ANALISIS IMPLEMENTASI DAN SERANGAN PADA *STEGANOGRAFI* STUDI KASUS : PERANGKAT LUNAK *"CAMOUFLAGE"*

Adi Purwanto Sujarwadi – NIM : 13506010

Perangkat lunak Studi Teknik Informatika, Institut Teknologi Bandung
Gedung Benny Subianto,Jl. Ganesha 10, Bandung
E-mail : if16010@students.if.itb.ac.id

Abstrak

Makalah ini akan membahas mengenai analisis implementasi dan serangan *Steganografi* pada perangkat lunak *Steganografi* adalah teknik menyembunyikan suatu pesan rahasia dalam pesan lain sehingga keberadaan pesan rahasia tersebut tidak dapat diketahui. *Steganografi* membutuhkan media untuk menyembunyikan pesan rahasia, adapun media yang digunakan sebagai studi kasus dalam makalah ini ialah sebuah gambar dengan format **.jpg*.

Perangkat lunak yang akan digunakan sebagai studi kasus ialah *Camouflage*, sebuah perangkat lunak *freeware* berbasis *Microsoft(r) Windows(tm)* yang diklaim memiliki tingkat keamanan yang tinggi sehingga perangkat lunak milik *National Security Agency (NSA)* tidak dapat mendeteksi keberadaan pesan rahasia yang disembunyikan.

Analisis dilakukan dengan teknik yang sederhana, membandingkan dua buah *file* yang sama sebelum dan setelah digunakan sebagai medium penyembunyian pesan. Perbandingan akan dilakukan dengan menggunakan perangkat lunak yang dapat menampilkan isi *file* dalam mode hex. Dari perbedaan antara kedua *file* tersebut akan dilakukan percobaan serangan untuk mendapatkan isi pesan yang disembunyikan.

***Kata kunci**: Steganography, software, attack, hiddentext ,covertext, stegotext, Camouflage,Steganalysis.*

1. Pendahuluan

Untuk menjaga kerahasiaan sebuah pesan, sejak berabad-abad yang lalu manusia telah menggunakan berbagai macam cara. Cara yang paling sederhana ialah dengan merubah pesan ke dalam bentuk tersandi yang tidak dapat dimengerti lagi maksudnya. Pengubahan pesan ke dalam bentuk tersandi ini kita kenal dengan nama kriptografi.

Seiring dengan perkembangan zaman, algoritma yang digunakan dalam kriptografi terus berkembang, sehingga untuk memecahkan pesan yang telah disimpan dalam bentuk tersandi pun menjadi kian sulit. Namun, kriptografi masih memiliki suatu kelemahan besar yaitu bentuk pesan tersandi terlihat aneh dan cenderung mencurigakan saat akan dikirimkan. Atas dasar hal itu, manusia mulai memikirkan sebuah cara lagi agar pesan rahasia yang dikirimkan tidak terlihat aneh dan mencurigakan. Salah satu cara untuk melakukan hal tersebut ialah dengan menyembunyikan pesan rahasia tersebut dalam pesan lainnya agar tidak terlihta mencurigakan. Cara menyembunyikan pesan seperti ini saat ini kita kenal dengan *steganografi.*

Pada dasarnya, *steganografi* dapat diartikan sebagai ilmu dan seni menyembunyikan sebuah pesan rahasia dalam pesan lain sehingga keberadaan pesan rahasia tersebut tidak diketahui.

Steganografi berasal dari bahasa Yunani, "*steganos*" yang berarti tulisan tersembunyi. Kata *steganografi* pertama kali digunakan pada tahun 1499 oleh *Johannes Tritemius* dalam bukunya, "Steganographia". Oleh kebanyakan orang, *steganografi* dipandang sebagai kelanjutan kriptografi, karena dalam prakteknya sebuah pesan biasanya dienkripsi terleih dahulu oleh pengirim pesan, lalu *ciphertext* hasil enkripsi tersbut, disembunyikan dalam media lain sehingga keberadaannya tidak disadari oleh pihak ketiga. Namun, pihak penerima pesan tentunya akan dapat mengekstraksi kembali pesan tersebut kedalam bentuk aslinya.

Dalam implementasinya, *steganografi* memerlukan dua properti yaitu pesan rahasia, dan media penampung pesan tersebut. Media yang digunakan sebagai penampung dapat berupa gambar, teks, suara, maupun video.

Adapun terminologi yang umum digunakan dalam *steganografi* antara lain :

1. *Hiddentext* : pesan yang disembunyikan
2. *Covertext* : pesan yang digunakan untuk menyembunyikan pesan rahasia, dapat juga berupa objek lain dinamakan *cover-object*.
3. *Stegotext* : Pesan yang sudah berisi pesan rahasia, dapat juga berupa objek lain dinamakan *stego-object.*

Dalam menyembunyikan pesan, ada dua metode yang umumnya digunakan dalam *steganografi* :

- Spasial

  Metode ini melakukan modifikasi langsung terhadap nilai *byte* dari *covertext*. Contoh yang termasuk kedalam metode ini ialah *LSB* (*Least Significant Bit*). LSB merupakan salah satu metode *steganografi* yang paling mudah diimplementasikan, yaitu dengan cara merubah bit yang paling kurang berarti dalam sebuah *file* . Perubahan pada LSB hanya akan mengakibatkan perubahan yang hampir tidak berarti pada *file*. Perubahan dilakukan dengan cara menaikkan nilai *bit* tersebut satu tingkat lebih tinggi dibandingkan nilai sebelumnya.

- *Transform*

  Metode ini melakukan modifikasi langsung hasil transformasi frekuensi sinyal. Contohnya adalah *Spread Spectrum*. Metode ini melibatkan konsep pemrosesan sinyal yang cukup rumit, sehingga tidak akan dibahas lebih lanjut dalam makalah ini.

Dalam era komputer seperti sekarang ini, banyak perangkat lunak yang dapat menyembunyikan data rahasia kita ke dalam data lainnya. Salah satu perangkat lunak yang tersedia secara gratis adalah *Camouflage*. Perangkat lunak ini diklaim memiliki tingkat keamanan yang tinggi sehingga perangkat lunak milik *National Security Agency (NSA)* tidak dapat mendeteksi keberadaan pesan rahasia yang disembunyikan.

2. Analisis Implementasi

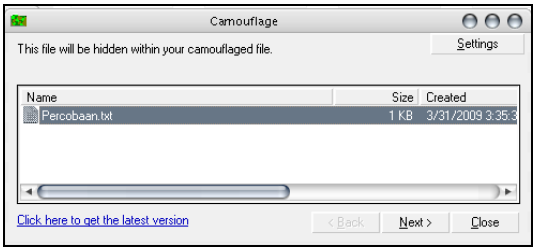
Untuk melakukan analisis terhadap perangkat lunak ini, hal yang pertama kali dilakukan adalah mencoba menyembunyikan sebuah pesan teks ke dalam sebuah citra.

Dalam ujicoba ini, digunakan media sebagai berikut :

- Pesan rahasia berupa sebuah *file* teks dengan nama "`Percobaan.txt`" dengan isi file : "`ujicoba keamanan perangkat lunak steganografi`". Ukuran dari *file* ini ialah 1 Kb.
- Media penyembunyian berupa sebuah *file* citra dengan nama "`arkavidia.jpg`". ukuran *file* sebelum disisipi pesan ialah 18.4 Kb

Gambar 1 : Kondisi awal arkavidia.jpg

Ujicoba dilakukan dengan cara menyisipkan *file* "`Percobaan.txt`" pada *file* "`arkavidia.jpg`" dengan menggunakan perangkat lunak *Camouflage*.



Gambar 2 : Proses Enkripsi

Sandi yang digunakan dalam percobaan ini adalah "kriptografi", *stego-object* akan disimpan dalam sebuah file bernama "`arkavidia_hasil.jpg`".
Setelah proses enkripsi dilakukan, didapatkan bahwa *file* "arkavidia_hasil.jpg" tampak sama

persis dengan *file* “arkavidia.jpg”. Namun jika diteliti lebih lanjut, terdapat perbedaan dimana ukuran “arkavidia_hasil.jpg” adalah 19,2 Kb, yang mana lebih besar 0,8 Kb dari *file* awal.

Gambar 3 : arkavidia.jpg setelah disusupi pesan

Untuk mendapatkan letak perbedaan tersebut, kedua *file* akan dibandingkan dalam mode *hex* dengan menggunakan perangkat lunak *CygnusHex*. Dari hasil pembacaan *file* asli, didapatkan bahwa potongan 3 kelompok *offset* terakhir dari file “arkavidia.jpg” adalah :

```
C7 D2 C6 5E 19 64 93 CE-C0 15 12 F6 C7
8B 25 74
00 2C 00 00 00 00 00 00-00 00 00 00 00
00 00 00
00 00 00 00 00 00 00 00-00 00 FF D9
```

Offset terakhir, yaitu FF D9, merupakan tanda dari akhir sebuah *file* *.jpg normal. Kemudian dilakukan pembacaan pada *file* “arkavidia_hasil.jpg” dengan hasil 3 kelompok *offset* terakhir :

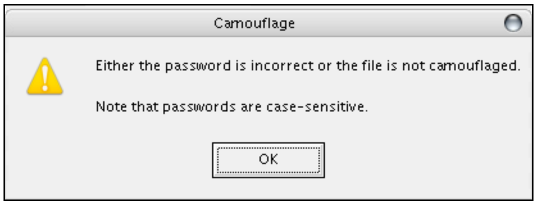
```
20 20 20 20 20 20 20 20-20 20 20 20 20
20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 74
A4 54 10
22 97 20 20 20 20 20 20-20 20 20 20 20
20 20 20
```

Ternyata hasil pembacaan *offset* terakhir pada *file* stego-object memberikan hasil yang berbeda. Dimana dari perbandingan kedua hasil, tidak ditemukan adanya karakter yang sama. Bahkan pada hasil pembacaan *file* kedua, tidak ditemukan karakter FF D9 pada akhir *file*. Setelah dilakukan pencarian, karakter FF D9 ditemukan di pertengahan *file* dengan struktur *file* sebelum FF D9 sama persis seperti struktur *file* sebelumnya. Hasil pembacaannya sebagai berikut :

```
C7 D2 C6 5E 19 64 93 CE-C0 15 12 F6 C7 8B 25 74
00 2C 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 FF D9 20 00 4F B1
C9 01 FA 61 5A 19 1B B2-C9 01 DA 7C E7 A0 4F B1
C9 01 34 35 CF 19 2D 00-00 00 77 FF 13 41 63 C4
75 C1 8A AA DE 08 41 01-FF DD B9 15 2F 21 9A 98
12 3F CC 57 ED 12 E9 47-86 97 C2 8A 39 B7 25 2F
68 AF 9F E8 7D 04 51 FF-FF FF FF 20 00 16 B2 C9
01 9E F2 CF 42 1B B2 C9-01 1C FF 8B A1 16 B2 C9
01 D2 69 BB 51 52 F0 08-41 63 C4 75 80 8F E1 CB
1D 54 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 63 E7 11 43-7A CF 70 88 80 E1 D5 15
47 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 2D 00 00 00 AC-49 00 00 02 00 69 E7 13
52 78 C9 73 93 80 A9 D6-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 74 A4 54 10
22 97 20 20 20 20 20 20-20 20 20 20 20 20 20 20
```

Data berwarna merah ialah data yang sama dengan data di akhir *file* awal, sedangkan data berwarna biru merupakan tempat penyimpanan *file* yang disisipkan, sedangkan selebihnya merupakan buffer kosong.

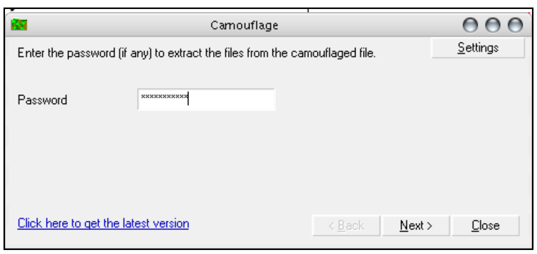
Setelah proses enkripsi dilakukan, dilakukan percobaan proses dekripsi dengan menggunakan kata sandi yang salah, dan ternyata perangkat lunak menampilkan sebuah pesan kesalahan :



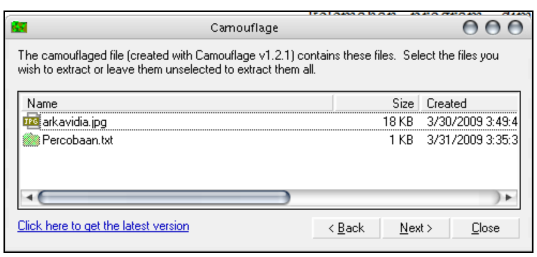
Gambar 4 : pesan kesalahan

Pesan tersebut menyatakan bahwa kata sandi salah ataupun bukan merupakan *file* yang disisipi. Menyikapi hal ini dapat ditarik sebuah asumsi yaitu perangkat lunak sebenarnya menyimpan kata sandi dalam struktur *stego-object* yang dibentuk.

Apabila proses dekripsi dilakukan dengan kata sandi yang benar, maka perangkat lunak akan menampilkan daftar nama *file* yang disisipkan beserta *file* yang digunakan sebagai media, untuk selanjutnya akan dapat diekstraksi kembali.



Gambar 5 : Proses pemasukan kata sandi



Gambar 6 : List File yang disembunyikan

3. Analisis Serangan

Dari analisis implementasi yang telah dilakukan, ada beberapa hal yang dapat disimpulkan :

- Perangkat lunak menyembunyikan pesan rahasia dengan cara melakukan penambahan pada akhir *file* media.
- Kata sandi disimpan dalam *file*

Untuk melakukan serangan, hal yang pertama kali dilakukan ialah menemukan letak *offset* penyimpanan kata sandi oleh perangkat lunak. Pencarian dilakukan dengan melakukan penyisipan *file* kembali dengan kata sandi yang berbeda dan melakukan perbandingan antar *stego-object* yang terbentuk.

Dalam ujicoba ini, digunakan media yang sama dengan percobaan analisis implementasi, namun digunakan kata sandi yang berbeda yaitu :

- “” (kata sandi dibiarkan kosong)
- “a” (kata sandi berupa satu karakter a)
- “aaaa” (kata sandi berupa empat karakter a)
- “aaaaaaa....aaaa” (kata sandi berisikan 255 karakter a)

Ternyata, setelah dibandingkan ditemukan bahwa selalu terdapat perbedaan isi setelah karakter `AC-49`, sehingga dapat ditarik kesimpulan bahwa kata sandi tersimpan disana dalam bentuk terenkripsi.

Hal yang menarik ialah dari keempat percobaan tersebut, tidak terdapat perbedaan lain selain perbedaan pada tempat penyimpanan kata sandi tersebut, hal ini mengindikasikan suatu kelemahan perangkat lunak, dimana penyisipan *file* ternyata tidak tergantung pada sandi lewat yang dimasukkan.

Hasil pembacaan *file* pertama (tanpa kata sandi):

```
20 20 20 2D 00 00 00 AC-49 00 00 02 00 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 74 A4 54 10
```

Hasil pembacaan *file* kedua(satu karakter a):

```
20 20 20 2D 00 00 00 AC-49 00 00 02 00 63 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 74 A4 54 10
```

Hasil pembacaan *file* ketiga(empat karakter a):

```
20 20 20 2D 00 00 00 AC-49 00 00 02 00 63 F4 1B
43 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 74 A4 54 10
```

Dari hasil pembacaan ketiga *file* diatas, kembali ditemukan sebuah kelemahan, dimana setiap ada perubahan 1 karakter (1 *byte*), juga akan terjadi perubahan 1 *byte* pada *file stego-object*. Hal ini sangat mengejutkan, karena jumlah karakter pada kata sandi dapat dengan mudah ditebak. Bahkan tanpa melakukan analisa lebih lanjut dapat diambil kesimpulan sementara bahwa perangkat lunak melakukan enkripsi dengan teknik sederhana yang sangat lemah. Untuk membuktikannya, akan dilakukan uji pada *file* keempat.

Hasil pembacaan *file* keempat(255 karakter a):

```
20 20 20 2D 00 00 00 AC-49 00 00 02 00 63 F4 1B
43 6D C7 75 80 80 AE DE-04 41 0E FF D2 F8 04 2B
32 9A 97 14 35 CC 42 AC-1F FD 48 86 9D 83 98 2C
B3 23 2F 67 A1 99 FB 7D-03 59 15 45 61 34 BE 20
AA 60 C3 D6 92 EE EB BC-CD 52 E2 01 48 92 19 45
5F 1B 8A B2 85 FC FC 22-F5 2B A6 24 0C 44 15 8A
6A F9 A8 1D 9D A9 DB 53-0A 61 B2 A4 A3 F5 55 CE
D1 84 F4 1C 4B E5 C5 3E-84 0F 46 4B BA F7 1F 5F
29 58 27 AE 0E 10 CB 5D-50 FB C8 FF EE E8 12 D2
58 AB 53 B4 91 50 38 1D-63 4F E7 56 98 4A 1F 30
93 20 E0 6D B5 04 74 96-11 B5 78 F9 41 DE 41 D9
34 06 AD E0 79 ED 72 5D-02 5D F3 70 85 3A 7A 69
43 01 2D 2B A4 EB D2 A4-14 A2 F1 1B 93 D3 D7 A9
B1 59 EB A3 E7 91 CD 88-AB 3D 2F 5F 68 48 19 48
F8 3B E5 B4 DB 3F B4 F3-1B 59 9B B1 01 8D 94 46
DB 8F D6 BF FE FA BF 04-B5 17 58 17 FD BB 09 EC
C9 C1 C7 7F B8 BA 6E 2C-CA F3 AC 10 74 A4 54 10
```

Ada sebuah kesimpulan lagi yang dapat kita ambil setelah mengamati perbandingan keempat hasil pembacaan *file* tersebut, yaitu panjang kata sandi maksimal ialah 255 byte, dimana kata sandi terletak diantara **AC-49 00 00 02 00** dan **74 A4 54 10**.

Dari percobaan sebelumnya, kita telah mengetahui bahwa fungsi enkripsi yang dimiliki oleh perangkat lunak ini tidak bergantung kepada kata sandi yang digunakan. Oleh karena itu, kemungkinan besar kata sandi tersebut hanya disisipkan untuk mencegah file yang disembunyikan diekstraksi oleh pihak ketiga. Dengan telah ditemukannya tempat penyisipan kata sandi, untuk langkah selanjutnya akan dicari fungsi enkripsi penyimpanan kata sandi ini sendiri.

Karena hanya disisipkan saja, diambil asumsi bahwa kata sandi dienkripsi dengan menggunakan fungsi XOR, asumsi ini diambil karena fungsi XOR merupakan fungsi yang lazim digunakan dalam melakukan enkripsi sebuah file. Mengingat bahwa fungsi XOR adalah fungsi simetri, tentunya apabila kita melakukan XOR antara hasil enkripsi dengan kata sandi sebenarnya, kita akan dapat menemukan fungsi enkripsi yang digunakan.

Percobaan pembuktian asumsi tersebut akan dilakukan dengan cara melakukan XOR terhadap hasil pembacaan file keempat, dengan kata sandi aslinya, yaitu 255 buah karakter "a", yang mana dalam *Hex* dapat dinyatakan sebagai "61", sehingga hasil pembacaan *file* tersebut akan di XOR dengan 255 buah *Hex* "61"(61616161…)

Hasil dari XOR tersebut adalah :

```
20 20 20 2D 00 00 00 AC-49 00 00 02 00 02 95 7A
22 0C A6 14 E1 E1 CF BF 65 20 6F 9E B3 99 65 4A
53 FB F6 75 54 AD 23 CD 7E 9C 29 E7 FC E2 F9 4D
D2 42 4E 06 C0 F8 9A 1C 62 38 74 24 00 55 DF 41
CB 01 A2 B7 F3 8F 8A DD AC 33 83 60 29 F3 78 24
3E 7A EB D3 E4 9D 9D 43 94 4A C7 45 6D 25 74 EB
0B 98 C9 7C FC C8 BA 32 6B 00 D3 C5 C2 94 34 AF
B0 E5 95 7D 2A 84 A4 5F E5 6E 27 2A DB 96 7E 3E
48 39 46 CF 6F 71 AA 3C 31 9A A9 9E 8F 89 73 B3
39 CA 32 D5 F0 31 59 7C 02 2E 86 37 F9 2B 7E 51
F2 41 81 0C D4 65 15 F7 70 D4 19 98 20 BF 20 B8
55 67 CC 81 18 8C 13 3C 63 3C 92 11 E4 5B 1B 08
22 60 4C 4A C5 8A B3 C5 75 C3 90 7A F2 B2 B6 C8
D0 38 8A C2 86 F0 AC E9 CA 5C 4E 3E 09 29 78 29
99 5A 84 D5 BA 5E D5 92 7A 38 FA D0 60 EC F5 27
BA EE B7 DE 9F 9B DE 65 D4 76 39 76 9C DA 68 8D
A8 A0 A6 1E D9 DB 0F 4D AB 92 CD 71 74 A4 54 10
```

Menurut asumsi awal, hasil XOR ini merupakan fungsi enkripsi penyimpanan kata sandi, oleh karena itu untuk membuktikannya akan kita gunakan hasil pembacaan *file* dengan kata sandi "kriptografi". Kata sandi tersimpan dari *file* tersebut adalah :

```
69 E7 13 52 78 C9 73 93 80 A9 D6
```

Kita telah mengetahui bahwa panjang kata sandi yang digunakan akan sama dengan panjang kata sandi yang disisipkan, oleh karena itu kata sandi tersimpan tersebut akan kita XOR kan dengan 11 byte awal fungsi kata sandi yaitu :

```
02 95 7A 22 0C A6 14 E1 E1 CF BF
```

Hasil XOR dari kedua *hex* tersebut adalah :

```
6B 72 69 70 74 6F 67 72 61 66 69
```

Yang bila diterjemahkan ke dalam bentuk *string* adalah "kriptografi":

- 6B : K
- 72 : R
- 69 : I
- 70 : P
- 74 : T
- 6F : O
- 67 : G
- 72 : R
- 61 : A
- 66 : F
- 99 : I

Agar hasilnya dapat lebih terjamin, dilakukan pengujian kembali dengan menggunakan media yang sama, namun kali ini kata sandi yang digunakan ialah : "informatika135".

Pembacaan bagian penyimpanan kata sandi *file stego-object* dengan kata sandi "informatika135" adalah :

```
6B FB 1C 4D 7E CB 75 95 88 A4 DE 54 13 5A
```

Karena kata sandi tersimpan berjumlah 14 block byte, maka akan dilakukan XOR dengan 14 block awal dari kunci yang telah didapatkan, yaitu :

```
02 95 7A 22 0C A6 14 E1 E1 CF BF 65 20 6F
```

Hasil XOR dari kedua *hex* tersebut adalah :

```
69 6E 66 6F 72 6D 61 74 69 6B 61 31 33 35
```

Yang bila diterjemahkan kedalam bentuk string akan menjadi "informatika135" yang sama dengan kata sandi yang digunakan.

Kedua percobaan tersebut telah berhasil membuktikan asumsi awal, sehingga untuk perangkat lunak *Camouflage* ini, fungsi kunci XOR akan selalu sama apapun kata sandi yang digunakan. Sehingga dapat ditarik kesimpulan langkah-langkah serangan terhadap perangkat lunak *Camouflage* adalah:

- Lakukan pembacaan *file stego-object* dengan menggunakan *hex editor*
- Lakukan pencatatan *block byte* yang terletak antara block **AC–49 00 00 02 00** dan **74 A4 54 10**.
- XOR kan *block byte* tersebut dengan fungsi kunci :

```
02 95 7A 22 0C A6 14 E1 E1 CF BF 65 20 6F
9E B3 99 65 4A 53 FB F6 75 54 AD 23 CD 7E
9C 29 E7 FC E2 F9 4D D2 42 4E 06 C0 F8 9A
1C 62 38 74 24 00 55 DF 41 CB 01 A2 B7 F3
8F 8A DD AC 33 83 60 29 F3 78 24 3E 7A EB
D3 E4 9D 9D 43 94 4A C7 45 6D 25 74 EB 0B
98 C9 7C FC C8 BA 32 6B 00 D3 C5 C2 94 34
AF B0 E5 95 7D 2A 84 A4 5F E5 6E 27 2A DB
96 7E 3E 48 39 46 CF 6F 71 AA 3C 31 9A A9
9E 8F 89 73 B3 39 CA 32 D5 F0 31 59 7C 02
2E 86 37 F9 2B 7E 51 F2 41 81 0C D4 65 15
F7 70 D4 19 98 20 BF 20 B8 55 67 CC 81 18
8C 13 3C 63 3C 92 11 E4 5B 1B 08 22 60 4C
4A C5 8A B3 C5 75 C3 90 7A F2 B2 B6 C8 D0
38 8A C2 86 F0 AC E9 CA 5C 4E 3E 09 29 78
29 99 5A 84 D5 BA 5E D5 92 7A 38 FA D0 60
EC F5 27 BA EE B7 DE 9F 9B DE 65 D4 76 39
76 9C DA 68 8D A8 A0 A6 1E D9 DB 0F 4D AB
92 CD 71
```

- Lakukan konversi hasil XOR ke dalam bentuk string
- Hasil dari konversi tersebut ialah kata sandi yang digunakan

4. Kesimpulan

Perangkat lunak *Camouflage* dapat menyembunyikan sebuah data rahasia dalam berkas lain, sehingga tidak terlihat mencurigakan.

Penyembunyian data dilakukan dengan cara mengacak data lalu menyisipkannya pada akhir dari *file* media yang digunakan.

Untuk pemakaian sehari-hari, Camouflage merupakan perangkat lunak yang cukup baik untuk digunakan sebagai penyembunyi data rahasia karena kemampuannya menyisipkan data rahasia tanpa melakukan perubahan yang berarti pada *file* yang digunakan sebagai media.

Untuk keperluan penyembunyian data yang sangat penting, penggunaan *Camouflage* tidak disarankan , hal ini mengingat tingkat keamanan pada perangkat lunak *Camouflage* masih cukup rendah.

Kelemahan yang terdapat pada perangkat lunak *Camouflage* ini adalah :

- Fungsi pengacakan data yang sederhana, hanya menggunakan fungsi XOR tanpa melibatkan kata sandi sama sekali.
- Penyimpanan kata sandi dalam *stego-object* tanpa perlindungan yang memadai, sehingga mudah ditemukan.
- Pertambahan ukuran *stego-object* yang selalu sama untuk panjang kata sandi berapapun.

Kelemahan program ini sendiri baru akan muncul apabila sebuah media telah terdeteksi sebagai *stego-object*, yang mana pada prakteknya sulit untuk mendeteksi bahwa sebuah *file* merupakan *stego-object*.

DAFTAR PUSTAKA

[1] Johnson, Neil. "Steganalysis of Images Created Using Current Steganography Software". George Mason University http://www.jjtc.com/ihws98/jjgmu.html Tanggal Akses : 29 Maret 2009

[2] Westfield, Andreas. "Attacks on Steganographic Systems". Dresden University of Technology. www.ece.cmu.edu/~adrian/487-s06/westfeld-pfitzmann-ihw99.pdf Tanggal Akses : 31 Maret 2009

[3] Munir, Rinaldi, "Kriptografi", Institut Teknologi Bandung, 2009.

[4] Camouflage Official Site http://camouflage.unfiction.com/ Tanggal Akses : 12 Maret 2009

[5] Analyzing steganography softwares http://www.guillermito2.net/stegano/ Tanggal Akses : 12 Maret 2009

[6] HEX Logic Table http://www.garykessler.net/library/byte_logic_table.html Tanggal Akses : 1 April 2009

[7] ASCII Table http://www.asciitable.com/ Tanggal Akses : 1 April 2009