STUDI MENGENAI APLIKASI STEGANOGRAFI *CAMOUFLAGE* BESERTA PEMECAHAN ALGORITMANYA

Dini Armyta – NIM : 13503025

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13025@students.if.itb.ac.id

Abstrak

Steganografi adalah ilmu dan seni menyembunyikan pesan rahasia di dalam pesan lain sehingga keberadaan pesan rahasia tersebut tidak dapat diketahui secara kasat mata. Secara umum, steganografi dapat dikatakan sebagai suatu teknik yang digunakan untuk menyimpan data di dalam data lainnya. Penggunaan steganografi dapat dilakukan untuk berbagai jenis data, seperti citra, audio, atau bahkan video. Beberapa metode dapat digunakan untuk menyimpan informasi dalam berbagai jenis data tersebut, antara lain metode LSB (*least significant byte*), *spread spectrum*, ataupun *injection*.

Saat ini telah terdapat banyak aplikasi yang diciptakan untuk memfasilitasi penggunaan steganografi, bahwa steganografi tidak hanya diperuntukkan untuk para ahli namun juga dapat digunakan secara luas oleh masyarakat awam. Salah satu contoh aplikasi steganografi yang ada saat ini adalah *Camouflage*. *Camouflage* memungkinkan pengguna komputer untuk menjaga keamanan arsip-arsip personal yang dimilikinya tetap aman dari pengganggu. *Camouflage* memungkinkan pengguna untuk menyembunyikan arsip dengan mengacaknya dan menyisipkannya ke dalam arsip lain yang dipilih. Arsip yg telah di-*camouflage* akan tetap tampak dan berlaku seperti arsip normal lainnya, dan dapat disimpan atau dikirim tanpa menimbulkan kecurigaan apapun.[1]

Makalah ini akan membahas sekelumit pengertian mengenai steganografi, pengenalan terhadap aplikasi *Camouflage* berikut dengan cara penggunaannya, pembahasan mengenai bentuk data yang telah dimanipulasi oleh *Camouflage*, dan pemecahan algoritma yang digunakan oleh *Camouflage* dalam menyembunyikan informasi ke dalam data yang ada. Pembahasan akan diakhiri dengan suatu studi kasus, memecahkan suatu data yang telah di-*camouflage* untuk ditemukan pesan rahasia yang tersembunyi di dalamnya, dan juga terdapat kesimpulan untuk merangkum keseluruhan isi dari makalah ini.

Kata kunci: *Steganography application*, *Camouflage*, *Cracking Camouflage*

1. Pendahuluan

Seperti telah dijelaskan sebelumnya, steganografi adalah ilmu dan seni untuk menyembunyikan pesan rahasia di dalam pesan lain sehingga keberadaan pesan rahasia tersebut tidak dapat diketahui. Steganografi dapat dipandang sebagai kelanjutan kriptografi dan dalam prakteknya pesan rahasia dienkripsi terlebih dahulu, kemudian cipherteks disembunyikan di dalam media lain sehingga pihak ketiga tidak menyadari keberadaannya. Pesan rahasia yang disembunyikan dapat diekstraksi kembali persis sama seperti aslinya. Terdapat beberapa teknik dalam melakukan penyembunyian data menggunakan steganografi, dan sejumlah perangkat lunak telah tersedia, antara lain yang banyak digunakan yaitu metode LSB (*least significant byte)*, *spread spectrum*, kunci publik steganografi, domain transformasi, dan *embedding/injection*.

Dari sejumlah aplikasi yang diciptakan untuk memfasilitasi penggunaan steganografi, salah satunya adalah *Camouflage*. *Camouflage* adalah suatu aplikasi steganografi yang memungkinkan pengguna komputer untuk menjaga keamanan dari arsip-arsip yang dimilikinya dari pihak yang tidak bertanggung jawab. Sesuai dengan bidangnya, *Camouflage* melakukan penyembunyian arsip dengan teknik steganografi yaitu menyembunyikan suatu arsip rahasia ke dalam arsip lainnya. Sebagai contoh, pengguna dapat membuat suatu arsip

gambar yang tampak seperti arsip normal lainnya namun sebetulnya terdiri dari suatu arsip enkripsi yang tersembunyi. Untuk keamanan tambahan, *Camouflage* juga menyediakan *password* dalam penyembunyian arsip. *Password* tersebut kemudian akan dibutuhkan saat akan mengekstraksi arsip yang bersangkutan menjadi arsip normal yang telah di-*uncamouflage*. [1] Seperti aplikasi lainnya, *Camouflage* juga memiliki suatu teknik tertentu dalam melakukan penyisipan data ke dalam data lainnya.

Dalam bab-bab selanjutnya akan dibahas mengenai ulasan singkat steganografi, seperti sejarah steganografi, kegunaan steganografi, dan metode-metode yang digunakan dalam steganografi. Selain itu juga akan diulas mengenai aplikasi *Camouflage* itu sendiri, baik dari cara penggunaannya hingga pemecahan algoritma yang digunakan oleh *Camouflage*. Dari diketahuinya algoritma yang digunakan oleh *Camouflage*, maka akan dapat pula diketahui metode apa yang digunakan oleh *Camouflage* dalam menyembunyikan informasi.

2. Steganografi

2.1. Pengertian Steganografi

Steganografi adalah suatu teknik untuk menyembunyikan informasi yang bersifat pribadi dengan sesuatu yang hasilnya akan tampak seperti informasi normal lainnya. Steganografi biasanya sering disalahkaprahkan dengan kriptografi karenanya keduanya sama-sama bertujuan untuk melindungi informasi yang berharga. Perbedaan yang mendasar antara keduanya yaitu steganografi berhubungan dengan informasi tersembunyi sehingga tampak seperti tidak ada informasi tersembunyi sama sekali. Jika seseorang mengamati objek yang menyimpan informasi tersembunyi tersebut, ia tidak akan menyangka bahwa terdapat pesan rahasia dalam objek tersebut, dan karenanya ia tidak akan berusaha memecahkan informasi (dekripsi) dari objek tersebut.

Kata steganografi berasal dari bahasa Yunani, yaitu dari kata *Steganós* (tersembunyi) dan *Graptos* (tulisan). Steganografi di dunia modern biasanya mengacu pada informasi atau suatu arsip yang telah disembunyikan ke dalam suatu arsip citra *digital*, audio, atau video. Satu hal esensial yang menjadi kelebihan steganografi adalah kemampuannya untuk menipu persepsi manusia, manusia tidak memiliki insting untuk mencurigai adanya arsip-arsip yang memiliki informasi yang tersembunyi di dalamnya, terutama bila arsip tersebut tampak seperti arsip normal lainnya. Namun begitu terbentuk pula suatu teknik yang dikenal dengan *steganalysis*, yaitu suatu teknik yang digunakan untuk mendeteksi penggunaan steganografi pada suatu arsip. Seorang *steganalyst* tidak berusaha untuk melakukan dekripsi terhadap informasi yang tersembunyi dalam suatu arsip, yang dilakukan adalah berusaha untuk menemukannya. Terdapat beberapa cara yang dapat digunakan untuk mendeteksi steganografi seperti melakukan pengamatan terhadap suatu arsip dan membandingkannya dengan salinan arsip yang dianggap belum direkayasa, atau berusaha mendengarkan dan membandingkan perbedaannya dengan arsip lain bila arsip tersebut adalah dalam bentuk audio.[11]

2.2. Sejarah Steganografi

Seperti kriptografi, penggunaan steganografi sebetulnya telah digunakan berabad-abad yang lalu bahkan sebelum istilah steganografi itu sendiri muncul. Berikut adalah contoh penggunaan steganografi di masa lalu:

1. Selama terjadinya Perang Dunia ke-2, tinta yang tidak tampak (*invisible ink*) telah digunakan untuk menulis informasi pada lembaran kertas sehingga saat kertas tersebut jatuh di tangan pihak lain hanya akan tampak seperti lembaran kertas kosong biasa. Cairan seperti air kencing (*urine*), susu, vinegar, dan jus buah digunakan sebagai media penulisan sebab bila salah satu elemen tersebut dipanaskan, tulisan akan menggelap dan tampak melalui mata manusia.
2. Pada sejarah Yunani kuno, masyarakatnya biasa menggunakan seorang pembawa pesan sebagai perantara pengiriman pesan. Pengirim pesan tersebut akan dicukur rambutnya, untuk kemudian dituliskan suatu pesan pada kepalanya yang sudah botak. Setelah pesan dituliskan, pembawa pesan harus menunggu hingga rambutnya tumbuh kembali sebelum dapat mengirimkan pesan kepada pihak penerima. Pihak penerima kemudian akan mencukur rambut pembawa pesan tersebut untuk melihat pesan yang tersembunyi.
3. Metode lain yang digunakan oleh masyarakat Yunani kuno adalah dengan menggunakan lilin sebagai media penyembunyi pesan mereka. Pesan dituliskan pada suatu lembaran, dan

lembaran tersebut akan ditutup dengan lilin untuk menyembunyikan pesan yang telah tertulis. Pihak penerima kemudian akan menghilangkan lilin dari lembaran tersebut untuk melihat pesan yang disampaikan oleh pihak pengirim.

2.3. Kegunaan Steganografi

Seperti perangkat keamanan lainnya, steganografi dapat digunakan untuk berbagai macam alasan, beberapa diantaranya untuk alasan yang baik, namun dapat juga untuk alasan yang tidak baik. Untuk tujuan legitimasi dapat digunakan pengamanan seperti citra dengan *watermarking* dengan alasan untuk perlindungan *copyright*. *Digital watermark* (yang juga dikenal dengan *fingerprinting*, yang dikhususkan untuk hal-hal menyangkut *copyright*) sangat mirip dengan steganografi karena menggunakan metode penyembunyian dalam arisp, yang muncul sebagai bagian asli dari arsip tersebut dan tidak mudah dideteksi oleh kebanyakan orang. Steganografi juga dapat digunakan sebagai cara untuk membuat pengganti suatu nilai *hash* satu arah (yaitu pengguna mengambil suatu masukan panjang variabel dan membuat sebuah keluaran panjang statis dengan tipe string untuk melakukan verifikasi bahwa tidak ada perubahan yang dibuat pada variabel masukan yang asli). Selain itu juga, steganografi dapat digunakan sebagai *tag-notes* untuk citra *online*. Terakhir, steganografi juga dapat digunakan untuk melakukan perawatan atas kerahasiaan informasi yang berharga, untuk menjaga data tersebut dari kemungkinan sabotasi, pencuri, atau dari pihak yang tidak berwenang.

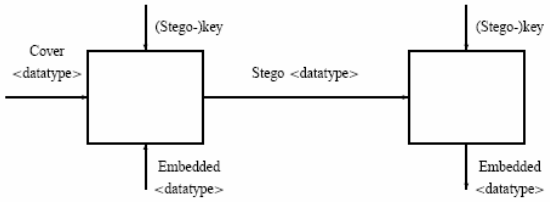
Sayangnya, steganografi juga dapat digunakan untuk alasan yang ilegal. Sebagai contoh, jika seseorang telah mencuri data, mereka dapat menyembunyikan arsip curian tersebut ke dalam arsip lain dan mengirimkannya keluar tanpa menimbulkan kecurigaan siapapun karena tampak seperti *email* atau arsip normal. Selain itu, seseorang dengan hobi menyimpan pornografi, atau lebih parah lagi, menyimpannya dalam *hard disk*, mereka dapat menyembunyikan hobi buruk mereka tersebut melalui steganografi. Begitu pula dengan masalah terorisme, steganografi dapat digunakan oleh para teroris untuk menyamarkan komunikasi mereka dari pihak luar. [8]

2.4. Metode Steganografi

Terdapat banyak metode yang digunakan dalam melakukan penyembunyian data ke dalam data lainnya. Berikut adalah penjelasan mengenai beberapa metode yang banyak digunakan dalam steganografi.

2.4.1. Metode *Embedding*

Steganografi menyimpan pesan rahasia dalam suatu arsip yang biasanya diparameterisasi oleh suatu kunci-stego, dan pendeteksian atau pembacaan atas informasi tersembunyi tersebut dapat dilihat pada gambar di bawah ini.



Gambar 1. Proses steganografi dengan metode *embedding*

Metode *embedding* ini juga biasa disebut dengan metode *injection* karena pesan rahasia "disuntikkan" langsung pada arsip lainnya dengan sedikit pengacakan atau enkripsi. [9]

2.4.2. *Fingerprinting* dan *Watermarking*

Saat ini steganografi semakin penting peranannya terutama dalam industri publikasi dan *broadcasting*, dimana tanda *copyright* yang tersembunyi atau nomer serial sangat dibutuhkan dalam film *digital*, foto, dan produk multimedia lainnya. Beberapa aplikasi steganografi dapat memindai Internet, dan mendeteksi adanya salinan dari suatu citra yang spesifik, atau adanya modifikasi terhadap citra yang dipublikasikan tersebut – maka penggunaan ilegal atas citra yang memiliki *copyright* dapat dideteksi.

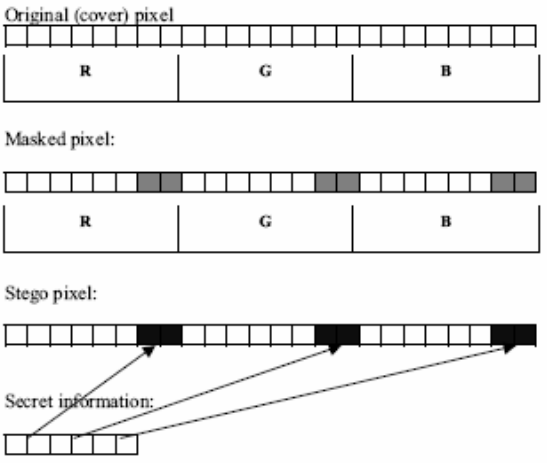
Pada kasus arsip audio, pengawasan otomatis atas iklan radio sangat dimungkinkan, pemilik iklan dapat secara otomatis menghitung berapa banyak suatu iklan tertentu ditransmisikan oleh suatu stasiun radio. Kemungkinan lain dari aplikasi steganografi dalam citra adalah untuk menyisipkan pesan tertentu dan informasi lainnya pada gambar tersebut sehingga pesan rahasia dapat disembunyikan dengan sempurna dan tidak menimbulkan kecurigaan siapapun.

Saat tujuan keamanan ditujukan untuk hak intelektual, maka banyak digunakan *fingerprinting* dan *watermarking*. *Watermarking* memungkinkan adanya informasi *copyright* yang tersimpan dalam media *digital* dan media ini dapat disebarkan luas kepada para pengguna dengan aman. Sedangkan *fingerprinting* memungkinkan penyimpanan tanda terpisah atas suatu salinan dari media *digital*, informasi tersembunyi ini biasanya merupakan suatu nomer serial yang dapat dideteksi oleh penyalur resmi dari media tersebut. [9]

2.4.3. Metode *Least Significant Bit*

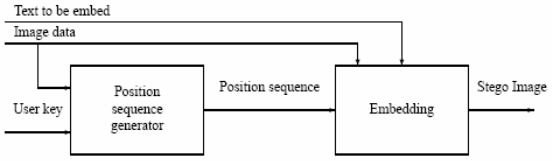
Biasanya arsip 24-bit atau 8-bit digunakan untuk menyimpan citra *digital*. Representasi warna dari pixel-pixel dapat diperoleh dari warna-warna primer yaitu merah, hijau, dan biru. Citra 24-bit menggunakan 3 bytes untuk masing-masing pixel, dimana setiap warna primer direpresentasikan dengan ukuran 1 byte. Penggunaan citra 24-bit memungkinkan setiap pixel direpresentasikan dengan nilai warna sebanyak 16.777.216 macam. Dua bit dari saluran warna ini dapat digunakan untuk menyembunyikan data, yang akan mengubah jenis warna untuk pixelnya menjadi 64-warna, namun hal ini akan mengakibatkan sedikit perbedaan yang dapat dideteksi secara kasat mata oleh manusia. Metode sederhana ini disebut dengan *Least Significant Bit* (LSB). [5] [10]

Dengan penggunaan metode ini, dimungkinkan adanya penambahan sejumlah besar informasi tanpa adanya degradasi tampilan dari citra itu sendiri. Gambar di bawah ini menunjukkan proses kerja LSB.



Gambar 2. Proses steganografi dengan metode *Least Significant Bit* (LSB)

Beberapa versi dari metode LSB telah bermunculan. Kini sangat memungkinkan untuk menggunakan menggunakan pembangkit nomer acak yang diinisialisasi dengan kunci-stego dan mengkombinasikan keluarannya dengan data masukan, dan kemudian menyembunyikannya dalam suatu arsip citra. Kehadiran seorang pengawas tidak cukup untuk meyakinkan keberhasilan penambahan sebuah pesan di lokasi tertentu (pada rentetan bit tertentu), karena pengawas itu sendiri sangat mungkin mengubah letak dari pesan rahasia tersebut, bahkan walaupun ia tidak mengetahui lokasi dari pesan rahasia tersebut atau ia tidak dapat membacanya karena telah dienkripsi. Karena itulah penggunaan kunci-stego menjadi penting karena keamanan atas suatu system proteksi tidak dapat didasarkan pada kerahasiaan dari algoritmanya itu sendiri, namun karena adanya keberadaan dari suatu kunci rahasia. [16] Gambar di bawah ini menunjukkan proses tersebut.



Gambar 3. Proses penggunaan kunci-stego pada steganografi

Metode LSB pada umumnya beroperasi pada citra *bitmap*. Data yang disembunyikan tidak dapat dikategorikan sebagai *watermark* karena bahkan jika terjadi perubahan kecil pada citra tersebut (pemotongan, kompresi, atau degradasi warna) maka informasi tersembunyi tersebut akan hila – walaupun perubahan yang terjadi selama proses *embedding* adalah tidak terlihat. [9]

2.4.4. Metode Kunci Publik Steganografi

Cara lain yang mungkin dalam melakukan steganografi adalah suatu algoritma yang membutuhkan kehadiran suatu *shared* kunci rahasia untuk memodifikasi pixel dari arsip yang akan disembunyikan tersebut. Pada kasus ini, kedua pihak baik pengirim maupun penerima harus memiliki kunci rahasia ini. Misalkan pihak-pihak tersebut tidak mencapai kesepakatan mengenai kunci rahasia yang akan digunakan, namun salah satu dari mereka (misal Bob) memiliki sepasang kunci privat/publik, dan rekannya mengetahui kunci public tersebut. Pada kasus Alice sebagai pengawas pasif yang mengetahui kunci publik Bob

dan mengenkripsi pesannya dan menyembunyikannya pada suatu lokasi tertentu pada suatu media citra, yang lalu mengirimnya kepada Bob agar dipecahkan. Bob tidak yakin atas lokasi citra yang menyimpan pesan tersembunyi tersebut, namun ia dapat mencoba untuk mendekrip rentetan *string* acak dengan kunci privatnya, dan mengecek apakah menghasilkan suatu pesan atau tidak. [5]

Pendekatan lainnya yaitu dengan menggunakan *cover image escrow scheme* (atau ekstraksi sumber), dimana sebuah ekstraktor dibutuhkan dengan citra asli, dan citra tersebut telah dikurangi dari citra stego sebelum ekstraksi dari informasi tersembunyi. Pada skema ini, pengguna tidak dapat membaca pesan yang tersembunyi, hal ini hanya dapat dimungkinkan jika memiliki sejumlah gambar asli yang mengalami modifikasi. Namun algoritma ini dikategorikan sebagai algoritma yang kuat dalam menghadapi distorsi-distorsi sinyal.

2.4.5. Metode Domain Transformasi

Algoritma ekstraksi tujuan dapat dibagi menjadi 2 grup, yaitu teknik domain ruang/waktu dan domain transformasi. Untuk kasus domain ruang diterapkan pada materi citra, dan untuk domain waktu diterapkan pada materi audio. Metode domain transformasi dioperasikan pada *Discrete Cosine Transform*, Fourier atau domain transformasi dari sinyal. [6] [16] [3]

Algoritma *Patchwork* (yang dikembangkan di MIT) memilih sejumlah pasang pixel acak, dan meningkatkan *brightness* dari pixel yang terang, dan menurunkan *britghtness* tersebut untuk pixel yang gelap. Algoritma ini menunjukkan ketahanan tinggi atas semua modifikasi citra nirgeometrik. Jika dianggap penting untuk menyediakan proteksi atas serangan penyaringan, maka kapasitas penyimpanan informasi tersebut dapat dibatasi. [5]

Citra dengan kualitas *high-color* yang dikompres biasanya akan menggunakan metode *lossy compression*, sebagai contoh untuk citra JPEG. Algoritma JPEG akan pertama-tama melakukan transformasi pixel menjadi ruang *luminance-chrominance*. *Chrominance* ini lalu di-*downsampled* – hal ini mungkin karena HVS (*human vision system*) jauh kurang sensitif atas *chrominance* dibandingkan dengan perubahan *luminance* – sehingga ukuran dari data menjadi berkurang. *Discrete Cosine Transform* lalu diaplikasikan pada kelompok pixel dengan ukuran 8 x 8. Langkah berikutnya akan menyebabkan kehilangan paling banyak untuk kasus JPEG, dimana koefisien-koefisiennya dihitung secara scalar (hal ini mungkin karena jika dilakukan reduksi atas koefesien dari frekuensi yang lebih tinggi menjadi nol, perubahan yang terjadi pada citra asli hanya akan menyebabkan perubahan yang sangat tidak mencolok dan tidak mampu dideteksi oleh kemampuan mata manusia). Langkah terakhir yaitu *lossless*, yaitu bila pengurangan terhadap koefesien ini juga dikompresi dan sebuah *header* ditambahkan pada arsip citra JPEG.

Aplikasi steganografi pada umumnya beroperasi setelah langkah perhitungan, sebagai contoh adalah aplikasi Jpeg-Jste, dan SysCoP. SysCoP menggunakan generator rentetan posisi. Masukan dari generator tersebut adalah data citra dan sebuah kunci eksternal, dan keluaran yang dihasilkan adalah suatu rentetan posisi yang dapat dipilih untuk menentukan posisi blok tempat pesan rahasia disembunyikan. [3] [15]

Pada kasus ini, bloknya terdiri dari pixel dengan ukuran 8 x 8, yang dapat *contigous* – blok adalah suatu kotak dalam citra – atau terdistriubsi, dimana pixel akan dipilih secara acak. Sebuah bit label akan ditambahkan melalui pengaturan atas hubungan spesifik antara ketiga elemen kuantitas dari suatu blok, dan algoritma yang mengandung suatu mekanisme pengecekan untuk mengetes apakah blok yang asli mampu atau tidak untuk menyimpan informasi tersebut, dan berapa banyak modifikasi yang dibutuhkan untuk menyimpan 1 bit informasi diantara pixel-pixelnya.

Metode popular pada domain frekuensi yaitu dengan memodifikasi ukuran relative dari 2 atau lebih koefesien DCT pada blok citra, dan menambahkan 1 bit informasi pada tiap blok. Kedua koefisien tersebut harus berkorespondensi dengan fungsi cosine dan dengan frekuensi tengah yang berarti informasi disimpan pada suatu bagian signifikan dari suatu sinyal. Algoritma yang digunakan harus kebal terhadap kompresi JPEG, sehingga koefisien DCT dengan nilai kuantitas yang sama harus dipilih, sesuai dengan kuantitas tabel dari JPEG.

Pada domain frekuensi, proses penambahan informasi biasanya mampu menyimpan lebih sedikit informasi ke dalam citra, tidak ada batas yang pasti atas ukuran dari objek yang ditambahkan seperti pada kasus LSB, dimana jumlah dari pixel dan kedalaman warna ditentukan oleh ukuran maksimum dari data tambahan

tersebut (dan tentunya perubahan yang terjadi selama penambahan informasi akan tidak terlihat). Pada kasus operasi domain transformasi, proses penambahan informasi dapat tampak jika ukuran data yang ditambahkan terlalu besar, dan batas yang diberikan atas ukuran data tambahan yang tidak akan mengubah properti visual dari citra tersebut adalah *image dependent*.

Berikut adalah gambar yang menunjukkan hasil dari proses *embedding* pada domain transformasi.

Gambar 4. Citra 30 KB dengan data tersembunyi bernilai "jhps"

Gambar 5. Citra 50 KB dengan data tersembunyi bernilai "jhps"

Gambar 6. Citra 60 KB dengan data tersembunyi bernilai "jhps"

Pada kasus jika gambar jam dengan ukuran 50 KB yang berisi data kemudian memodifikasi property *visible* dari citra, maka saat citra-stego dibandingkan dengan citra asli akan memungkinkan untuk dideteksi adanya perbedaan.

3. *Camouflage*

3.1. Mengenai *Camouflage*

Pada zaman sekarang ini, perusahaan-perusahaan telah diberi kuasa lebih untuk memonitor dan memeriksa arsip-arsip personal pegawainya. Dan dengan semakin menjamurnya perangkat lunak *spy* dengan tujuan tidak baik, pengguna komputer semakin membutuhkan keamanan untuk menjaga arsip-arsip yang mengandung informasi sensitif jauh dari pihak-pihak yang tidak berkepentingan. Keamanan elektronik juga sudah tidak dapat lagi dijamin – siapa yang bisa mengetahui apabila terdapat pihak yang mengintai *email* atau memindai *hard drive* tanpa sepengetahuan orang yang bersangkutan?

Alasan tersebutlah yang mendorong terbentuknya aplikasi *Camouflage* ini. *Camouflage* memungkinkan pengguna komputer untuk menyembunyikan arsip dengan mengacaknya dan melampirkannya ke dalam arsip lain. Arsip yang telah di-*camouflage* akan tampak dan berlaku seperti arsip normal lainnya, dan dapat disimpan ataupun dikirimkan sebagai *email* tanpa menarik perhatian. Pengguna bahkan dapat melakukan *camouflage* terhadap arsip yang sebelumnya juga telah ter-*camouflage*.

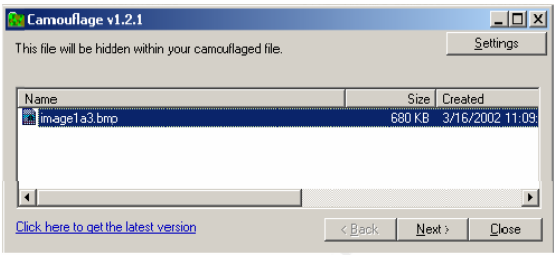
Sebagai contoh, pengguna dapat membentuk suatu arsip gambar yang tampak dan berlaku seperti arsip gambar lainnya kecuali bahwa arsip tersebut menyimpan arsip tersembunyi yang telah dienkripsi, atau pengguna dapat menyembunyikan arsip di dalam dokumen Word yang tidak akan mengundang perhatian pihak lain. Arsip-arsip tersebut tentunya kemudian dapat diekstraksi dengan aman. Dan untuk keamanan tambahan, *Camouflage* juga memungkinkan pengguna untuk memberi *password* atas arsip yang di-*camouflage* tersebut. *Password* nantinya akan dibutuhkan kembali saat melakukan ekstraksi arsip tersebut.

Camouflage adalah suatu produk *freeware* dengan *copyright* oleh Twisted Pear Productions yang hingga sekarang ini telah mengeluarkan 4 buah versi. Versi pertama yaitu v1.0.4, diikuti dengan v1.1.1, v1.1.2, dan terakhir yaitu v1.2.1. [1]

3.2. Cara Menggunakan *Camouflage*

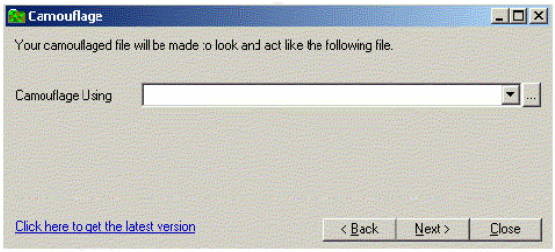
3.2.1. *Camouflaging* terhadap Arsip

Camouflage sangat mudah digunakan, hanya dengan meng-klik kanan arsip dan memilih opsi '*Camouflage*'. Berikut adalah layar *Camouflage* yang muncul sesaat setelah pemilihan arsip untuk di-*camouflage*.



Gambar 7. Layar deskripsi mengenai arsip yang akan di-*camouflage*

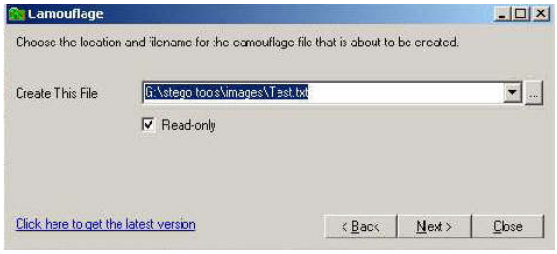
Citra yang dipilih "image1a3.bmp" yaitu citra yang akan di-*camouflage* ke dalam arsip lain yang akan ditentukan pada langkah selanjutnya. Ingat bahwa ukuran arsip citra yang asli yaitu 680 KB. Dengan menekan tombol 'Next>' akan memunculkan layar di bawah ini, yaitu layar untuk memilih arsip yang akan dilekatkan dengan arsip citra.



Gambar 8. Layar untuk memilih arsip yang akan dilekatkan pada arsip yang akan di-*camouflage*

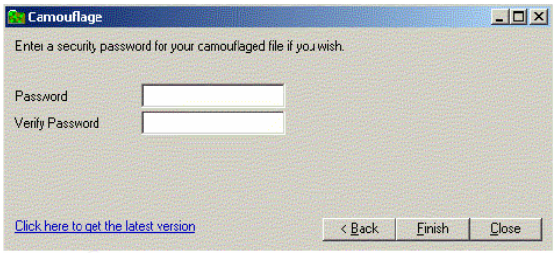
Pada langkah ini, proses *camouflage* akan menggunakan tombol jelajah yang terdapat pada sebelah kanan kotak masukan. Arsip yang dipilih untuk dilekatkan dapat dalam format arsip apapun. Tidak terdapat ketentuan bahwa arsip tersebut harus sama dengan arsip yang akan di-*camouflage*. Setelah memilih arsip, layer opsi untuk mengganti nama arsip baru yang telah di-*camouflage* akan muncul seperti pada Gambar 3. Dapat dilihat pada gambar tersebut bahwa terdapat 2 poin yang penting.

Pertama, nama dari arsip baru sama sekali tidak berhubungan dengan arsip asli. Sebuah *bitmap* telah dipilih untuk di-*camouflage*, dan sebuah file teks kemudian dipilih untuk menyimpan arsip *bitmap* (hal tersebut menjadi penting pada bab penjelasan mengenai kekurangan dari *Camouflage*). Kedua, kemampuan untuk melakukan verifikasi bahwa arsip dapat dibaca, sekali lagi meyakinkan bahwa kerusakan terhadap arsip tidak akan terjadi.



Gambar 9. Layar untuk mengganti nama arsip baru yang telah di-*camouflage*

Layar akhir yang akan muncul dalam proses *camouflage* ini adalah cara lain untuk meyakinkan bahwa arsip tersebut aman. *Password* yang disimpan dalam arsip adalah opsional namun dibutuhkan untuk membuka arsip jika sebelumnya *password* digunakan untuk membuat arsip yang telah di-*camouflage*. Kemampuan untuk menambahkan *password* pada arsip yang di-*camouflage* minimal akan menambah banyak pekerjaan bagi seseorang yang memiliki program *Camouflage* namun lupa atau tidak memiliki *password* tersebut. Gambar 4 adalah kotak dialog *password* yang merepresentasikan proses finalisasi pembuatan arsip *camouflage*. Opsi ini akan selalu muncul namun *password* tidak harus diterapkan pada arsip tersebut.

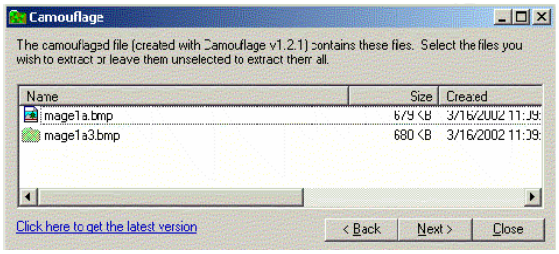


Gambar 10. Layar akhir proses *camouflage*

Untuk melengkapi proses *camouflage*, tekan tombol 'Finish' dan sebuah arsip baru telah diciptakan. Keseluruhan proses *camouflage* ini memakan tidak lebih dari 1 menit. Arsip citra atau teks tersebut tidak memiliki perbedaan dengan yang asli. Jika suatu citra di-*camouflage* menjadi suatu dokumen MS Word, maka saat arsip tersebut dibuka akan muncul dan berlaku seperti arsip Word biasa. Kemampuan ini berlaku untuk hamper keseluruhan tipe arsip. Namun tentunya hal ini sedikit menimbulkan kecurigaan apakah arsip asli telah berhasil di-*camouflage* ke arsip yang baru. Untuk mengatasi hal tersebut, sebaiknya terdapat aspek lain dari arsip yang bertujuan untuk mengalamatkan bahwa arsip tersebut telah melalui proses *camouflage*. [4]

3.2.2. *Uncamouflaging* terhadap Arsip

Semudah proses untuk melakukan *camouflage* terhadap suatu arsip, langkah untuk melakukan *uncamouflage* bahkan dilakukan hanya dalam 1 langkah. Untuk *uncamouflaging* arsip, cukup klik kanan pada arsip yang bersangkutan dan pilih opsi '*Uncamouflage*'. Gambar 5 menunjukkan tampilan arsip yang asli dan arsip apapun yang telah di-*camouflage*. Saat memilih opsi *uncamouflage*, jika arsip tersebut mengandung *password*, maka *password* perlu diketikkan untuk menyelesaikan proses. Pada arsip ini, sebuah arsip teks 1 KB di-*camouflage* menjadi arsip citra 678KB (image1a3.bmp) dan arsip citra kemudian di-*camouflage* kembali dalam arsip citra 679 KB (image1a.bmp). Hasil dari *camouflage* akan meningkatkan ukuran arsip, yang pada poin tertentu dapat menentukan apakah suatu arsip telah di-*camouflage*.



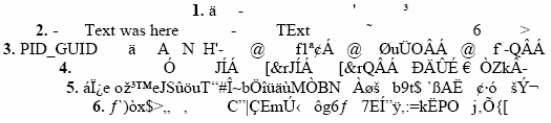
Gambar 11. Layar *uncamouflaging* arsip

3.3. Kekurangan pada *Camouflage*

Untuk menentukan apakah suatu arsip telah digunakan untuk steganografi adalah masalah utama yang akan terus berkembang seiring dengan perkembangan steganografi itu sendiri. *Camouflage* menggunakan sebuah teknik untuk menambah isi arsip yang akan mengakibatkan ukuran dari arsip meningkat dari arsip aslinya. Walaupun hal ini hanya dapat dibuktikan bila kedua arsip tersedia, namun kasus ini sangat jarang terjadi. Dua tipe dasar dari arsip dapat dialamatkan untuk kasus ini. Arsip citra menjadi salah satu yang menjadi tipe favorit untuk dialamatkan. Dasar dari *Camouflage* membuatnya menjadi sangat rentan untuk terdeteksi melalui atribut ukuran arsip tersebut.

Jika terdapat suatu dasar ukuran atas suatu arsip citra, maka akan sangat mudah untuk mendeteksi peningkatan ukuran arsip. Apabila arsip asli tidak tersedia, sebuah salinan dapat dibuat dari arsip yang dicurigai tersebut. Membuat ulang arsip dan memberikan manipulasi warna akan menunjukkan perbedaan ukuran arsip jika telah di-*camouflage*. Misal jika sebuah arsip *bitmap* 680 KB (arsip a) telah di-*camouflage* dengan suatu arsip *bitmap* lain yang juga berukurang 680 KB (arsip b), menghasilkan arsip baru berukuran 1360 KB (arsip c). Semua citra dalam *true color* dan terlihat sama persis jika hanya menggunakan mata manusia. Bila tidak terdapat arsip sumber sama sekali, sebuah salinan dapat dibuat dari arsip c dapat dibuat dan dimanipulasi. Dengan membuka arsip *bitmap* c, menggantinya menjadi 256 KB dan menggantinya kembali menjadi *true color* akan menghasilkan suatu *bitmap* dengan ukuran 680 KB (ukuran yang sama dengan arsip sumber). Dari hal ini, maka sangat jelas bahwa arsip tersebut telah dikenai steganografi.

Arsip teks juga disembunyikan dengan digabung dengan arsip sumber dan menghasilkan peningkatan ukuran, namun kemampuan untuk menyalin dan memanipulasi arsip teks tidak sejelas dengan arsip citra. Ada beberapa teknik yang dapat digunakan untuk menampilkan apakah suatu arsip teks telah di-*camouflage*. Membuka arsip teks dengan program yang tidak standar biasanya akan menghasilkan karakter tersembunyi diantaranya yang tidak dapat ditampilkan oleh program standar. Arsip Word yang telah di-*camouflage* ke dalam arsip Word lainnya dan dibuka menggunakan MS Word, tidak akan mengindikasikan adanya arsip yang telah di-*camouflage* dngan arsip lain. Membuka kedua arsip dengan Notepad dapat menunjukkan sebuah perbedaan yang mendasar antara kedua arsip. Yang paling mudah untuk disadari yaitu adanya data terenkripsi pada akhir arsip yang telah di-*camouflage*. Data pada akhir arsip *camouflage* tidak menunjukkan kemiripan dengan sisa arsip lainnya dan terlihat sangat dipadatkan (*compressed*). Teks di bawah ini adalah cuplikan dari arsip MS Word yang ditampilkan menggunakan Notepad. Baris pertama hingga keempat adalah keseluruhan bagian dari dokumen asli, dan baris kelima dan keenam adalah awal dari dokumen *camouflage*. Perhatikan bahwa baris lima dan enam tidak mengandung spasi diantara karakter-karakternya dan karakter spesial menjadi sangat banyak digunakan. Dokumen asli berakhir pada baris keempat dan tidak memiliki salah satupun atribut yang terdapat pada dokumen baris kelima dan keenam.



Gambar 12. Contoh tampilan arsip teks *camouflage*

Kemampuan untuk membuka arsip menggunakan format yang tidak standar adalah cara yang sangat jitu, dan dapat membantu dalam menentukan apakah suatu arsip telah di-*camouflage*. Kemampuan untuk menentukan apakah suatu arsip telah melalui proses steganografi akan terus dikembangkan seiring dengan semakin banyaknya metode steganografi yang dikembangkan.

Salah satu factor kunci dalam menggunakan *Camouflage* adalah mengenai pengaturan *registry*. Kemampuan untuk mencari *registry* dapat membawa banyak informasi mengenai arsip yang telah diproduksi menggunakan *Camouflage*. HKEY_CURRENT_USER\Software*Camouflage*\frmMain\ *Camouflage*FileList menyimpan suatu daftar atas keseluruhan arsip yang memiliki data *camouflage* dalam arsip lainnya. Daftar ini selalu di-*update* seiring dengan adanya arsip baru yang di-*camouflage*. Kemampuan untuk mengetahui arsip, termasuk nama arsip yang telah di-*camouflage* akan menjadikannya sangat mudah untuk mendeteksi adanya steganografi.

Camouflage memiliki proses yang sangat buruk saat dilakukannya proses *uninstall* dan meninggalkan suatu *fingerprint* yang sangat jelas. Setelah proses *uninstall* selesai, suatu pencarian pada *registry* akan membuka sejumlah kehadiran informasi yang tidak dibuang. Yang lebih penting, catatan yang tertinggal tersebut masih mengandung data yang mengindikasi bahwa arsip telah digunakan dengan program *Camouflage*. Walaupun arsip tidak dapat di-*camouflage* menggunakan program, namun proses *install* ulang yang sederhana akan memecahkan masalah tersebut.

Apa yang *Camouflage* kurang miliki dalam keamanan dan kemampuan untuk menutupi fakta dari suatu arsip tersembunyi, diimbangi dengan adanya fleksibilitas. Pengaturan *registry*, data gabungan yang menghasilkan arsip dengan ukuran yang lebih besar, dan data terenkripsi yang berada pada arsip normal membuat *Camouflage* sangat mudah untuk dideteksi. Hal ini diimbangi dengan banyaknya jenis format arsip yang dapat digunakan dengan *Camouflage*. Kesemua ekstensi arsip tersebut dapat dikenai steganografi oleh *Camouflage*. Tentunya kemampuan untuk menambahkan data ke dalam arsip dengan tipe apapun membuat program ini menjadi sangat cakap, namun sayangnya sangat mudah dideteksi. [4]

3.4. Pemecahan Algoritma *Camouflage*

Camouflage sendiri saat ini sudah sangat jarang digunakan karena telah banyak orang yang mampu untuk memecahkan algoritma yang digunakan oleh *Camouflage* dalam menyembunyikan informasi. Salah satunya adalah seorang pemerhati steganografi bernama Guillermito. Ia berusaha memecahkan algoritma dari *Camouflage* ini setelah mendapat tantangan dari seorang ahli komputer yang mengatakan dalam suatu wawancara di televisi Prancis bahwa penggunaan aplikasi steganografi sangat sulit untuk dideteksi bahkan dengan aplikasi secanggih dan secerdas apapun.

Guillermito yang menyangkal pendapat tersebut lalu ditantang oleh si ahli komputer untuk mendeteksi adanya suatu dokumen Word tersembunyi pada suatu arsip dengan dikirimkannya 2 buah arsip gambar JPG. Guillermito menerima tantangan tersebut dan hanya dalam beberapa menit dengan menggunakan *editor* heksadesimal, ia mampu mendeteksi citra mana yang telah dimodifikasi, dan kurang dari 1 jam kemudian ia telah berhasil memecahkan isi dari dokumen tersembunyi tersebut.

Guillermito mulai melakukan pemecahan algoritma tersebut dengan pertama-tama mendeteksi bahwa salah satu dari gambar tersebut telah ditambahkan pada bagian akhirnya (dengan menggunakan *editor* heksadesimal). Karena hampir seluruh format arsip memiliki struktur yang tetap, dan begitu juga dengan JPG, maka akan sangat mudah untuk mengetahui letak citra tersebut berakhir, dan dimana data yang tersembunyi tersebut mulai diletakkan.

Guillermito dapat mengetahui bahwa aplikasi yang digunakan oleh si ahli komputer adalah *Camouflage* dengan pertimbangan bahwa hanya sedikit aplikasi steganografi yang menyembunyikan data pada akhir dari suatu arsip karena itu sangatlah lemah dan mudah untuk dideteksi. Dari beberapa aplikasi yang melakukan metode tersebut, Guillermito dapat mengenalinya sebagai *Camouflage* dengan membandingkan antarmuka-nya saat si ahli komputer mendemokan program tersebut dalam wawancaranya di televisi.

Walaupun data pada arsip tersebut disembunyikan pada akhir dari arsip, dan mudah untuk dideteksi, Guillermito menemukan bahwa data tersembunyi tersebut telah dienkripsi atau diacak dengan suatu cara. Ia lalu melakukan eksplorasi terhadap *Camouflage* dan menemukan bahwa datanya

dienkripsi dengan sangat lemah dan hanya dengan beberapa tes dengan *password* yang telah dipilih akan cukup untuk memecahkan aplikasi tersebut.

Berikut adalah contoh pemecahan algoritma *Camouflage* yang diberikan oleh Guillermito dengan menggunakan contoh sebuah foto koleksi seni hologram “Lucy in a Tin Hat” oleh seniman Inggris Patrick Boyd. Untuk contoh ini, arsip yang tersembunyi yaitu dalam format txt dengan isi arsip “This is the secret message.”.

Gambar JPG asli, tanpa informasi tersembunyi di dalamnya, dengan ukuran 5139 bytes.

Gambar 13. Citra 1: JPG asli

Gambar asli dengan pesan rahasia yang ditambahkan, tanpa password, ukuran 6021 bytes.

Gambar 14. Citra 2: JPG dengan pesan rahasia tanpa password

Gambar asli, dengan pesan rahasia di dalamnya, dengan password "aaaa", ukuran 6021 bytes

Gambar 15. Citra 3: JPG dengan pesan rahasia dan password

Gambar asli, dengan pesan rahasia di dalamnya, password "a" diulang 255 kali, ukuran 6021 bytes

Gambar 16. Citra 4: JPG dengan pesan rahasia dan password diulang 255 kal

Sebagai catatan, tampilan heksadesimal yang akan ditampilkan berikut ini tidak sama persis dengan konversi dari arsip citra yang di atas, namun struktur datanya adalah sama persis.

Hal yang harus diperhatikan saat membandingkan citra asli dengan citra lainnya yaitu adanya blok besar dari data yang sangat mudah dikenali pada akhir arsip, tepat setelah FF D9 – tanda akhir dari suatu arsip JPG.

Dimulai dengan “20 00”, lalu sejumlah variabel *header*, mungkin dengan beberapa data tersembu-

nyi seperti ukuran dari arsip rahasia dan arsip asli, lalu akan ditemukan data arsip tersembunyi yang terenkripsi, dan terdapat pula sejumlah angka “20” (32 dalam decimal, dalam kode ASCII berarti spasi, *buffer* ini kemungkinan untuk menyimpan *string* ASCII) dengan 2 daerah data yang terenkripsi, dan terakhir terdapat *signature* final yang tetap, sebagai tanda berakhirnya data.

Berikut adalah contoh data yang terdapat pada akhir citra 1 dengan pesan rahasia namun tanpa *password:*

```
1190    D2 46 97 A7 36 4B 11 FE E5 88 F5 5C DF F2 5F FF   End of the JPG file
11A0    D9 20 00 10 5E C2 01 B0 6C B1 38 10 5E C2 01 50   Start of Camouflage data
11B0    B6 E7 88 10 5E C2 01 00 34 6A 25 1B 00 00 00 56
11C0    FD 13 51 2C CF 67 C1 95 A7 DA 45 53 0A FD C1 FC
11D0    11 6A 3E 9E 85 06 35 CA 46 E3 FF FF FF FF 20 00   Some data
11E0    10 5E C2 01 20 12 83 53 10 5E C2 01 80 3D E9 88   + encrypted hidden file
11F0    10 5E C2 01 90 22 3F 72 71 F0 19 50 69 D2 4B 8C
1200    84 BC CC 04 47 0A B0 C7 E1 11 20 20 20 20 20 20
1210    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1220    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1230    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1240    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1250    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20   Empty buffer
1260    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1270    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1280    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1290    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
12A0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
12B0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
12C0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
12D0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
12E0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
12F0    20 20 20 20 20 20 20 67 F8 0A 56 75 88 7E 91 86   Some data
1300    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1310    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1320    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1330    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1340    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1350    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1360    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20   Empty buffer
1370    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1380    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1390    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
13A0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
13B0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
13C0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
13D0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
13E0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
13F0    20 20 20 20 20 20 1B 00 00 00 A1 11 00 00 02 00   Some data
1400    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1410    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1420    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1430    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1440    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1450    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1460    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20   Empty buffer
1470    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1480    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1490    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14A0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14B0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14C0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14D0    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

```
14E0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14F0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 74
1500     A4 54 10 22 97 20 20 20 20 20 20 20 20 20 20 20   Some data
1510     20 20 20
```

Akan menjadi mudah untuk mengetahui secara tepat arti dari keseluruhan *field* tersebut (sebagai contoh, ukuran dari teks tersembunyi, yaitu 27 dalam desimal atau 1B dalam heksa, yang muncul 2 kali, pada bagian yang digarisbawahi), namun sebetulnya hal tersebut tidak diperlukan. Akan lebih menarik bila pencarian difokuskan pada *password*.

Hal menarik lain yang dapat dilihat yaitu bahwa jika *password* berubah, blok data pertama, yang mengandung pesan terenkripsi dan tersembunyi, tidak berubah sama sekali! Hal ini sangat aneh karena enkripsi datanya ternyata tidak tergantung oleh *password*. Hanya beberapa bytes yang dimodifikasi pada bagian angka-angka "20".

Berikut adalah perbandingan bagian angka-angka "20" (dimulai dari *offset* 1400h) dari arsip berbeda yang telah di-*camouflage* dengan *password* yang berbeda. Pertama-tama, citra dengan pesan tersembunyi tanpa *password* (sama seperti contoh di atas). Semuanya kosong:

```
1400     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1410     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1420     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1430     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1440     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1450     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1460     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1470     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1480     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1490     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14A0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14B0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14C0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14D0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14E0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14F0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

Kedua, citra dengan pesan tersembunyi dan dengan *password* bernilai "aaaa". Bagian diberi warna biru adalah bytes yang termodifikasi. Dapat dilihat bahwa *password* dengan panjang 4 bytes akan menghasilkan modifikasi sepanjang 4 bytes pula. Hal ini menunjukkan bahwa enkripsi yang digunakan sangatlah lemah:

```
1400     63 F4 1B 43 20 20 20 20 20 20 20 20 20 20 20 20
1410     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1420     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1430     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1440     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1450     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1460     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1470     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1480     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
1490     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14A0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14B0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14C0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14D0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14E0     20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
14F0     20 20 20 20 20 20 20 20 20 20
```

Dan kemudian untuk citra dengan pesan tersembunyi yang dilakukan dengan perulangan sebanyak 255 kali atas karakter “a”. Bagian yang

diberi warna hijau adalah bytes yang termodifikasi. Kesemua bytes pada *buffer* kali ini termodifikasi. Dapat dilihat bahwa 4 nilai pertama sama dengan nilai untuk citra sebelumnya, yaitu dengan pesan tersembunyi dengan password bernilai “aaaa”.

```
1400    63 F4 1B 43 6D C7 75 80 80 AE DE 04 41 0E FF D2
1410    F8 04 2B 32 9A 97 14 35 CC 42 AC 1F FD 48 86 9D
1420    83 98 2C B3 23 2F 67 A1 99 FB 7D 03 59 15 45 61
1430    34 BE 20 AA 60 C3 D6 92 EE EB BC CD 52 E2 01 48
1440    92 19 45 5F 1B 8A B2 85 FC FC 22 F5 2B A6 24 0C
1450    44 15 8A 6A F9 A8 1D 9D A9 DB 53 0A 61 B2 A4 A3
1460    F5 55 CE D1 84 F4 1C 4B E5 C5 3E 84 0F 46 4B BA
1470    F7 1F 5F 29 58 27 AE 0E 10 CB 5D 50 FB C8 FF EE
1480    E8 12 D2 58 AB 53 B4 91 50 38 1D 63 4F E7 56 98
1490    4A 1F 30 93 20 E0 6D B5 04 74 96 11 B5 78 F9 41
14A0    DE 41 D9 34 06 AD E0 79 ED 72 5D 02 5D F3 70 85
14B0    3A 7A 69 43 01 2D 2B A4 EB D2 A4 14 A2 F1 1B 93
14C0    D3 D7 A9 B1 59 EB A3 E7 91 CD 88 AB 3D 2F 5F 68
14D0    48 19 48 F8 3B E5 B4 DB 3F B4 F3 1B 59 9B B1 01
14E0    8D 94 46 DB 8F D6 BF FE FA BF 04 B5 17 58 17 FD
14F0    BB 09 EC C9 C1 C7 7F B8 BA 6E 2C CA F3 AC 10
```

Kesimpulan yang dapat diambil yaitu bahwa *password* disimpan pada posisi tersebut (awal mulai terdapat data terenkripsi), kemungkinan dilakukan penyamaran menggunakan operasi XOR dengan kunci yang dibentuk oleh suatu *string* statik dari bytes. Kali ini *string* menjadi mudah untuk didapatkan. Karena XOR sifatnya dapat dibalik, maka operasi yang dilakukan adalah cukup dengan melakukan operasi XOR pada data di atas dengan *password*-nya, yaitu “aaaa…”, yang dalam desimal adalah “61616161…”. Hasil dari operasi ini adalah *string* kunci yang digunakan oleh *Camouflage* dalam mengenkripsi *password* masukan dari pengguna:

```
63F41B436DC7758080AEDE04410EFFD2  F8042B329A971435CC42AC1FFD48869D
83982CB3232F67A199FB7D0359154561  34BE20AA60C3D692EEEBBCCD52E20148
9219455F1B8AB285FCFC22F52BA6240C  44158A6AF9A81D9DA9DB530A61B2A4A3
F555CED184F41C4BE5C53E840F464BBA  F71F5F295827AE0E10CB5D50FBC8FFEE
E812D258AB53B49150381D634FE75698  4A1F309320E06DB504749611B578F941
DE41D93406ADE079ED725D025DF37085  3A7A6943012D2BA4EBD2A414A2F11B93
D3D7A9B159EBA3E791CD88AB3D2F5F68  481948F83BE5B4DB3FB4F31B599BB101
8D9446DB8FD6BFFEFABF04B5175817FD  BB09ECC9C1C77FB8BA6E2CCAF3AC10

                                XOR

61616161616161616161616161616161  61616161616161616161616161616161
61616161616161616161616161616161  61616161616161616161616161616161
61616161616161616161616161616161  61616161616161616161616161616161
61616161616161616161616161616161  61616161616161616161616161616161
61616161616161616161616161616161  61616161616161616161616161616161
61616161616161616161616161616161  61616161616161616161616161616161
61616161616161616161616161616161  61616161616161616161616161616161
61616161616161616161616161616161  61616161616161616161616161616161

                                =

02957A220CA614E1E1CFBF65206F9EB3  99654A53FBF67554AD23CD7E9C29E7FC
E2F94DD2424E06C0F89A1C6238742400  55DF41CB01A2B7F38F8ADDAC33836029
F378243E7AEBD3E49D9D43944AC7456D  2574EB0B98C97CFCC8BA326B00D3C5C2
9434AFB0E5957D2A84A45FE56E272ADB  967E3E483946CF6F71AA3C319AA99E8F
8973B339CA32D5F031597C022E8637F9  2B7E51F241810CD46515F770D4199820
BF20B85567CC81188C133C633C9211E4  5B1B0822604C4AC58AB3C575C3907AF2
B2B6C8D0388AC286F0ACE9CA5C4E3E09  297829995A84D5BA5ED5927A38FAD060
ECF527BAEEB7DE9F9BDE65D47639769C  DA688DA8A0A61ED9DB0F4DAB92CD71
```

Dari sini dapat diketahui lokasi dari *password* yang digunakan (lokasi tetap yang relative terhadap akhir dari arsip, *offset* -275 dalam desimal), dan bagaimana cara untuk melakukan decipher.

Dari citra ketiga (citra yang mengandung pesan rahasia dengan *passwor*d "aaaa") dapat ditemukan bahwa *buffer password* mengandung nilai berikut:

```
63 F4 1B 43
```

Yang harus dilakukan untuk menemukan nilai *password* yaitu "aaaa" adalah dengan mengenakan operasi XOR antara *buffer password* tersebut dengan kunci enkripsi *Camouflage* yang telah ditemukan sebelumnya. Maka:

```
63 F4 1B 43

    XOR

02 95 7A 22

     =

61 61 61 61
```

Karakter ASCII dari 61 adalah "a", karena itu dapat dipecahkan bahwa pada citra ketiga *password* yang dikenakan terhadap citra *camouflage* tersebut adalah "aaaa". [8]

Dengan dapat dideteksinya nilai dari *password* suatu arsip *camouflage* maka akan sangat mudah untuk mendekripsi arsip tersebut dan mencari pesan rahasia di dalamnya. Hal tersebut dapat dilakukan dengan menggunakan bantuan dari aplikasi *Camouflage* itu sendiri. Pilih arsip yang dicurigai adalah arsip *camouflage*, lalu klik kanan pada arsip tersebut dan pilih opsi "*Uncamouflage*". Lalu masukkan *password* yang telah berhasil didapatkan untuk *uncamouflaging* arsip tersebut.

Pada contoh kasus arsip citra 3 ini, maka setelah dilakukan *uncamouflaging* dengan masukan *password* "aaaa", maka akan diketahui adanya arsip tersembunyi di dalamnya bernama "secret_message.txt" dengan isi teks yaitu "This is the secret message."

Guillermito sendiri bahkan membuat aplikasi sederhana untuk memecahkan *password* dari arsip yang telah di-*camouflage*. Aplikasi tersebut dinamakan "*Camouflage*_Password_Finder" yang dikembangkan menggunakan bahasa Assembly. Berikut adalah penjelasan umum mengenai langkah kerja aplikasi *cracking password* tersebut.

Digunakan aturan seperti yang telah dijelaskan sebelumnya untuk menemukan akhir dari suatu *password* yaitu dengan menghentikan pencarian bila telah mencapai karakter `20` dalam heksadesimal (nilai *password* dan data lain dipisahkan oleh sejumlah besar nilai `20` yang berulang). Namun cara ini juga dapat membawa ketidakberhasilan dengan kemungkinan 1/256. Seperti untuk contoh jika karakter ketiga dari *password* adalah "Z", karena Z yang memiliki nilai heksadesimal `5a`, jika di XOR dengan kunci dari *Camouflage* untuk urutan ketiga juga, yaitu `7a` akan menghasilkan nilai `20`. Sehingga bila aplikasi ini dijalankan untuk arsip *camouflage* dengan *password* bernilai "aaZaaa", maka akan dihasilkan bahwa *password* yang digunakan adalah "aa".

Lalu terdapat pula kasus jika ditemukannya nilai `20` pada *password* yang dienkripsi. Maka aplikasi akan melakukan pengecekan apakah semua dari bytes lainnya adalah juga `20` sebelum memutuskan telah mencapai akhir dari *password*. Selain itu program milik Guillermito tersebut juga tidak melakukan pengecekan jika suatu arsip mengandung data yang telah di-*camouflage*. Maka, jika hasil dari program tersebut menampilkan banyak karakter biner, maka kemungkinan besar arsip yang dicek tersebut tidak di-*camouflage*. [2]

Dari penjelasan ini, maka dapat diketahui bahwa *Camouflage* bukannya suatu aplikasi steganografi yang dapat diandalkan karena pola penyembunyian datanya sangat mudah dideteksi, terutama dengan melihat ukuran arsip yang meningkat akibat disisipkannya arsip rahasia ke dalamnya. Dan langkah yang dilakukan oleh *Camouflage* dalam mengenkripsi data rahasia tersebut adalah dengan cara menambahkan bytes dari data rahasia tersebut ke bagian akhir dari arsip lain. Kemudian nilai *password* yang menjadi kunci dari proses *uncamouflaging* menggunakan aplikasi *Camouflage* itu sendiri ternyata sangat tidak terjaga kerahasiaannya karena mudah untuk dilacak. Dengan mencari tahu *password* dari suatu arsip *camouflage* seperti pada langkah yang telah dijelaskan sebelumnya, maka sangat mudah untuk mencari tahu arsip rahasia yang tersembunyi, dengan bantuan proses *uncamouflage* dari aplikasi *Camouflage* itu sendiri.

4. Kesimpulan

Dari seluruh bahasan di atas maka dapat ditarik beberapa kesimpulan. Antara lain yaitu bahwa steganografi adalah suatu teknik untuk

menyembunyikan informasi yang bersifat pribadi dengan sesuatu yang hasilnya akan tampak seperti informasi normal lainnya. Terdapat banyak metode yang dapat digunakan dalam mengaplikasikan steganografi, antara lain yaitu metode LSB (*least significant byte)*, *spread spectrum*, kunci publik steganografi, domain transformasi, dan *embedding/injection*.

Saat ini, aplikasi steganografi telah banyak bermunculan untuk yang bertujuan untuk menjaga keamanan informasi pribadi seseorang dari pihak yang tidak berwenang. Salah satu dari aplikasi steganografi tersebut adalah *Camouflage*. *Camouflage* memungkinkan pengguna komputer untuk menyembunyikan arsip dengan mengacaknya dan melampirkannya ke dalam arsip lain. Arsip yang telah di-*camouflage* akan tampak dan berlaku seperti arsip normal lainnya, dan dapat disimpan ataupun dikirimkan sebagai *email* tanpa menarik perhatian. Selain itu *Camouflage* juga memungkinkan pengguna untuk memberi *password* atas arsip yang di-*camouflage* tersebut. *Password* nantinya akan dibutuhkan kembali saat melakukan ekstraksi arsip tersebut.

Namun sayangnya, algoritma yang digunakan oleh *Camouflage* dalam menyembunyikan arsip dalam arsip lainnya sangat lemah dan mudah dipecahkan. Hanya dengan membandingkan arsip asli dengan arsip yang telah di-*camouflage* maka seseorang dapat dengan mudah mencari tahu *password* yang digunakan untuk *uncamouflaging* arsip tersebut. Dari kenyataan tersebut, tentunya *Camouflage* tidak dapat lagi dinyatakan sebagai suatu aplikasi steganografi yang aman dan tidak dapat dideteksi.

Dan dari langkah pemecahan algoritma yang dilakukan terhadap aplikasi *Camouflage*, dapat diketahui bahwa *Camouflage* menyisipkan informasi rahasia pada akhir dari arsip lainnya. Maka dapat disimpulkan bahwa metode yang digunakan oleh *Camouflage* dalam menyembunyikan informasi adalah dengan metode *embedding/injection*.

Daftar Pustaka

[1] Arsip *Camouflage* v1.2.1. *readme*

[2] Arsip *Camouflage_Password_Finder readme*

[3] Anderson, R. J. – Petitcolas, F. A. P., *On The Limits of Steganography, IEEE Journal of*

[4] Bartlett, John. The *Easy of Steganography and Camouflage GSEC V1.3*. Sans Institute, March 17th 2002.

[5] Bender, W. – Gruhl, D. – Morimoto, N., *Techniques for Data Hiding*, Massachusetts Institute of Technology, Media Laboratory Cambridge, Massachusetts 02139 USA, From the *Proceedings of the SPIE*, 2420:40, San Jose CA, February, 1995.

[6] Currie, D. L., *Surmounting the Effects of Lossy Compression on Steganography*, III Fleet Information Warfare Center 2555 Amphibious Drive NAB Little Creek Norfolk, VA 23521 3225 currie@msn.comi Cynthia E. Irvine Computer Science Department Code CS/Ic Naval Postgraduate School Monterey, CA 93943 5118 irvine@cs.nps.navy.mil *Proceedings of the 19th National Information System Security Conference*, Baltimore, Md, October 1996, pp. 194–201.

[7] Craver, Scott. *On Public-key Steganography in the Presence of an ActiveWarden,* Intel Corporation Microcomputer Research Labs 2200 Mission College Blvd., Santa Clara, CA 95052–8119 Department of Mathematical Sciences Northern Illinois University DeKalb, IL 60115.
Selected Areas in Communications, 16 (4) pp. 474-481, May 1998. Special Issue on Copyright & Privacy Protection. ISSN 0733-8716.

[8] Guillermito. (easily) Breaking a (very weak) Steganography Software, September 16th 2002. *http://www.guillermito2.net/stegano/camouflage.html*

[9] Lenti, Jozsef. *Steganographic Methods*. Department of Control Engineering and Information Technology. Budapest University of Technology and Economics, June 5th 2000.

[10] Le, Tien. Destegging Tutorial, 2002. *http://www.unfiction.com/dev/tutorial*

[11] Mangarae, Aelhaeis. *Steganography FAQ*. Zone-H.Org, March 18th 2006.

[12] Munir, Rinaldi. *Diktat Kuliah IF5054 Kriptografi*. Program Studi Teknik Informatika, Institut Teknologi Bandung, 2006.

[13] Peikari, Cyrus and Seth Fogie. Steganography. *http://www.informit.com/guides/contents.asp*. Security Resource Center.

[14] Rago, Michael T. Steganography, Steganalysis, and Cryptanalysis. VeriSign.

[15] Smith, J. R. – Comiskey, B. O., fjrs, *Modulation and Information Hiding in Images*, elwoodg@media.mit.edu Physics and Media GroupMIT Media Lab 20 Ames Street Cambridge, MA 02139 USA *Proceedings of the First Information Hiding Workshop*, Isaac Newton Institute, Cambridge, U.K., May

1996. Springer-Verlag *Lecture Notes in Computer Science* Volume 1174.

[16] Zhao, J. – Koch, E., *Embedding Robust Labels Into Images For Copyright Protection*, Fraunhofer Institute for Computer Graphics Wilhelminenstr. 7, 64283 Darmstadt, Germany Email: {zhao, ekoch}@igd.fhg.de *Proc. of the Int. Congress on Intellectual Property Rights for Specialized Information, Knowledge and New Technologies*, Vienna, August 1995.