Steganografi Melalui *Source Code*

Azra Haikal Fahlevi (13222045[1,2])
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13222045@std.stei.itb.ac.id, [2]azrahaikal75@gmail.com

Abstrak*—Di era digital, keamanan komunikasi menjadi kebutuhan penting. Meskipun kriptografi dapat melindungi data, pesan yang teracak sering kali memancing kecurigaan pihak ketiga. Steganografi menawarkan alternatif dengan menyembunyikan keberadaan pesan itu sendiri. Makalah ini mengimplementasikan steganografi menggunakan source code sebagai mediumnya, mengingat pertukaran kode adalah hal yang umum dalam lingkungan pengembangan perangkat lunak. Metode yang diusulkan menggunakan source code berbasis Python dengan dua teknik utama: pemanfaatan karakter tak terlihat (spasi dan tab) di akhir baris untuk merepresentasikan data biner, dan manipulasi gaya penulisan (seperti snake case, camel case) pada komentar program berdasarkan pemetaan biner tertentu. Metode ini memungkinkan penyisipan pesan rahasia – termasuk teks, gambar, dan dokumen – tanpa merusak logika atau menyebabkan error pada eksekusi program. Pengujian menunjukkan bahwa data yang disisipkan dapat diekstraksi kembali dengan tetap menjaga integritas visual dan fungsional dari program pembawa – kecuali pada file pesan berbentuk pdf.

***Kata kunci*—Komentar, *Space, Source Code*, Steganografi, *Tab*.**

I. PENDAHULUAN

Di zaman sekarang, komunikasi secara digital sudah menjadi kebutuhan setiap orang. Dengan teknologi yang ada, 2 orang dapat saling bertukar informasi meskipun berada di lokasi yang berjauhan. Salah satu hal yang harus menjadi perhatian adalah keamanan dari proses pertukaran informasi tersebut.

Kriptografi biasanya digunakan untuk mengamankan suatu informasi. Pada kriptografi, suatu pesan diacak sehingga tidak bisa dibaca oleh pihak ketiga. Namun, keberadaan pesan yang sudah teracak ini dapat memancing kecurigaan pihak ketiga.

Teknik lain yang dapat digunakan untuk mengamankan pertukaran informasi adalah steganografi. Berbeda dengan kriptografi, steganografi tidak mengacak-acak bentuk suatu pesan. Pada steganografi, suatu pesan disembunyikan sedemikian rupa sehingga keberadaannya tidak dapat disadari oleh pihak ketiga.

Sebagian besar implementasi steganografi yang ada pada saat ini berfokus pada media-media digital seperti gambar, suara, atau *video*. Metode steganografi melalui medium *source code* masih jarang digunakan.

Dalam lingkungan pengembang perangkat lunak, pertukaran kode adalah hal yang sering terjadi. Kondisi ini menjadikan *source code* sebagai medium steganografi yang ideal untuk menyembunyikan pesan tanpa menimbulkan kecurigaan. Pada metode steganografi ini, tantangan utamanya adalah ketika menyisipkan pesan ke *source code* tanpa merusak logika program atau menyebabkan *error* saat dieksekusi.

A. Tujuan

Berikut adalah tujuan penulisan dari makalah ini.

1. Mengembangkan metode steganografi melalui medium *source code*.
2. Mengevaluasi implementasi steganografi melalui medium *source code*.
3. Mengenali batasan-batasan pada metode steganografi melalui medium *source code*.

B. Batasan Masalah

Pada makalah ini, metode steganografi yang dikembangkan hanya bisa digunakan pada kondisi-kondisi tertentu. Berikut adalah batasan-batasannya.

1. *Source code* yang bisa digunakan adalah yang berbasis bahasa Phyton.
2. Utamanya, program dibuat untuk menyisipkan pesan dalam bentuk teks.
3. Program mendukung pesan berekstensi .docx, .png, .txt, dan .jpg.

II. DASAR TEORI

A. Steganografi

Kata "steganografi" berasal dari bahasa Yunani, yaitu *steganos* yang berarti "tersembunyi" atau "tertutup" dan *graphein* yang berarti "menulis". Secara definisi, steganografi adalah seni dan ilmu komunikasi di mana keberadaan pesan itu sendiri disembunyikan. Berbeda dengan kriptografi yang mengacak isi pesan agar tidak terbaca, steganografi bertujuan agar pesan tersebut tidak terdeteksi sama sekali oleh pihak ketiga [1].

Tujuan utama steganografi adalah menyembunyikan data rahasia (*payload*) ke dalam media pembawa (*carrier*) sedemikian rupa sehingga tidak menimbulkan kecurigaan. Media yang telah disisipi pesan disebut sebagai *stego-object*. Jika metode steganografi berhasil, maka *stego-object* tidak terlihat berbeda secara signifikan dibandingkan dengan *cover-object* (media asli) [1].

B. Steganografi Teks

Steganografi pada teks (*text steganography*) dianggap sebagai salah satu bentuk steganografi yang paling sulit dilakukan karena kurangnya informasi *redundant* dalam file teks dibandingkan dengan gambar atau audio. Perubahan kecil pada teks dapat dengan mudah dideteksi secara visual atau semantik.

Namun, dengan medium *source code*, terdapat hal-hal tertentu yang dapat dieskploitasi dan tidak memengaruhi eksekusi program. Teknik steganografi pada *source code* umumnya memanfaatkan area yang diabaikan oleh *compiler* atau *interpreter*. Menurut penelitian yang dilakukan oleh Por et al. (2008) [2], metode steganografi pada teks dapat diklasifikasikan menjadi beberapa teknik.

1. *Format-based*: memanfaatkan spasi, tab, atau format baris.
2. *Random and Statistical Generation*: menghasilkan teks yang tampak acak namun menyembunyikan pola.
3. *Linguistic*: Memanfaatkan struktur bahasa (sinonim, sintaksis).

C. Representasi Data Biner

Komputer modern bekerja secara digital. Artinya, komputer melakukan semua proses komputasi sebagai sekumpulan bilangan biner (0 dan 1). Jenis data teks, gambar, dokumen pdf, *video,* dan lain-lain, dapat direpresentasikan dalam bentuk biner-nya. Sebuah teks biasanya direpresentasikan menggunakan standar ASCII atau UTF-8: setiap karakter diwakili oleh 8 bit (1 *byte*). Data berjenis pdf atau dokumen *word* terdiri dari struktur tertentu yang lebih kompleks.

Untuk menyisipkan *file* ke dalam *source code*, *file* tersebut harus terlebih dahulu dibaca dalam mode *byte*, kemudian dikonversi menjadi string bit yang panjang. Proses ekstraksi (*decoding*) melakukan kebalikannya: membaca *string bit*, mengelompokkannya per 8 bit, dan menulisnya kembali menjadi *file*.

D. Gaya Penulisan pada Komentar

Pada suatu *source code*, komentar berfungsi sebagai dokumentasi bagi pengembang dan tidak akan dieksekusi oleh mesin. Sifat dari komentar ini dapat dieksploitasi menjadi tempat ideal untuk menyembunyikan data.

Dalam dunia pemrograman, terdapat berbagai jenis konvensi penamaan variabel atau fungsi yang popular [3], seperti

1. *Snake case*. Kata-kata dipisahkan oleh garis bawah: contohnya, hello_world.
2. *Camel case*. Huruf pertama di kata pertama adalah non-kapital dan huruf-huruf pertama di kata-kata berikutnya adalah kapital: contohnya, helloWorld.
3. *Pascal case*. Setiap huruf pertama dari suatu kata ditulis secara kapital: contohnya, HelloWorld.
4. *Kebab case*. Setiap hutuf dipisahkan oleh tanda hubung: contohnya, hello-world.

Jenis-jenis gaya penulisan di atas dapat diadaptasi ke steganografi. Misalnya, dengan dibuat pemetaan antara kombinasi bit-bit biner terhadap gaya penulisan tertentu.

III. PERANCANGAN PROGRAM

A. Penggunaan Spasi dan Tab

Karakter spasi (*space*) dan *tab* dapat digunakan untuk merepresentasikan data biner. Metode ini dipilih karena karakter-karakter tersebut tidak terlihat secara visual pada *text editor* dan umumnya diabaikan oleh *compiler/interpreter* sehingga tidak membuat program menjadi *error*.

Pada program yang dirancang, bit 0 direpresentasikan oleh karakter *space* (ASCII 32) dan bit 1 direpresentasikan oleh karakter *tab* (ASCII 9). Proses penyisipan dilakukan di akhir setiap baris. Program dapat diatur untuk menentukan jumlah N bit yang akan disisipkan di setiap barisnya.

Untuk menggunakan *space* dan *tab*, pertama pesan dikonversi dahulu menjadi bentuk *bitstream* (deretan biner). Kemudian, pesan dipotong-potong menjadi segmen-segmen kecil yang berukuran N bit. Lalu, *string* spasi/*tab* tersebut ditempelkan di setiap ujung baris yang ada di *source code*. Jika baris pada *source code* terlalu pendek sehingga masih ada *bitstream* pesan yang belum disisipkan, sistem dirancang untuk menghasilkan baris baru di bagian *source code* yang paling bawah. Baris baru tersebut hanya berisi *space/tab* yang merepresentasikan pesan rahasia.

Gambar 1. *Flowchart* untuk proses menyisipkan pesan melalui *space/tab*.

Untuk meng-ekstraksi pesan yang dikodekan melalui *space/tab*, program memindai tiap baris dan mulai membaca baris tersebut dari bagian akhirnya. Selama karakter yang ditemukan adalah *space/tab*, karakter tersebut dicatat. Pemindaian pada baris berhenti ketika ditemukan karakter selain *space/tab* atau ketika mencapai awal baris. Jika di *source code* yang sesungguhnya terdapat *space/tab* di akhir suatu baris, *space/tab* tersebut akan dihapus supaya tidak terjadi konflik.

B. Penggunaan Gaya Penulisan pada Komentar

Pada program yang dibuat, didefinisikan 8 jenis komentar. Setiap 2 kata pada suatu komentar akan

digunakan untuk merepresentasikan 3 bit.

TABEL 1
Definisi Jenis-Jenis Komentar

No.	Jenis Komentar	Nilai Biner
1.	*Snake case*	000
2.	*Kebab case*	001
3.	*Camel case*	010
4.	*Pascal case*	011
5.	2 kata dan semuanya huruf non-kapital	100
6.	2 kata dan lalu hanya huruf pertama di kata pertama yang kapital	101
7.	1 kata dan huruf pertamanya kapital	110
8.	1 kata dan huruf pertamanya non-kapital	111

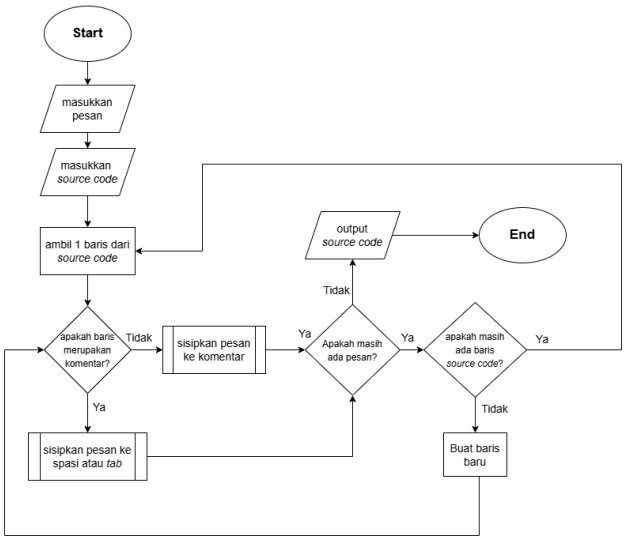
Untuk jenis 7 dan 8, jika pada suatu komentar hanya tersisa 1 kata, maka kata keduanya adalah sama dengan kata pertama dan diakhiri dengan dua buah titik: contohnya, untuk biner 110, komentar "hello" menjadi "Hello Hello..".

Proses penyisipan pada komentar dirancang agar tidak mengubah makna kalimat dari komentar tersebut: hanya mengubah tampilan visualnya saja. Pertama, program memindai baris kode untuk menemukan penanda komentar (misal, pada Python komentar dapat diawali dengan "#"). Kemudian, isi komentar dibersihkan dari symbol-simbol non-alfabet dan dipecah-pecah menjadi *array of word*. Program dilanjutkan dengan mengambil 3 bit dari pesan rahasia, lalu mengambil 2 kata dari *array of word*, lalu kedua kata tersebut diubah formatnya sesuai dengan tabel 1. Kata-kata yang telah diubah gayanya digabungkan kembali menjadi kalimat komentar yang baru.

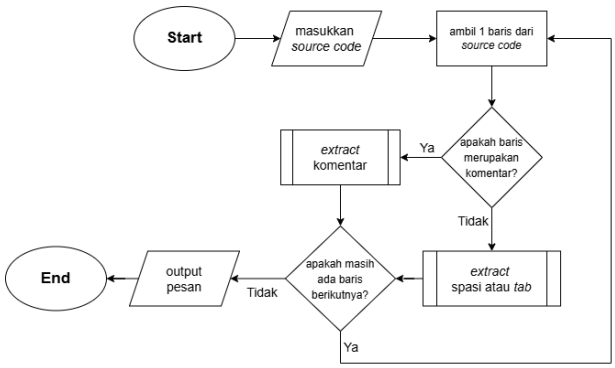
Pada lingkungan *programmer*, variasi gaya penulisan seperti *camelCase* atau *snake_case* adalah hal yang sangat umum dalam koding, sehingga penggunaan berbagai jenis gaya penulisan komentar tidak terlalu memancing kecurigaan.

C. Flowchart Program

Berikut adalah *flowchart* dari program yang dibuat.



Gambar 2. *Flowchart program* untuk proses *encoding* steganografi.



Gambar 3. *Flowchart program* untuk proses *decoding* steganografi.

IV. PENGUJIAN

Program yang dibuat diuji dengan berbagai bentuk pesan: teks, .txt, .png, .jpg, .pdf, dan .docx. Berikut adalah hasilnya.

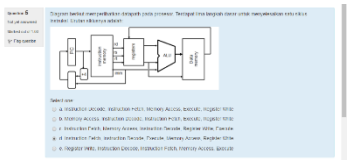
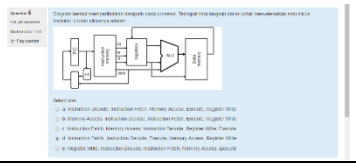
TABEL 2
Hasil Pengujian Program Steganografi

Jenis pesan	Hasil
Teks	Pesan asli: `>> Masukkan pesan rahasia: Ini pesan rahasia sekali!`

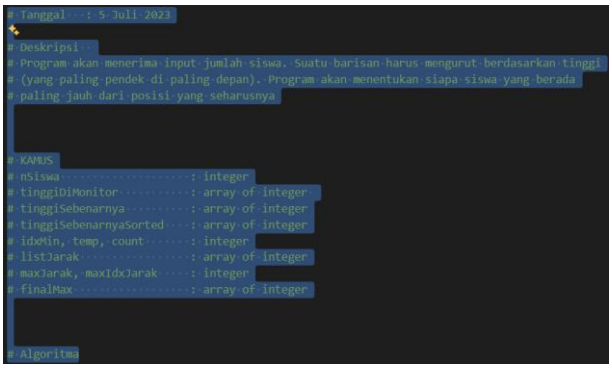
	Pesan hasil ekstraksi:
	`[PESAN RAHASIA]:`
	`Ini pesan rahasia sekali!`

.txt	Pesan asli:
	`1 ini adalah txt rahasia!`
	`2 ini adalah txt rahasia!`
	`3 v`
	`4 ini adalah txt rahasia!`
	`5 ini adalah txt rahasia!`
	`6 ini adalah txt rahasia!`
	`7 v`

	Pesan hasil eksktraksi:

	ini adalah txt rahasia! ini adalah txt rahasia! v ini adalah txt rahasia! ini adalah txt rahasia! ini adalah txt rahasia! v
.png	Pesan asli:  Pesan hasil ekstraksi: 
.jpg	Pesan asli:  Pesan hasil ekstraksi: 
.pdf	Pesan asli: Ini pesan rahasia ditulis di word! Pesan hasil ekstarksi: Invalid or corrupted PDF file. Less Information PDF.js v2.10.377 (build: 156762c48) Message: Invalid PDF structure.
.docx	Pesan asli: Ini pesan rahasia ditulis di word! Pesan hasil ekstarksi: Ini pesan rahasia ditulis di word!

Berikut adalah cuplikan *source code* yang digunakan pada pengujian program.



Gambar 4. Potongan *source code* yang dipakai sebagai medium steganografi.

Gambar 5. Potongan *source code* yang sudah disisipi pesan.

V. Analisis dan Diskusi

Berdasarkan pengujian yang telah dilakukan pada bab sebelumnya, metode steganografi yang diusulkan, yaitu manipulasi *whitespace* (spasi dan tab) serta manipulasi gaya penulisan komentar, menunjukkan hasil yang cukup bagus (kecuali pada file berformat pdf).

A. Analisis Integritas Data dan Fungsionalitas Sistem

Tujuan utama dari steganografi pada *source code* adalah menyisipkan pesan tanpa merusak logika program pembawa. Berdasarkan hasil pengujian pada Tabel 2, sistem berhasil mengekstraksi kembali berbagai format file (.txt, .docx, .png, dan .jpg) dengan baik (kecuali pada file pdf).

Keberhasilan ini disebabkan oleh sifat *interpreter* Python yang mengabaikan karakter *whitespace* di luar sintaks (seperti di akhir baris) dan mengabaikan seluruh teks yang berada di belakang tanda pagar (#) sebagai komentar.

Pada program yang dibuat, metode steganografi terbagi menjadi 2 bagian.

1. Metode spasi/tab. Penyisipan karakter ASCII 32 (spasi) dan ASCII 9 (tab) di akhir baris tidak memengaruhi eksekusi program karena *interpreter* menganggapnya sebagai *formatting* kosong. Namun, risiko muncul ketika penyisipan dilakukan secara berlebihan hingga membuat baris baru di bagian bawah kode, yang secara teknis tidak mengganggu program tetapi dapat terlihat mencurigakan jika file dibuka di *text editor*.
2. Metode gaya penulisan komentar. Perubahan gaya penulisan (misalnya dari *hello_world* menjadi *helloWorld*) hanya mengubah representasi visual string di dalam komentar. Karena komentar tidak dieksekusi oleh mesin, logika program tetap berjalan dengan lancar.

B. Analisis Overhead dan Rasio Ekspansi Ukuran File

Salah satu parameter dalam steganografi adalah efisiensi ruang atau besar pembengkakan ukuran file yang terjadi setelah pesan disisipkan.

Pada metode penggunaan spasi dan tab, setiap bit pesan (0 atau 1) direpresentasikan oleh satu karakter (1 byte). Bit

0 dipetakan menjadi spasi (1 byte) dan bit 1 dipetakan menjadi tab (1 byte).

Untuk menyembunyikan file berukuran 1 KB (1024 byte), sistem perlu menambahkan 1024×8 karakter spasi/tab, yang setara dengan penambahan ukuran file sebesar 8 KB pada *stego-object*. Jika pengguna menyisipkan gambar .jpg berukuran 1 MB, maka *source code* Python tersebut akan membengkak sebesar 8 MB hanya dari karakter spasi dan tab. Pembengkakan ini tergolong cukup besar. Hal ini menjadi indikator yang sangat jelas bagi pihak ketiga yang memantau ukuran file, meskipun secara visual kodenya terlihat pendek.

Sementara itu, metode manipulasi komentar lebih hemat ruang karakter tambahan. Sistem menggunakan 2 kata untuk merepresentasikan 3 bit. Artinya, kapasitas penyisipan sangat bergantung pada jumlah kata yang tersedia dalam komentar di *source code* asli. Metode ini tidak menambah ukuran file secara signifikan (hanya perubahan kapitalisasi atau tanda baca), namun memiliki kapasitas tampung (kapasitas untuk menyimpan pesan) yang sangat terbatas dibandingkan metode spasi/tab.

Keberhasilan ini disebabkan oleh sifat *interpreter* Python yang mengabaikan karakter *whitespace* di luar sintaks (seperti di akhir baris) dan mengabaikan seluruh teks yang berada di belakang tanda pagar (#) sebagai komentar.

C. Analisis Imperceptibility

Imperceptibility adalah kemampuan stego-object untuk tidak dibedakan dari *cover-object* oleh indra manusia [4]. Secara visual dari *text editor*, metode spasi dan *tab* memiliki tingkat *imperceptibility* yang sangat tinggi pada konfigurasi standar *text editor*. Karakter spasi dan tab sifatnya adalah *invisible* (tidak terlihat). Pihak ketiga tidak akan menyadari keberadaan ribuan karakter spasi di ujung baris kode kecuali jika mereka mengaktifkan fitur "*show invisibles/whitespaces*" pada editor mereka atau melakukan seleksi blok teks (*blocking*) yang akan memperlihatkan area kosong yang memanjang ke kanan atau di bawah. Secara linguistik, metode manipulasi penulisan gaya komentar memanfaatkan variasi gaya penulisan seperti s*nake case*, *camel case*, dan *pascal case,* yang memang umum digunakan dalam lingkungan *programmer*. Penggunaan gaya *camel case* (helloWorld) atau *snake case* (hello_world) tidak akan memancing kecurigaan karena keduanya adalah konvensi populer. Namun, terdapat kelemahan pada pemetaan biner tertentu. Berdasarkan Tabel 1, terdapat gaya penulisan yang mungkin terlihat tidak alami dalam konteks komentar kalimat bahasa manusia. *Kebab case* jarang digunakan dalam kalimat komentar standar bahasa Inggris/Indonesia, dan lebih sering digunakan di URL. Pola "‥" (titik dua kali), untuk menangani sisa kata yang hanya berjumlah satu, secara makna terlihat aneh dan dapat memancing kecurigaan linguistik jika dibaca oleh pihak ketiga secara teliti.

D. Analisis Ketahanan (Robustness)

Dalam pengembangan perangkat lunak modern, terdapat alat-alat yang secara otomatis dapat menghapus *trailing whitespace* (spasi di akhir baris). Oleh karena itu, metode spasi/*tab* dikategorikan memiliki ketahanan yang sangat rendah. Jika *source code* yang berisi pesan rahasia diformat ulang oleh IDE atau *linter*, seluruh pesan rahasia yang tersimpan di akhir baris akan terhapus permanen karena dianggap sebagai sampah (*redundant*).

Metode komentar sedikit lebih tahan (*robust*) karena *formatter code* – ekstensi di *text editor* untuk mengatur tampilan *source code* pada bahasa pemrograman tertentu – biasanya tidak mengubah isi string di dalam komentar, kecuali jika panjang barisnya melebihi batas tertentu.

E. Keamanan Kriptografis

Perlu diingat bahwa program yang dirancang ini adalah murni steganografi tanpa lapisan enkripsi tambahan. Pesan dikonversi langsung ke biner lalu ke format spasi/tab atau gaya penulisan komentar. Artinya, jika pihak ketiga menyadari metode yang digunakan (misalnya mengetahui bahwa spasi=0 dan tab=1), mereka dapat dengan mudah mengekstraksi dan membaca pesan tersebut. Untuk keamanan yang lebih baik, seharusnya pesan dienkripsi terlebih dahulu menggunakan algoritma kriptografi (seperti AES) sebelum disisipkan ke dalam *source code*.

F. Analisis Kegagalan Ekstraksi pada Format PDF

Meskipun pengujian menunjukkan keberhasilan pada format teks (.txt) dan gambar (.png, .jpg), percobaan ekstraksi pada file dokumen PDF mengalami kegagalan: *file* hasil ekstraksi tidak dapat dibuka (*corrupt*). Kegagalan ini dapat dianalisis melalui karakteristik struktur data biner file tersebut.

Data berjenis PDF memiliki struktur tertentu yang jauh lebih kompleks dibandingkan data teks biasa. Sebuah file PDF tidak hanya berisi karakter, tetapi juga *header*, *cross-reference table*, dan *trailer* yang mendefinisikan tata letak dokumen secara presisi [5].

Dalam proses steganografi yang dirancang, *file* disisipkan dengan cara dibaca dalam mode *byte* dan dikonversi menjadi string bit yang panjang. Proses ekstraksi melakukan kebalikannya: membaca bit, dikelompokkan per 8 bit, dan ditulis ulang. Kegagalan pada PDF terjadi karena sensitivitas format ini terhadap pergeseran bit [5]. Jika selama proses penyisipan (misalnya pada metode spasi/*tab*) terdapat satu karakter saja yang tidak sengaja terhapus atau tertambah – misalnya oleh ekstensi *text editor* – maka urutan bit akan bergeser.

Pada file .txt, pergeseran ini mungkin hanya mengubah satu huruf menjadi karakter acak dan pesan sisanya masih bisa terbaca. Sebaliknya, pada file .pdf, kerusakan satu bit pada bagian *header* atau struktur lainnya akan membuat seluruh file dianggap *corrupt* oleh PDF Reader dan tidak dapat dibuka sama sekali [5]. Hal ini mengonfirmasi bahwa metode steganografi ini memiliki risiko tinggi (*fragile*) untuk file biner kompleks jika medium *source code* tidak dijaga integritasnya secara ketat.

G. Rekomendasi Karakteristik Source Code

Agar metode steganografi ini berjalan lancar dan tidak memancing kecurigaan, karakteristik *source code* yang menjadi *cover-object* harus memenuhi kondisit tertentu. Berikut adalah beberapa poin yang menjadi rekomendasi.

1. Rekomendasi jumlah baris

Pada metode spasi/*tab*, jika baris pada *source code* terlalu pendek atau sedikit, program akan menghasilkan baris baru di bagian paling bawah dari *source code*. Hal ini berisiko memunculkan blok area kosong yang besar di akhir file yang sangat mencurigakan secara visual. Untuk menyisipkan pesan berukuran 1 KB (8192 bit) dengan asumsi rata-rata menyisipkan 8 bit (1 byte)/1 karakter per baris, maka

$$jumlah\ baris\ ideal = \frac{total\ bit\ pesan}{bit\ per\ baris} = \frac{8192}{8} = 1024$$

Disarankan supaya *source code* memiliki minimal 1.000 baris kode untuk setiap 1 KB pesan yang ingin disisipkan. Jika source code terlalu pendek, rasio antara kode asli dan whitespace tambahan akan menjadi tidak seimbang sehingga file source code terlihat tidak wajar.

2. Rekomendasi jumlah komentar

Metode manipulasi komentar memiliki kapasitas yang jauh lebih terbatas. Berdasarkan perancangan, setiap 2 kata pada komentar merepresentasikan 3 bit data. Jika ingin menyembunyikan sebuah kata pendek sepanjang 10 karakter (80 bit), perhitungannya adalah

$$jumlah\ kata\ diperlukan = \left(\frac{80}{3}\right) \times 2 \approx 54\ kata$$

Ini berarti, hanya untuk menyembunyikan 10 huruf, diperlukan komentar yang mengandung total 54 kata. Untuk pesan yang lebih panjang, jumlah komentar harus sangat banyak. Oleh karena itu, metode ini direkomendasikan hanya digunakan pada proyek perangkat lunak skala besar yang memiliki dokumentasi kode yang sangat besar dan metode ini tidak cocok untuk *script* pendek yang minim komentar.

H. Saran Pengembangan

Mengingat ekspansi *file* yang cukup besar pada metode spasi/*tab*, sangat disarankan untuk melakukan kompresi pada pesan sebelum proses penyisipan. Algoritma kompresi seperti Huffman Coding dapat digunakan. Ini akan mengurangi jumlah baris atau spasi yang dibutuhkan pada *source code*, sehingga meningkatkan aspek *imperceptibility* (lebih sulit dideteksi) dan efisiensi penyimpanan.

Saat ini, keamanan bergantung sepenuhnya pada kerahasiaan metode. Sebagaimana dijelaskan pada dasar teori, steganografi hanya menyembunyikan keberadaan pesan tanpa mengacak isinya, sehingga jika penyerang berhasil mengetahui tabel pemetaan biner (Tabel 1), maka seluruh informasi dapat langsung dibaca dengan mudah. Disarankan untuk menggunakan konsep enkripsi *hybrid*: menggabungkan teknik kriptografi simetris dengan metode steganografi yang telah dirancang. Penerapan algoritma enkripsi standar seperti AES-256 (*Advanced Encryption Standard*) sangat direkomendasikan untuk mengacak pesan (*payload*) sebelum dikonversi menjadi spasi, tab, atau komentar. Dengan skema *hybrid* ini, sistem akan memiliki lapisan keamanan ganda: steganografi bertugas untuk menyamarkan keberadaan pesan agar tidak memancing kecurigaan, sedangkan kriptografi bertugas melindungi kerahasiaan isi pesan seandainya metode penyembunyian tersebut berhasil dibongkar oleh pihak ketiga.

VI. KESIMPULAN

Berdasarkan perancangan, implementasi, dan pengujian yang telah dilakukan, dapat ditarik beberapa kesimpulan sebagai berikut.

1. Steganografi melalui medium *source code* dapat dilakukan dengan menyisipkan pesan ke spasi, *tab*, dan melalui gaya penulisan komentar.
2. Hasil pengujian menunjukkan bahwa program yang dibuat sudah dapat menyisipkan dan mengekstraksi pesan rahasia dalam berbagai bentuk seperti teks, .txt, .docx, .png, dan .jpg. Bentuk *file* .pdf gagal diekstraksi secara baik.
3. Batasan utama pada metode ini adalah terkait ukuran *file* dan *robustness*. Teknik manipulasi spasi/*tab* menyebabkan ekspansi *file* yang cukup besar. Beberapa ekstensi di *text editor* juga dapat secara otomatis menghapus spasi/*tab*.

LINK GITHUB

https://github.com/azrahaikal/TugasMakalah_13222045

REFERENCES

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/08-Steganografi-Bagian1-2025.pdf, diakses pada 26 Desember 2025.

[2] Por, L. Y., Wong, K., & Chee, K. O. (2008). UniSpaCh: A Text-Based Data Hiding Method Using Unicode Space Characters. Journal of Systems and Software, 81(5).

[3] https://www.freecodecamp.org/news/snake-case-vs-camel-case-vs-pascal-case-vs-kebab-case-whats-the-difference/, diakses pada 26 Desember 2025.

[4] Khalifa, A., & Guzman, A. (2022). Imperceptible Image Steganography Using Symmetry-Adapted Deep Learning Techniques. *Symmetry*, *14*(7), 1325. https://doi.org/10.3390/sym14071325.

[5] https://medium.com/@jberkenbilt/the-structure-of-a-pdf-file-6f08114a58f6, diakses pada 26 Desember 2025.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Desember 2025

Azra Haikal Fahlevi/13222045