnaissance payload*, karena tujuannya bukan melakukan spionase mendalam, melainkan menyediakan konteks sistem untuk menentukan langkah serangan berikutnya—misalnya pemilihan *dropper* yang sesuai.

Dengan demikian, meskipun *payload* pada *packer spyware* dalam penelitian ini tidak memiliki fungsi spionase kompleks seperti *data collection* lanjutan atau *exfiltration*, ia tetap berperan penting sebagai komponen yang menginisiasi tahap awal se-

rangan dengan mengirimkan *system information* sebagai dasar bagi server/C2 untuk menentukan *second-stage payload* (dropper). Hal ini menunjukkan bahwa bahkan *lightweight payload* sekalipun dapat menjadi bagian strategis dalam arsitektur multi-tahap (*multi-stage malware architecture*).

## **II.7 Studi Sebelumnya**

Studi sebelumnya diperlukan untuk memahami perkembangan penelitian terkait mekanisme penyembunyian data (*packer*), teknik eksekusi tersembunyi (*stealth*), serta berbagai metode evasi dan deteksi yang digunakan dalam keamanan siber. Kajian ini memberikan gambaran mengenai pendekatan, metode, serta keterbatasan penelitian terdahulu yang menjadi dasar dalam merumuskan ruang lingkup dan kontribusi penelitian ini.

Tabel II.1 Studi Sebelumnya (*Packer Spyware* dan Mekanisme Evasi)

No	Judul Penelitian (Penulis)	Metode	Hasil Utama	Keterbatasan
1	<i>Python and Malware: Developing Stealth and Evasive Malware Without Obfuscation</i> (Koutsokostas dan Patsakis (2021))	Perancangan sistem dan eksperimen	Mengembangkan <i>malware Python stealth</i> yang menghindari deteksi dengan meng-exploitasi kelemahan <i>multi-engine scanners</i> dalam memproses <i>bytecode Python</i> , menunjukkan efektivitas <i>packer stealth</i> .	Fokus utama pada evasi <i>static analysis bytecode Python</i> ; kurang membahas teknik <i>in-memory behavioral evasion</i> yang lebih luas.
2	<i>Bypassing Antivirus Detection: Old-School Malware, New Tricks</i> (E. dkk. (2023))	Eksperimen komparatif dan analisis teknis	Menunjukkan bahwa teknik lama seperti <i>packing</i> dan <i>obfuscation</i> tetap efektif melawan AV/EDR modern; hampir se-paruh mesin yang diuji gagal mendeteksi varian <i>malware</i> yang disamarkan.	Fokus pada kerentanan metode <i>signature-based</i> umum; tidak mengusulkan solusi arsitektur deteksi baru.
3	<i>Stealth in Plain Sight: The Hidden Threat of PowerShell Fileless Malware...</i> (Elghaly dan M. (2024))	Analisis teknis dan eksperimen bypass	Mengkaji bagaimana <i>malware fileless</i> memanfaatkan <i>PowerShell</i> untuk eksekusi kode langsung di memori, secara efektif menghindari EDR dan AV modern.	Fokus pada lingkungan <i>PowerShell</i> ; kurang membahas mekanisme <i>packer kustom</i> dan <i>bytecode evasion</i> .
4	<i>An Emerging Threat: Fileless Malware – A Survey and Research Challenges</i> (Sudhakar dan Kumar (2020))	Studi literatur	Menjelaskan teknik penyembunyian dan eksfiltrasi <i>fileless</i> yang dieksplorasi oleh <i>spyware</i> , termasuk penggunaan <i>PowerShell</i> , <i>registry abuse</i> , dan <i>memory-resident payload</i> .	Bersifat survei dan komprehensif; tidak mencakup implementasi uji coba <i>packer kustom</i> atau pengukuran kapabilitas deteksi.

## BAB III

### ANALISIS MASALAH

#### III.1 Analisis Kondisi Saat Ini

Kondisi keamanan siber saat ini ditandai dengan evolusi ancaman yang signifikan, bergerak dari *malware* tradisional berbasis *file* menuju serangan yang lebih canggih dan tersembunyi, seperti *fileless malware* dan *spyware mode stealth*. Evolusi ini menciptakan celah kritis dalam kemampuan deteksi sistem *anti-spyware (AS)* dan *Endpoint Detection and Response (EDR)* yang masih berpegangan pada mekanisme pertahanan konvensional.

Secara konseptual, serangan *spyware* modern yang berfokus pada *Initial Access* dan *Packer* melibatkan beberapa komponen utama: Target (sistem yang diserang), *Packer/Dropper* (Artefak *spyware* yang disamarkan), *Anti-Malware/EDR* (Sistem pertahanan), dan *Server* (Pengumpul informasi, seperti yang digambarkan dalam alur *General Flow*).

##### III.1.1 Masalah Kerentanan dan Ketertinggalan *Anti-Malware* Konvensional

Kondisi keamanan siber saat ini ditandai dengan evolusi ancaman yang signifikan, bergerak dari *malware* tradisional berbasis *file* menuju serangan yang lebih canggih dan tersembunyi, seperti *fileless malware* dan *spyware mode stealth*. Evolusi ini menciptakan celah kritis dalam kemampuan deteksi sistem *anti-malware (AV)* dan *Endpoint Detection and Response (EDR)* yang masih berpegangan pada mekanisme pertahanan konvensional.

##### III.1.2 Keterbatasan Deteksi Berbasis Tanda Tangan (*Signature-Based*)

*Anti-malware* tradisional masih sangat mengandalkan pencocokan tanda tangan (*signature-based detection*). *Spyware mode stealth* menggunakan teknik *Packer* (seperti yang dijelaskan dalam konsep arsitektur serangan) untuk melakukan kompresi, enkripsi,

dan *obfuscation* pada *payload*. Teknik ini secara efektif mengubah tanda tangan digital (*signature*) *spyware*, sehingga membuatnya lolos dari deteksi *signature-based* karena dianggap sebagai *file* baru atau tidak dikenal.

### **III.1.3 Gagal Menganalisis Code *Fileless Execution***

Situasi ini diperparah dengan temuan bahwa banyak alat keamanan gagal dapat mendeteksi *fileless malware*. *Fileless malware* memanfaatkan komponen sah dari sistem operasi (seperti PowerShell dan WMI) untuk menjalankan kode berbahaya langsung di memori tanpa menulis *file* ke disk, sehingga sulit dideteksi oleh *antivirus* konvensional.

### **III.1.4 Efektivitas Teknik Evasi Sederhana**

Meskipun serangan semakin canggih, penelitian menunjukkan bahwa metode evasi yang relatif sederhana, seperti enkripsi, injeksi proses, dan penambahan data sampah (*junk data*) ke file eksekusi, terbukti sangat efektif dalam menghindari deteksi. Bahkan, dalam sebuah studi E. dkk. (2023), hampir separuh dari 12 mesin *antivirus* yang diuji hanya mampu mendeteksi kurang dari setengah varian *malware* yang disamarkan.

### **III.1.5 Kurangnya Deteksi Proaktif Terhadap Arsitektur Serangan**

Kurangnya deteksi yang efektif oleh solusi keamanan menciptakan celah besar yang dieksloitasi oleh *Advanced Persistent Threats (APTs)*. Sistem pertahanan saat ini terlalu reaktif, berfokus pada *file* yang sudah terinstal, bukan pada deteksi perilaku evasif dari *Packer* di fase *Initial Access* dan *Establish Foothold*.

### **III.1.6 Gap Analysis Celah Deteksi *Packer Spyware Mode Stealth***

Berdasarkan kondisi saat ini dan studi literatur, sebuah Analisis Kesenjangan (*Gap Analysis*) dirumuskan untuk menyoroti perbedaan antara kondisi ideal deteksi dan realitas sistem *anti-spyware* saat ini.

Tabel III.1 Kebutuhan fungsional (Functional Requirements)

<b>ID</b>	<b>Critical to Quality(CTQ)</b>	<b>Kondisi Saat ini</b>	<b>Gap</b>	<b>Kondisi Ideal</b>
CTQ-01	Deteksi <i>Packer</i> pada <i>Initial Access</i>	Deteksi berfokus pada <i>signature file</i> yang mudah di- <i>bypass</i> oleh teknik enkripsi dan <i>obfuscation</i>	Celah <i>Signature-Evasion</i>	<i>Packer</i> harus menggunakan enkripsi unik dan <i>obfuscation</i> untuk <i>payload</i> .
CTQ-02	Gagal menganalisis kode <i>payload</i> ter- <i>obfuscated</i> .	Gagal menganalisis skrip yang ter- <i>obfuscated</i>	Celah <i>Obfuscation</i>	Prototipe harus dirancang dengan bahasa yang menghasilkan <i>binary</i> resisten ( <i>low-level Rust</i> ).
CTQ-03	Akurasi Deteksi <i>Fileless Execution</i>	Deteksi rendah karena <i>payload</i> berjalan langsung di memori (via PowerShell/WMIC)	Celah <i>In-Memory Behavioral</i>	<i>Packer</i> harus mampu mengintegrasikan <i>loader in-memory (single-stage)</i> yang <i>stealth</i> .

### III.2 Analisis Kebutuhan

Berdasarkan *gap analysis* (CTQ) di atas, diperlukan suatu artefak uji yang mampu mensimulasikan serangan *spyware mode stealth* secara terkontrol. Oleh karena itu, pada bagian berikut disusun kebutuhan fungsional dan nonfungisional dari prototipe *packer spyware*.

#### III.2.1 Kebutuhan Fungsional

Kebutuhan fungsional mendefinisikan kapabilitas yang harus dimiliki oleh *Packer Spyware* untuk menguji celah keamanan (CTQ).

Tabel III.2 Kebutuhan fungsional (Functional Requirements)

<b>ID</b>	<b>Kebutuhan</b>	<b>CTQ Terkait</b>
FR-01	Prototipe harus mampu mengenkripsi <i>payload</i> dan menyembunyikan <i>signature</i> asli.	CTQ-01
FR-02	Prototipe harus mampu melakukan pengemasan ( <i>packing</i> ) dan <i>obfuscation</i> tingkat tinggi pada <i>binary</i> akhir.	CTQ-02
FR-03	Prototipe harus mampu menjalankan <i>payload</i> secara <i>fileless</i> .	CTQ-03
FR-04	Prototipe harus memiliki kemampuan <i>Initial Access</i> dan <i>Persistence</i> dasar.	CTQ-01, CTQ-03
FR-05	Prototipe harus mampu mengirimkan sinyal keberhasilan <i>Initial Access</i> ke <i>Server</i> pengumpul data.	CTQ-01

### III.2.2 Kebutuhan Nonfungsional

Kebutuhan nonfungsional berfokus pada bagaimana sistem pengujian harus bekerja untuk memastikan hasil yang valid dan andal.

Tabel III.3 Kebutuhan nonfungsional (Non-Functional Requirements)

<b>ID</b>	<b>Kebutuhan</b>	<b>CTQ Terkait</b>
NFR-01	<i>Executable</i> yang dihasilkan harus kecil dan <i>reliable</i> di Windows 11.	Keandalan & Validitas
NFR-02	Lingkungan pengujian harus mencakup setidaknya empat produk <i>anti-malware/EDR</i> komersial yang berbeda.	Validitas komparatif
NFR-03	Implementasi harus menggunakan <i>code safety (low-level Rust)</i> untuk memastikan <i>debugging</i> yang efisien.	Audibilitas

### **III.3 Analisis Pemilihan Solusi**

Setelah masalah dianalisis, langkah selanjutnya adalah melaksanakan evaluasi berbagai solusi yang dapat mewujudkan tujuan penelitian. Analisis ini mencakup penilaian pendekatan teknis dan konseptual yang paling ideal sesuai dengan kebutuhan sistem dan kriteria desain yang telah ditetapkan. Tujuan utama dari penelitian ini adalah merancang dan mengimplementasikan *Packer Spyware Mode Stealth* untuk menguji kapabilitas deteksi *anti-malware*, sehingga dapat memvalidasi teknik *evasion* dan mengungkap kelemahan sistem keamanan konvensional.

#### **III.3.1 Alternatif Solusi**

Dalam upaya mencari solusi yang paling optimal untuk mengatasi permasalahan yang teridentifikasi, subbab ini akan menguraikan berbagai konsep implementasi alternatif untuk setiap Kebutuhan Fungsional (FR) dan Kebutuhan Non Fungsional (NFR) modul *packer spyware mode stealth*. Setiap alternatif ini akan dijelaskan secara ringkas mengenai pendekatannya dan bagaimana ia berpotensi memenuhi kebutuhan yang bersangkutan.

Tabel III.4 Alternatif Solusi

<b>Kode</b>	<b>Solusi Bahasa pemrograman</b>	<b>Pendekatan Utama (<i>Evasion</i>)</b>
S-01	Rust	Bahasa sistem yang menghasilkan <i>binary</i> statis, unggul dalam kontrol memori dan <i>low-level manipulation</i> .
S-02	C/C++	Bahasa sistem tradisional yang menawarkan kontrol penuh atas API Windows dan <i>memory injection</i> .
S-03	Python (PyInstaller)	Bahasa <i>scripting</i> . Mengandalkan <i>obfuscation bytecode</i> dan <i>packer wrapper</i> .

#### **III.3.2 Rust**

Rust adalah bahasa pemrograman sistem yang akan digunakan untuk mengembangan *Packer*. Bahasa ini dipilih karena mampu menghasilkan *binary* yang sangat efisien dan terkompilasi menjadi *native code*. Kontrol tingkat rendah yang disediakan Rust memungkinkan implementasi manipulasi memori dan panggilan API Windows

secara langsung, yang sangat penting untuk teknik *fileless execution* (FR-03). *Packer* yang dibangun dengan Rust dapat mengenkripsi *payload* (FR-01) dan memiliki resistensi tinggi terhadap analisis statis (*reverse engineering*), karena minimnya jejak *runtime* yang mudah dideteksi (Potensi *Stealth*).

### III.3.3 C/C++

C/C++ adalah bahasa pemrograman sistem tradisional yang merupakan standar umum untuk pengembangan alat *low-level*. Penggunaan bahasa ini memungkinkan pengembang untuk memiliki kontrol penuh atas manajemen memori dan akses langsung ke API Windows, yang esensial untuk implementasi *memory injection* dan *fileless execution* (FR-03). *Packer* yang dikembangkan dengan C/C++ dapat mengkompilasi *payload* menjadi *binary native* yang kuat, menjadikannya alternatif yang efektif untuk menyamarkan *payload* dan melakukan enkripsi tingkat lanjut (FR-01).

### III.3.4 Python (PyInstaller)

Python adalah bahasa *scripting* tingkat tinggi yang menawarkan kemudahan dan kecepatan dalam pengembangan *payload* dan logika enkripsi sederhana (FR-01). Ketika dikombinasikan dengan alat *packer* seperti PyInstaller, *script* Python dapat dibungkus menjadi *executable* Windows. Solusi ini memanfaatkan *obfuscation bytecode* dan *packer wrapper* sebagai lapisan pertahanan awal terhadap deteksi *signature-based*. Namun, solusi ini bergantung pada mekanisme *wrapper* yang seringkali kurang stealth dibandingkan kompilasi *low-level* murni.

### III.3.5 Analisis Penentuan Solusi

Untuk memastikan solusi yang dihasilkan dapat menyelesaikan masalah yang telah diidentifikasi sebelumnya, kriteria desain ditetapkan dengan mempertimbangkan hasil identifikasi masalah dan tujuan solusi. Kriteria ini berfungsi sebagai acuan dalam merancang, mengembangkan, dan mengevaluasi solusi secara sistematis. Tabel III.5 akan menjelaskan kriteria-kriteria desain tersebut yang akan menjadi dasar dalam menilai keberhasilan artefak yang dikembangkan dalam penelitian ini.

Tabel III.5 Kriteria Desain Packer Spyware

<b>Kode</b>	<b>Kriteria Desain</b>	<b>Keterkaitan dengan Evasion &amp; Non-Functional Requirement</b>
KD-1	Dukungan Enkripsi/Packing	Kemampuan <i>native</i> bahasa dalam mengimplementasikan enkripsi <i>payload</i> dan <i>packing</i> kode sumber secara ringkas, mendukung teknik <i>code obfuscation</i> dan <i>payload concealment</i> .
KD-2	Kontrol <i>Low-Level &amp; Fileless</i>	Mendukung pemanggilan Windows API, manipulasi memori, dan eksekusi <i>fileless</i> untuk meningkatkan <i>stealth</i> serta mengurangi artefak forensik.
KD-3	Resistensi Analisis Statis	Meningkatkan kesulitan <i>reverse engineering</i> pada <i>binary</i> ; bahasa yang kuat membantu keamanan, integritas, dan <i>reliability payload</i> terhadap analisis statis.
KD-4	Potensi Stealth & Footprint	Ukuran <i>binary</i> kecil dan minim dependensi <i>runtime</i> eksternal sehingga mengurangi jejak deteksi AV/EDR pada pengujian <i>stealth</i> .
KD-5	Kompleksitas Implementasi & Safety	Effort waktu dan tingkat kesulitan implementasi (termasuk keamanan memori dan <i>debugging</i> ) berpengaruh langsung pada efisiensi pengembangan artefak.
KD-6	Kompatibilitas Sistem Target	Kemampuan menghasilkan <i>executable</i> yang stabil dan kompatibel pada Windows, memastikan <i>reliability</i> selama pengujian <i>evasion</i> .

Untuk menentukan solusi terbaik secara objektif, digunakan pendekatan *MultiCriteria Decision Analysis (MCDA)* dengan *Entropy Weight Method (EWM)*. Metode ini digunakan untuk menentukan bobot kriteria secara ilmiah dan obyektif berdasarkan tingkat keragaman informasi (diversifikasi) dari setiap kriteria (Zhu dkk. 2020). Analisis MCDA dilakukan dengan melakukan normalisasi *decision matrix*. Skor dari setiap solusi terhadap suatu kriteria dinormalisasi menggunakan metode rasio

terhadap total skor dalam kriteria tersebut, seperti yang terlihat pada Rumus III.1.

$$P_{ij} = \frac{x_{ij}}{\sum_{i=1}^m x_{ij}} \quad (\text{III.1})$$

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

Nilai entropi ( $e_j$ ) dihitung untuk setiap kriteria  $j$  guna mengukur tingkat ketidakpastian atau dispersi data:

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

Derajat diversifikasi ( $d_j$ ) diperoleh melalui:

$$d_j = 1 - e_j \quad (\text{III.3})$$

Bobot relatif ( $w_j$ ) untuk setiap kriteria kemudian dihitung dengan:

$$w_j = \frac{d_j}{\sum_{j=1}^n d_j} \quad (\text{III.4})$$

Bobot mencerminkan kontribusi setiap kriteria terhadap pengambilan keputusan, di mana kriteria dengan variabilitas tinggi mendapat bobot lebih besar.

Skor komposit akhir ( $S_i$ ) untuk setiap alternatif solusi dihitung menggunakan:

$$S_i = \sum_{j=1}^n w_j P_{ij} \quad (\text{III.5})$$

Alternatif dengan skor tertinggi dipilih sebagai solusi yang paling optimal berdasarkan pertimbangan *multi-kriteria* yang objektif.

### III.3.6 Hasil Analisis Penentuan Solusi

Perhitungan MCDA-EWM dilakukan terhadap matriks keputusan dengan enam kriteria desain dan tiga alternatif solusi. Kriteria yang dievaluasi adalah: KD-1 (Dukungan Enkripsi/*Packing*), KD-2 (Kontrol *Low-Level Fileless*), KD-3 (Resistensi Analisis Statis), KD-4 (Kemudahan Implementasi *Anti-Analysis*), KD-5 (Performa Eksekusi), dan KD-6 (*Ecosystem Tools/Libraries*). Hasil perhitungan entropi

menunjukkan bahwa KD-2 memiliki entropi terendah dan derajat diversifikasi tertinggi, mengindikasikan bahwa kriteria ini paling efektif dalam membedakan ketiga alternatif. Tabel III.6 merangkum hasil perhitungan bobot kriteria.

Tabel III.6 Bobot kriteria hasil metode EWM

<b>Kode Kriteria</b>	$e_j$	$d_j$	$w_j$
KD-1	0.9979	0.0021	0.081
KD-2	0.9609	0.0391	0.1518
KD-3	0.8716	0.1284	0.4984
KD-4	0.9641	0.0359	0.1394
KD-5	0.9489	0.0511	0.1984
KD-6	1.0000	0.0000	0.0000
<b>Total</b>		0.2576	1.000

Nilai bobot menunjukkan bahwa KD-3 memberikan kontribusi terbesar (49.84%) terhadap pengambilan keputusan, diikuti oleh KD-5 (19.84%), sedangkan kriteria lainnya memiliki kontribusi kecil.

Skor komposit akhir untuk setiap alternatif disajikan pada Tabel III.7. Rust memperoleh skor tertinggi (0.3546), diikuti oleh C/C++ (0.3235), dan Python (0.1994). Dominasi skor Rust terutama disebabkan oleh performanya pada KD-3 dan KD-5, yaitu resistensi terhadap analisis statis serta kompleksitas implementasi yang lebih aman. Kedua kriteria tersebut memiliki bobot EWM tertinggi, sehingga kontribusinya paling signifikan terhadap total skor. Rust juga menunjukkan performa baik pada KD-2 dan KD-4, yang berkaitan dengan kontrol *low-level*, kemampuan *file-less*, serta jejak *binary* yang lebih *stealth*. Dengan demikian, Rust menjadi alternatif paling optimal dalam konteks *evasion* dan kebutuhan non-fungsional yang telah didefinisikan.

Tabel III.7 Hasil Perhitungan Bobot EWM dan Skor Solusi

<b>Kriteria Desain</b>	<b>Bobot EWM</b>	<b>S-1</b>	<b>S-2</b>	<b>S-3</b>
KD-1	0.0081	0.0025	0.0028	0.0028
KD-2	0.1518	0.0621	0.0621	0.0276
KD-3	0.4984	0.2243	0.1994	0.0748
KD-4	0.1394	0.0570	0.0506	0.0316
KD-5	0.1984	0.0660	0.0440	0.0881
KD-6	0.0000	0.0000	0.0000	0.0000
<b>Total Skor</b>	<b>1.0000</b>	<b>0.3546</b>	<b>0.3235</b>	<b>0.1994</b>

Berdasarkan hasil analisis MCDA-EWM, Rust dipilih sebagai alternatif terbaik untuk implementasi *packer spyware mode stealth*. Keputusan ini didukung oleh skor komposit tertinggi serta kontribusi dominan pada kriteria dengan bobot terbesar, yaitu resistensi terhadap analisis statis (KD-3) dan kompleksitas implementasi yang aman (KD-5). Selain itu, Rust menunjukkan performa kuat pada aspek kontrol *low-level* dan kemampuan *fileless* (KD-2), yang merupakan fondasi utama dalam membangun artefak berprofil rendah dan sulit dideteksi. Konsistensi performa ini menjadikan Rust pilihan paling efektif untuk menghasilkan *packer* yang mampu meminimalkan jejak deteksi, menjaga integritas *payload*, serta memenuhi kebutuhan *evasion*.

## BAB IV

### DESAIN KONSEP SOLUSI

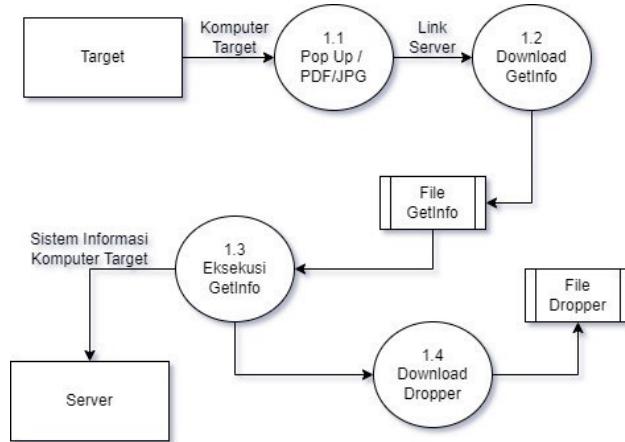
Bab ini menyajikan penjelasan rinci mengenai rancangan sistem yang diusulkan dalam penelitian, yaitu desain modul *packer* pada *spyware* dengan mode *stealth* pada fase *Initial Access* yang akan digunakan untuk menguji kapabilitas deteksi antivirus dalam lingkungan terisolasi. Sebagai pedoman metodologis, kerangka *Design Research Methodology* yang telah diuraikan pada Bab I diadaptasi dalam bab ini. Pendekatan tersebut diterapkan karena dapat memvisualisasikan alur pengujian modul *packer* pada anti-*spyware*. Oleh karena itu, seluruh proses desain dan pengembangan dalam bab ini diarahkan untuk menciptakan artefak yang tidak hanya memiliki fungsi teknis, tetapi juga memenuhi kebutuhan pengujian parameter keamanan yang telah ditetapkan pada Bab III.

#### IV.1 Diagram Konseptual

Bagian ini menyajikan pemetaan alur sistem sebelum dan sesudah diterapkannya solusi, sehingga terlihat jelas bagaimana *packer* berperan sebagai perbaikan terhadap proses *existing*.

##### IV.1.1 Diagram *Packer* konvensional

Diagram “before” menunjukkan bagaimana *spyware* bekerja tanpa mekanisme *packing* atau penyamaran. *Payload* langsung diunduh dan dijalankan sehingga *anti-spyware* memiliki peluang lebih besar untuk mendeteksi struktur *file* yang masih jelas.

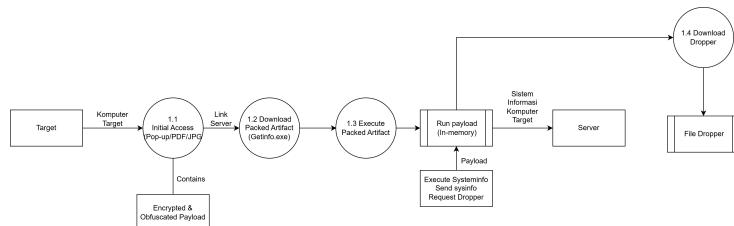


Gambar IV.1 *Packer Konvensional*

Pada sistem IV.1, *payload* *GetInfo* diunduh dalam bentuk yang masih mudah dianalisis oleh *anti-spyware*. Struktur file, signature, dan byte-pattern masih menyerupai malware konvensional sehingga tingkat deteksi cukup tinggi.

#### IV.1.2 Diagram Packer dengan Artefak Stealth

Diagram ini memperlihatkan bagaimana artefak *packer* mengubah alur kerja secara signifikan. *Payload* tidak lagi tampil sebagai *executable* biasa, melainkan sebagai *packed artifact* yang terenkripsi, ter-*obfuscate*, dan lebih sulit dideteksi.



Gambar IV.2 *Packer dengan artefak Stealth*

Perubahan utama terjadi pada tahap 1.2 dan 1.3, di mana *anti-spyware* dihadapkan pada berkas *executable* yang telah dimodifikasi oleh modul *packer*. Pada kondisi ini, berkas tersebut tidak lagi memiliki *signature* malware yang dikenal, menunjukkan tingkat entropi yang tinggi sebagai akibat dari proses kompresi atau enkripsi, serta mengandung *payload* yang disembunyikan melalui mekanisme enkripsi internal. Selain itu, artefak ini juga mampu memulai proses eksekusi secara *fileless*, sehingga menghindari banyak indikator statis yang lazim digunakan oleh mesin deteksi. Kombinasi karakteristik tersebut secara signifikan mengurangi kemampuan sistem

deteksi tradisional untuk mengenali pola ancaman, sehingga membuat mekanisme *anti-spyware* konvensional jauh lebih sulit dalam memprofilkan dan mengidentifikasi aktivitas berbahaya pada tahap awal.

#### IV.1.3 Perbandingan *Packer* Konvensional dan *Packer* dengan Artefak *Stealth*

Tabel IV.1 Perbandingan *Packer* Konvensional dan *Packer* dengan Artefak *Stealth*

Aspek	Packer Konvensional	Packer dengan Artefak <i>Stealth</i>
Bentuk Payload	File GetInfo asli dengan signature jelas	Payload terenkripsi dan tersembunyi dalam stub
Pola Deteksi	Mudah dikenali oleh AV berbasis signature	Signature berubah total, sulit dikenali
Eksekusi Payload	<i>On-disk execution</i>	<i>In-memory unpacking</i> (fileless behavior)
Jejak Sistem	Tinggi, mudah dilacak	Minim, struktur berubah dan <i>obfuscate</i>
Realisme Serangan	Rendah	Tinggi — menyerupai spyware modern

## IV.2 Klarifikasi Tugas Desain

Langkah pertama dalam desain solusi adalah menerjemahkan masalah menjadi menjadi *intended support* berupa desain artefak *packer*. Berdasarkan analisis kesenjangan (*Gap Analysis*) pada Bab III, tugas desain utama adalah menciptakan mekanisme penyembunyian *payload*.

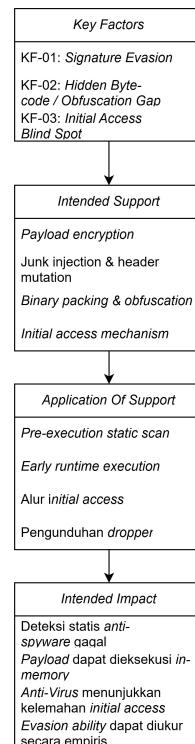
#### IV.2.1 Penentuan *Key Factors* dan *Intended Impact Model*

Berdasarkan *Gap Analysis* pada Bab III, diperoleh tiga *Key Factors* (*KF*) yang harus diatasi oleh artefak *packer*:

Tabel IV.2 Key Factors prototipe *packer*

<b>Key Factor (KF)</b>	<b>Celah Deteksi Anti-Spyware</b>	<b>Penjelasan</b>
KF-01 – <i>Signature Evasion</i>	CTQ-01	<i>Anti-spyware</i> gagal mendeteksi malware yang <i>signature</i> -nya berubah akibat teknik <i>packing</i> dan <i>encryption</i> .
KF-02 – <i>Hidden Byte-code / Obfuscation Gap</i>	CTQ-02	<i>Anti-virus</i> tidak mampu membaca <i>payload</i> yang terenkripsi atau ter- <i>obfuscate</i> di dalam biner <i>packer</i> .
KF-03 – <i>Initial Access Blind Spot</i>	CTQ-03	Banyak produk <i>anti-virus</i> hanya mendekksi saat <i>runtime</i> , bukan ketika <i>file</i> baru diunduh.

Untuk memastikan desain artefak *packer* benar-benar merespons *Key Factors* tersebut, dirumuskan Intended *Impact Model* (*IM*) yang menggambarkan hubungan kausal antara desain solusi dan dampak yang diharapkan.



Gambar IV.3 *Intended Impact Model*

Model dampak ini menunjukkan bahwa solusi yang dirancang tidak hanya bersifat

teknis, tetapi juga merupakan intervensi metodologis untuk menghasilkan data empiris mengenai kelemahan *anti-spyware* saat menghadapi artefak *stealth* pada tahap paling *initial access*.

#### IV.2.2 Desain Artefak Packer (*Intended Support*)

Berdasarkan *key factors* tersebut, *intended support* berupa artefak *packer* dirancang dengan karakteristik sebagai berikut:

1. Payload terenkripsi dan ter-*obfuscate* (tidak berbahaya).

Payload berisi instruksi sederhana, yaitu:

- menjalankan perintah systeminfo,
- mengirim hasilnya ke server,

Payload tidak berisi aktivitas spionase aktif, tidak mengakses file, tidak melakukan *keylogging*, dan tidak berkomunikasi secara berulang. Dengan demikian, artefak aman dan legal untuk penelitian.

Payload dienkripsi dan disisipkan ke dalam *stub executable*, sehingga:

- *signature* asli *payload* hilang,
- *static detection* oleh antivirus menjadi sulit.

2. Binary *packing* dan *obfuscation*.

Packer memodifikasi struktur binary menjadi bentuk yang tidak mudah dibaca oleh mesin statis, melalui:

- *header mutation*,
- *junk insertion*,
- *string obfuscation*,
- *section entropy manipulation*.

Transformasi ini secara langsung menargetkan KF-01 dan KF-02.

3. Eksekusi *payload* secara *in-memory*.

*Stub* mendekripsi *payload* langsung di RAM dan mengeksekusinya tanpa menuliskan file tambahan ke disk.

Metode ini secara langsung menguji *blind spot* anti-spyware pada fase awal eksekusi (KF-03).

4. Integrasi ke flow *Initial Access Packer* dengan artefak *Stealth*.

Flow ini mengikuti diagram IV.2:

- (a) Target mengakses pop-up/PDF/JPG.
- (b) GetInfo.exe diunduh dari server.
- (c) Artefak dieksekusi.
- (d) Payload *in-memory* menjalankan:

- systeminfo,
  - mengirim hasil ke server,
  - meminta dropper.
- (e) Server mengirimkan dropper.  
(f) Artefak mengunduh dropper (1,4).  
Tidak ada aktivitas spyware aktif pada tahap ini.

#### IV.2.3 Spesifikasi Mesin Uji

Pengujian akan dilakukan menggunakan *controlled environment* yang terisolasi untuk memastikan kondisi yang konsisten di seluruh produk anti-spyware yang diuji.

Tabel IV.3 Spesifikasi Lingkungan Target untuk Pengujian

<i>Komponen</i>	<i>Spesifikasi (Target VM)</i>
Sistem Operasi	Windows 11
Kapasitas	Intel Core i5 Gen 8
RAM	8 GB
Storage	SSD 1 TB
Jaringan	Public (ITB) & local (untuk pengujian)
Kakas Pendukung	Windows Defender, AVG, Avast, McAfee

### IV.3 Konseptualisasi Arsitektur Solusi

Konseptualisasi arsitektur solusi dilakukan untuk memberikan representasi menyeluruh mengenai bagaimana artefak *packer* bekerja sebagai komponen *initial access* dalam skenario serangan multi-tahap. Arsitektur ini dirancang berdasarkan *intended support* yang telah dijelaskan sebelumnya, serta dikonstruksi untuk mendukung evaluasi terhadap efektivitas deteksi *anti-spyware* pada tahap awal infiltrasi. Secara keseluruhan, arsitefak terdiri atas dua komponen inti, yakni *stub packer (loader)* sebagai entitas eksekusi utama pada sisi target dan *encrypted lightweight payload* sebagai muatan yang dijalankan secara *in-memory*. Kedua komponen ini berkolaborasi untuk menciptakan mekanisme penyamaran (*evasion*) yang relevan terhadap *key factors* penelitian, namun tetap menjaga batasan etis dan legalitas dengan tidak memasukkan fungsi *spyware* agresif atau destruktif.

#### IV.3.1 Komponen Arsitektur *Packer*

Arsitektur *packer* yang dikembangkan mencakup dua elemen fundamental yang berperan dalam proses packing, penyamaran, dan eksekusi *payload*.

#### **IV.3.1.1 *Stub Packer (Loader)***

*Stub packer* merupakan komponen *executable* yang bertanggung jawab untuk menjalankan seluruh tahapan kritis pada sisi target. Secara fungsional, stub memuat algoritma rekonstruksi kunci yang digunakan untuk mendekripsi *payload* yang tertanam di dalam artefak. Proses dekripsi dilakukan sepenuhnya di memori (*in-memory decryption*), sehingga tidak meninggalkan artefak fisik yang dapat dianalisis oleh mekanisme *file-based scanning*. Setelah payload berhasil didekripsi, stub mengeksekusi instruksi-instruksi yang terkandung di dalamnya, termasuk menjalankan perintah systeminfo untuk memperoleh informasi dasar mengenai sistem target. Informasi tersebut kemudian dikirimkan ke server sebagai bagian dari proses *lightweight reconnaissance* dan sekaligus sebagai pemicu permintaan dropper. Selain itu, stub juga menangani respons server dengan menginisiasi proses pengunduhan *file dropper*. Dengan demikian, *stub packer* menjadi pusat pengendali proses *initial access* sekaligus modul yang memungkinkan terjadinya transisi dari tahap pengintaian awal menuju tahap infeksi berikutnya.

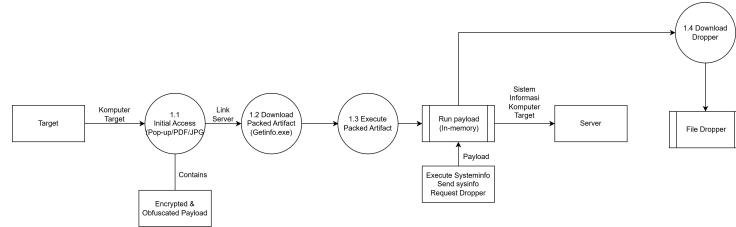
#### **IV.3.1.2 *Encrypted Lightweight Payload***

Komponen kedua dalam arsitektur packer adalah *payload* terenkripsi yang disisipkan ke dalam stub. Payload ini bersifat ringan dan tidak mengandung fungsi spionase, perusakan, atau mekanisme *persistensi*. Muatan ini hanya terdiri atas tiga instruksi inti, yaitu menjalankan perintah systeminfo, mengirim hasil eksekusi tersebut ke server, dan meminta pengiriman file dropper. Payload tidak melakukan interaksi terhadap berkas sistem, tidak memodifikasi *registry*, serta tidak melakukan aktivitas laten lainnya yang berpotensi dikategorikan sebagai perilaku spyware berbahaya. Payload didesain sedemikian rupa agar merepresentasikan tahap awal serangan spyware modern, yaitu pengumpulan informasi minimal untuk menentukan bagaimana tahap berikutnya akan diluncurkan oleh server. Enkripsi payload memastikan bahwa mesin deteksi statis tidak dapat mengidentifikasi pola instruksi internal sebelum proses dekripsi terjadi di memori.

### **IV.3.2 Alur Packer dengan artefak Stealth**

Alur sistem IV.4 menggambarkan bagaimana artefak *packer* beroperasi pada lingkungan target setelah solusi diimplementasikan. Alur ini terdiri atas serangkaian tahapan berurutan yang merepresentasikan fase *initial access* hingga pengunduhan *dropper*. Setiap tahapan dirancang untuk menguji kemampuan deteksi *anti-spyware* pada titik-titik kritis, dengan tetap menjaga agar *payload* yang digunakan bersifat

ringan dan tidak destruktif.



Gambar IV.4 *Packer* dengan artefak *Stealth*

1. Initial Access - Tahap 1.1 Tahap pertama merupakan fase *initial access* di mana target berinteraksi dengan konten yang telah dimanipulasi sebagai vektor serangan. Konten tersebut dapat berupa pop-up, dokumen PDF, atau gambar (JPG) yang secara sengaja dikemas untuk menampilkan tautan atau aksi yang mengarahkan pengguna ke server penyerang. Dari sudut pandang penelitian, tahap ini mensimulasikan skenario *social engineering* yang lazim digunakan dalam distribusi malware modern, namun tanpa menyertakan eksplorasi kerentanan tertentu. Pada titik ini, sistem target belum menjalankan kode berbahaya apa pun; yang terjadi hanyalah interaksi pengguna dengan media yang tampak *benign*.
2. Download Artefak - Tahap 1.2 Setelah pengguna mengikuti tautan yang disediakan, sistem akan mengunduh sebuah artefak eksekutabel, yaitu berkas GetInfo.exe. Artefak ini merupakan hasil proses *packing* yang mengandung payload terenkripsi dan ter-*obfuscate*. Proses pengunduhan ini menggambarkan fase di mana banyak produk *anti-spyware* melakukan pemindaian awal (misalnya *on-access scanning* terhadap file yang baru dibuat di disk). Pada penelitian ini, struktur biner GetInfo.exe telah dimodifikasi sedemikian rupa sehingga *fingerprint*-nya tidak lagi identik dengan payload asli. Dengan demikian, tahap 1.2 menjadi titik penting untuk mengamati apakah *engine anti-spyware* masih mampu mengidentifikasi artefak sebagai ancaman berdasarkan karakteristik statisnya.
3. Execute Packed Artifact - Tahap berikutnya adalah eksekusi artefak yang telah diunduh. Pada saat pengguna atau sistem menjalankan GetInfo.exe, produk *anti-spyware* umumnya akan melakukan kombinasi antara pemindaian statis tambahan dan analisis perilaku awal. Di sisi lain, dari perspektif solusi, *stub packer* mulai berfungsi sebagai *entry point* yang menyiapkan lingkungan eksekusi payload. Pada tahap ini belum terjadi aktivitas spionase ataupun pengumpulan data sensitif; stub hanya mempersiapkan proses dekripsi payload. Tujuan utama tahap 1.3 adalah mengevaluasi apakah transisi dari

status “file di disk” menjadi “proses yang dieksekusi” akan memicu deteksi dini dari *anti-spyware*, mengingat struktur biner telah mengalami *packing* dan *obfuscation*.

4. Run Payload (In-Memory) Setelah stub berhasil dijalankan, proses berlanjut ke eksekusi payload secara *in-memory*. Stub melakukan rekonstruksi kunci enkripsi berdasarkan algoritma yang tertanam di dalamnya, kemudian mendekripsi payload ringan yang disisipkan sebelumnya. Proses dekripsi ini tidak menulis file baru ke disk, melainkan langsung memuat kode hasil dekripsi ke memori dan mengeksekusinya. Pendekatan ini dirancang untuk mengeksploitasi kelemahan mekanisme deteksi yang bergantung pada artefak file, sekaligus mensimulasikan teknik *fileless malware* yang banyak digunakan pada serangan kontemporer. Tahap ini menjadi krusial untuk menguji sejauh mana *anti-spyware* mampu melakukan inspeksi memori proses dan mendeteksi aktivitas dekripsi serta pemanggilan fungsi internal yang tampak *benign*.
5. Payload Execution - Sysinfo dan Permintaan Dropper Payload yang telah berhasil didekripsi kemudian menjalankan tiga instruksi dasar. Pertama, payload mengeksekusi perintah systeminfo atau fungsi setara untuk memperoleh informasi lingkungan sistem, seperti versi sistem operasi, arsitektur, dan parameter penting lainnya. Kedua, hasil eksekusi systeminfo diproses menjadi representasi data yang ringkas (misalnya bentuk *string* atau struktur lain) sehingga dapat dikirimkan melalui jaringan. Ketiga, payload mengirimkan informasi tersebut ke server dan secara eksplisit meminta pengiriman file dropper. Seluruh rangkaian aktivitas ini diklasifikasikan sebagai *lightweight reconnaissance* karena hanya mengumpulkan informasi umum sistem tanpa menyentuh berkas, *registry*, maupun kredensial pengguna. Dari perspektif penelitian, tahap ini memungkinkan pengamatan apakah *anti-spyware* akan menganggap kombinasi eksekusi systeminfo dan komunikasi jaringan sederhana ini sebagai perilaku mencurigakan.
6. Server Response Setelah menerima data systeminfo dari target, server bertindak sebagai *command and control (C2)* emulator yang bertugas memilih dan mengirimkan file dropper yang sesuai. Logika pemilihan dropper dapat mempertimbangkan informasi sistem yang dikirimkan, misalnya arsitektur atau versi sistem operasi, meskipun dalam penelitian ini dapat disederhanakan sesuai kebutuhan eksperimen. Respons server berupa pengiriman lokasi atau langsung berkas dropper menjadi indikator bahwa tahap *reconnaissance* awal berhasil dijalankan tanpa terputus oleh mekanisme pertahanan. Pada titik ini, seluruh fungsi packer masih berada dalam koridor penelitian karena artefak

belum menjalankan aktivitas spyware tingkat lanjut.

7. Dropper Download - Tahap 1.4 Tahap terakhir dalam alur “after” adalah pengunduhan dropper oleh artefak di sisi target. Proses ini menandai selesainya fase *initial access* dan perpindahan kontrol menuju artefak generasi berikutnya. Secara konseptual, tahap 1.4 penting karena menunjukkan bahwa rangkaian aktivitas mulai dari pengunduhan packer, eksekusi payload *in-memory*, hingga komunikasi dengan server dapat berlangsung tanpa dihentikan oleh *anti-spyware*. Dalam konteks penelitian, keberhasilan mencapai tahap ini menjadi indikator bahwa teknik *packing* dan eksekusi *fileless* yang diimplementasikan pada artefak packer cukup efektif dalam mengurangi visibilitas terhadap mekanisme deteksi, setidaknya hingga batas awal serangan yang di-modelkan.

#### IV.4 Actual Support

Pada tahap *Prescriptive Study – Actual Support*, desain konseptual packer diwujudkan menjadi artefak konkret yang dapat dieksekusi dan diuji pada lingkungan eksperimen. Payload hanya menjalankan instruksi minimal berupa eksekusi systeminfo, pengiriman hasil ke server, serta permintaan pengiriman dropper sebagai tahap selanjutnya dalam skenario multi-tahap. Dengan demikian, *Actual Support* yang dihasilkan bersifat aman, terkendali, dan sepenuhnya digunakan untuk tujuan penelitian ilmiah dalam mengevaluasi kemampuan deteksi *anti-spyware* pada fase *initial access*. Artefak packer yang direalisasikan terdiri atas beberapa modul sebagai berikut.

##### IV.4.1 Modul yang Diimplementasikan

###### 1. Payload Builder

Modul ini bertugas mengubah instruksi systeminfo menjadi representasi byte-array yang dapat dienkripsi dan disisipkan ke dalam stub. Proses ini meliputi encoding, kompresi ringan, dan penyiapan struktur payload sehingga dapat diproses oleh *stub loader* pada saat runtime. Modul ini memastikan bahwa payload tidak muncul sebagai kode plaintext pada binary final, sehingga tidak dapat diidentifikasi melalui analisis statis.

###### 2. Stub Loader

*Stub loader* merupakan komponen inti yang dieksekusi saat artefak dijalankan pada mesin target. Loader bertanggung jawab untuk merekonstruksi kunci enkripsi, mendekripsi payload sepenuhnya di memori (*in-memory decryption*)

*tion*), serta memulai eksekusi payload tanpa menuliskan file tambahan ke disk. Pendekatan ini dirancang untuk mengeksploitasi kelemahan deteksi berbasis file dan mensimulasikan teknik *fileless execution*.

### 3. Sysinfo Executor

Setelah payload berhasil didekripsi, modul Sysinfo Executor menjalankan instruksi systeminfo melalui mekanisme yang telah ditentukan oleh sistem operasi, tanpa melakukan modifikasi pada konfigurasi OS maupun *registry*. Output systeminfo kemudian diekstraksi dan diformat ke dalam bentuk yang siap dikirimkan ke server. Modul ini berfungsi sebagai *lightweight reconnaissance agent* yang tidak menimbulkan perilaku mencurigakan pada tingkat sistem.

### 4. Network Handler

Modul ini mengelola komunikasi dengan server. Fungsi utamanya meliputi pengiriman data hasil eksekusi systeminfo dan pemanggilan *endpoint* server untuk meminta pengiriman dropper. Modul ini dirancang untuk menghasilkan pola komunikasi *outbound* yang sederhana dan natural, sehingga tidak memiliki deteksi berbasis anomali lalu lintas jaringan.

### 5. Dropper Downloader

Setelah menerima respons dari server, modul ini memfasilitasi pengunduhan file dropper sebagai tahap lanjutan dari skenario serangan. Modul bekerja berdasarkan instruksi server dan menjadi titik akhir alur *initial access*. Tidak ada eksekusi dropper yang dilakukan oleh packer pada tahap ini, sehingga artefak tetap berada dalam batasan aman dan non-destruktif.

## IV.5 Verifikasi Internal dan Rencana Evaluasi

Bagian ini menjelaskan mekanisme verifikasi awal terhadap artefak packer untuk memastikan bahwa implementasi Actual Support sesuai dengan desain konseptual (Intended Support) dan siap diuji pada tahap Descriptive Study II (Bab V). Verifikasi dilakukan melalui pengujian internal terhadap performa packer dalam menjalankan fungsinya dan pengamatan terhadap respons awal produk anti-spyware.

### IV.5.0.1 Kriteria Keberhasilan (Internal Success Criteria)

Kriteria keberhasilan berikut digunakan untuk menentukan apakah artefak packer telah berfungsi sesuai harapan sebelum memasuki evaluasi formal:

#### 1. Kegagalan deteksi pada tahap download (1.2)

Artefak packer diharapkan tidak terdeteksi sebagai ancaman oleh *anti-spyware* ketika diunduh oleh target. Keberhasilan pada tahap ini menunjukkan bahwa

struktur binary hasil *packing* dan *obfuscation* mampu menurunkan efektivitas *signature-based detection* maupun pemindaian statis yang dilakukan pada file yang baru dibuat pada disk.

2. Payload berhasil dieksekusi secara *in-memory* tanpa menghasilkan peringatan berat

Kriteria ini mengukur apakah *anti-spyware* dapat mendeteksi aktivitas dekripsi payload dan eksekusinya di memori. Keberhasilan tercapai jika payload dapat dijalankan tanpa menghasilkan blokir proses atau peringatan tingkat tinggi, yang mengindikasikan bahwa mekanisme *fileless execution* efektif dalam menghindari inspeksi statis maupun perilaku umum.

3. Pengiriman systeminfo berhasil dilakukan

Setelah payload dieksekusi, hasil systeminfo harus berhasil dikirimkan ke server. Keberhasilan pengiriman menunjukkan bahwa payload dan *network handler* bekerja sesuai rancangan, serta tidak dihentikan oleh mekanisme perlindungan jaringan atau heuristik perilaku.

4. Server mengirimkan dropper dan artefak berhasil mengunduhnya (1.4)

Tahap akhir verifikasi adalah keberhasilan artefak menerima respons server berupa dropper dan menginisiasi proses pengunduhan. Hal ini memastikan bahwa keseluruhan alur *initial access* (1.1 hingga 1.4) dapat diselesaikan tanpa interupsi dari *anti-spyware*.

#### **IV.5.0.2 Ruang Lingkup Evaluasi**

Rencana evaluasi tidak berfokus pada kerusakan sistem atau fungsionalitas dropper tahap kedua, melainkan menganalisis:

- kemampuan *anti-spyware* dalam mendeteksi artefak pada fase awal infiltrasi,
- efektivitas *packing* dan *in-memory execution* dalam menurunkan deteksi statis dan dinamika awal,
- respons terhadap komunikasi *outbound* minimal,
- ketahanan terhadap pemindaian heuristik selama proses download, eksekusi, dan pengiriman sistem informasi.

Dengan demikian, penelitian ini memosisikan artefak packer sebagai alat evaluatif untuk mengidentifikasi celah deteksi *anti-spyware* pada *initial access*, bukan sebagai malware operasional.

## **BAB V**

### **RENCANA SELANJUTNYA**

#### **V.1 Rencana Implementasi**

Rencana implementasi di sini merujuk pada langkah-langkah praktis dan alat yang diperlukan untuk menyiapkan dan menjalankan lingkungan pengujian prototipe, yang merupakan prasyarat sebelum evaluasi formal (*DS-II*) dapat dimulai.

##### **V.1.1 Langkah-langkah Implementasi**

Implementasi dilakukan dalam lingkungan pengujian yang dikontrol (*Controlled Environment*) untuk mengisolasi efek prototipe dari variabel eksternal perancu.

Tabel V.1 Rencana Implementasi Prototipe Stealth

<b>Langkah</b>	<b>Justifikasi Metodologis (DRM)</b>	<b>Alat yang Dibutuhkan</b>
Persiapan Infrastruktur Laboratorium Host	Mengatur Host Windows 11 untuk pengujian ( <i>baseline</i> ). Isolasi dilakukan dengan memutus koneksi jaringan eksternal saat binary dieksekusi, memastikan malware tidak menyebar ( <i>R-I</i> ). Digunakan untuk menjamin <i>Internal Validity</i> .	Windows 11 Host Fisik.
Instalasi dan Konfigurasi Produk Keamanan ( <i>Target Evaluasi</i> )	Menginstal dan mengaktifkan berbagai produk <i>Anti-Malware</i> dan <i>EDR</i> komersial yang telah dipilih (misalnya, Produk A, B, dan C) pada Host Windows 11. Konfigurasi keamanan harus ditetapkan pada pengaturan <i>default</i> atau konfigurasi yang konsisten.	Lisensi Produk Keamanan.
Finalisasi <i>Actual Support</i> (Prototipe)	Menyuntikkan payload (Prototipe Stealth) ke Host Windows 11. Prototipe diuji fungsionalitas internalnya ( <i>Support Evaluation</i> dari <i>Prescriptive Study</i> ) sebelum digunakan dalam evaluasi.	<i>Scripting Environment</i> (misalnya <i>PowerShell ISE</i> , terminal), <i>Custom Packer/Loader</i> .

## V.2 Rencana Evaluasi

Rencana evaluasi akan membagi proses pengujian menjadi *Application Evaluation* (fokus pada fungsi langsung) dan *Success Evaluation* (fokus pada MSC/dampak keseluruhan), menggunakan desain *quasi-experimental comparative study*.

### V.2.1 Metode Pengujian: Studi Kuasi-Eksperimental Komparatif

Metode pengujian menggunakan pendekatan *Blackbox Testing* dan *Whitebox Analysis* di lingkungan virtual:

1. *Blackbox Testing* (Pengujian Fungsionalitas Deteksi) Mengukur status deteksi yang terlihat oleh pengguna atau sistem *anti-spyware* tanpa melihat kode internal. Meliputi:

- Deteksi Statis (*Pre-Execution*) Menguji apakah Packer Spyware terdeteksi hanya karena *file signature*.
  - Deteksi Dinamis (*Runtime*) Menguji apakah produk keamanan mendeteksi perilaku *in-memory* saat *payload* di-*execute*.
2. *Whitebox Analysis* (Validasi Stealth) Mengukur apakah antivirus mendeteksi perilaku runtime ketika stub melakukan dekripsi *in-memory*, *payload* menjalankan *systeminfo*, *payload* mengirim data ke server.

### V.2.2 Kriteria Keberhasilan

Keberhasilan diukur berdasarkan efektivitas prototipe dalam mengeksplorasi celah yang telah diidentifikasi, melampaui performa *control payload*.

Tabel V.2 Kriteria Keberhasilan dan Indikator Pengukuran

Success Criterion (SC)	Measurable Success Criteria (MSC)	Definisi Operasional & Indikator Pengukuran
SC-1: Keberhasilan <i>evaluation</i> statis	MSC-1: Artefak tidak terdeteksi pada fase download (1.2)	Antivirus tidak memblokir atau mengkarantina file saat proses unduhan; hasil <i>scan</i> awal menunjukkan status <i>clean / unknown</i> ; <i>signature-based detection</i> gagal membaca struktur biner hasil <i>packing</i> .
SC-2: Eksekusi <i>in-memory</i> berhasil dan tidak menghasilkan <i>alert</i> tinggi	MSC-2: Payload berhasil didekripsi dan dijalankan sepenuhnya di memori tanpa <i>alert</i> tingkat tinggi dari <i>anti-spyware</i>	<i>Process Monitor</i> menunjukkan proses stub berjalan normal; tidak ada pemblokiran <i>runtime</i> ; antivirus tidak mendeteksi aktivitas dekripsi maupun eksekusi instruksi <i>systeminfo</i> .
SC-3: Komunikasi <i>outbound</i> berhasil ( <i>sysinfo transmission</i> )	MSC-3: Hasil <i>systeminfo</i> terkirim ke server dan diterima dengan benar	<i>Traffic outbound</i> minimal terlihat stabil; server mencatat log penerimaan data <i>systeminfo</i> ; <i>firewall / EDR</i> tidak memblokir <i>request</i> .

### V.2.3 Analisis Risiko

Risiko-risiko berikut dapat mempengaruhi validitas maupun efisiensi proyek:

Tabel V.3 Analisis Risiko Pengembangan dan Pengujian Prototipe Stealth

Risiko	Deskripsi Risiko	Strategi Mitigasi
<i>Accidental Execution</i> di Host	Binary spyware bocor dan tidak sengaja dieksekusi pada host.	Isolasi penuh VM dan menggunakan <i>host-only network</i> .
Deteksi <i>Anti-Malware</i> terhadap VM	Produk keamanan mendeteksi lingkungan virtual dan mengubah perlakunya.	<i>VM masking</i> , standarisasi konfigurasi VM.
Kompleksitas Implementasi Rust	<i>Debugging</i> low-level membutuhkan waktu lebih lama.	Modularisasi dan memanfaatkan <i>memory safety</i> Rust.
<i>False Negative</i> pada Anti-Spyware	Lisensi terbatas mengurangi jumlah produk yang diuji.	Menggunakan <i>trial</i> , sandbox publik ( <i>VirusTotal</i> , <i>Cuckoo</i> ) untuk <i>pre-screening</i> .

Dengan metrik-metrik tersebut, proses evaluasi menghasilkan penilaian kuantitatif dan kualitatif terhadap efektivitas Packer Spyware Mode Stealth, terutama dalam evasi *signature*, eksekusi *in-memory*, serta kemampuan menghindari analisis perilaku *anti-spyware*.

## **DAFTAR PUSTAKA**

- Ainslie, Scott, Dean Thompson, Sean Maynard, dan Atif Ahmad. 2023. “Cyber-threat intelligence for security decision-making: A review and research agenda for practice”. *Computers & Security* 132:103352. <https://doi.org/10.1016/j.cose.2023.103352>.
- Domínguez-Dorado, Manuel, Francisco J. Rodríguez-Pérez, Jesús Galeano-Brajones, Jesús Calle-Cancho, dan David Cortés-Polo. 2024. “FLECO: A tool to boost the adoption of holistic cybersecurity management”. *Software Impacts* 19:100614. <https://doi.org/10.1016/j.simpa.2024.100614>.
- E., Chatzoglou, Kambourakis G., Karupoulos G., dan Tsiatsikas Z. 2023. “Bypassing antivirus detection: old-school malware, new tricks”. Disiniasi dari file ”2023 Bypassing antivirus detection old-school malware, new tricks.pdf”.
- Elghaly dan Yehia M. 2024. “Stealth in Plain Sight: The Hidden Threat of PowerShell Fileless Malware and Its Evasion of Modern EDRs & AVs”. Disiniasi dari file ”2024 Stealth in Plain Sight The Hidden Threat of PowerShell Fileless Malware and Its Evasion of Modern EDRs & AVs.pdf”.
- Kareem. 2024. “A Comprehensive Analysis of Pegasus Spyware and Its Implications for Digital Privacy and Security”. Disiniasi dari file ”A Comprehensive Analysis of Pegasus Spyware and Its Implications for Digital Privacy and Security.pdf”.
- Kerkour, Sylvain. 2021. *Black Hat Rust: Applied offensive security with the Rust programming language*. Self-published.
- Koutsokostas, V, dan C Patsakis. 2021. “Python and Malware: Developing Stealth and Evasive Malware Without Obfuscation”. Disiniasi dari file ”2021 Python and Malware Developing Stealth and Evasive Malware Without Obfuscation.pdf”.

- Sudhakar, Unknown, dan Sushil Kumar. 2020. “An emerging threat Fileless malware: a survey and research challenges”. *Cybersecurity* 3 (1): 1. <https://doi.org/10.1186/s42400-019-0043-x>.
- Verma, V., S. K. Dubey, T. Khan, dan Pandey H. Prem. 2021. “The Study on Information Gathering”. Disiniasi dari file ”2021 Cyber Security The Study on Information Gathering.pdf”.
- Zhu, Yuxin, Tian, Dazuo, dan Feng Yan. 2020. “Effectiveness of Entropy Weight Method in Decision-Making”. *Mathematical Problems in Engineering* 2020 (Article ID 3564835): 1–5. <https://doi.org/10.1155/2020/3564835>. <https://doi.org/10.1155/2020/3564835>.