Analisis Implementasi Teknik *Universal Message Authentication Code* pada Aplikasi *Instant Messenger* Pidgin 2.2.0

Heryanto [1)]

1) Program Studi Teknik Informatika ITB, Bandung 40132, email: if14081@students.if.itb.ac.id

Abstract – *Makalah ini membahas bagaimana analisis implementasi salah satu varian algoritma Message Authentication Code (MAC) pada sebuah aplikasi messaging client. Varian MAC yang dipilih adalah Universal Message Authentication Code, yaitu MAC yang berdasar pada universal hashing. UMAC dihitung dengan memilih fungsi hash dari beberapa kelas fungsi hash, masing-masing kelas memiliki proses tersendiri yang dirahasiakan. Hasil dari fungsi hash tersebut kemudian dienkripsi untuk menyembunyikan fungsi hash yang dipakai. Seperti MAC lainnya, UMAC dapat digunakan untuk menguji integritas data dan otentikasi pesan. Teknik UMAC memiliki kekuatan kriptografi yang teruji dan relatif memiliki komputasi yang sederhana dibandingkan MAC lainnya*

Aplikasi messaging client yang dipilih untuk implementasi UMAC ini adalah Pidgin. Pidgin adalah salah satu messaging client open-source yang berlisensi GNU dan dikembangkan dengan Gtk+. Oleh karena itu, pengembangan aplikasi ini cenderung lebih mudah, baik secara teknis, maupun non-teknis. Ada beberapa versi Pidgin yang terdapat di internet, tetapi versi engine terbaru yang dipakai masih versi 2.2.0. Oleh karena itu, pengembangan akan dipilih menggunakan API untuk versi 2.2.0. Untuk membangun plugin Pidgin, telah tersedia libpurple, yaitu library yang digunakan aplikasi pengembang untuk berhubungan dengan core Pidgin. Jadi aplikasi plugin baru bisa memanfaatkan libpurple untuk mengakses koneksi, conversation, account list, dsb.

Kata Kunci*: Message Authentication Code (MAC), Universal Message Authentication Code (UMAC), messaging client, Pidgin, Gtk+, libpurple.*

1. PENDAHULUAN

Dewasa ini, penggunaan *messaging client* berkembang cukup pesat dan bervariasi. Awalnya *messaging client* hanya digunakan beberapa orang untuk sekedar *chatting* atau ngobrol biasa. Namun seiring perkembangan jaman, penggunaan *messaging client* tidak terbatas hanya sebagai komunikasi ngobrol antara dua orang saja. *Messaging client* bisa juga dapat digunakan untuk melakukan percakapan serius, seperti ekonomi, politik, bahkan sosial. Akibatnya dibutuhkan juga suatu sarana yang dapat menjamin keamanan suatu percakapan, baik dari segi integritas data, maupun ketersediaan otentikasi pesan. Hal ini juga diikuti pesatnya perkembangan teknik untuk mencuri pesan dalam Internet.

Salah satu cara yang dapat digunakan untuk menjaga integritas data sekaligus menyediakan fitur otentikasi pesan adalah teknik *Message Authentication Code* (MAC). Teknik ini adalah salah satu teknik kriptografi yang masih dipakai. MAC menghitung sebuah nilai (bisa berupa string, data biner, atau data heksadesimal) yang diperoleh dari sebuah pesan. Prinsip MAC mirip dengan dengan fungsi hash pada umumnya, yaitu menghasilkan suatu komputasi dengan ukuran yang tetap untuk berbagai ukuran pesan. Bedanya dengan fungsi hash, MAC mempunyai input tambahan yaitu sebuah kunci yang digunakan sebagai masukan dan mempengaruhi hasil MAC-nya.

Ada banyak varian MAC yang berhasil dikembangkan oleh berbagai ahli. Salah satunya adalah UMAC. UMAC atau *Universal Message Authentication Code* adalah MAC yang dihitung menggunakan fungsi hash universal. Kemudian hasil hash tersebut dienkripsi untuk menyembunyikan fungsi hash yang dipakai. Teknik UMAC dipercaya memiliki kekuatan kriptografi yang baik dan komputasi yang relatif lebih rendah dibanding teknik MAC lainnya.

2. PEMBAHASAN

2.1 Fungsi Hash Universal

Suatu fungsi hash dikatakan universal jika misal ada suatu kelas fungsi hash H dan ada himpunan pesan ringkas D. Suatu kelas fungsi H dikatakan universal jika ada anggota dalam kelas fungsi H yang mampu menghasilkan pesan ringkas anggota D yang sama dan maksimal hanya ada sejumlah |H|/|D| fungsi dalam H yang mampu menghasilkan anggota D yang sama. Teknik tersebut menyatakan bahwa jika ada seorang penyerang yang ingin mengganti pesan yang dikirim dan fungsi hash yang dipilih adalah acak, maka probabilitas perubahan tersebut diketahui sebanding dengan 1/|D|. Namun, definisi ini masih kurang kuat untuk menangani serangan-serangan kriptografi. Misalkan ada dua kemungkinan hasil pesan ringkas 0 dan 1 (D ={0,1}) dan H terdiri dari fungsi identitas dan fungsi not, H tetap memenuhi definisi universal, tetapi jika hasil hash dienkripsi dengan menggunakan operasi penambahan modular. Penyerang dapat mengubah pesan dan hasil ringkasnya secara bersamaan tanpa diketahui oleh penerima.

Fungsi hash universal yang kuat bisa dibentuk dengan

aturan sebagai berikut:
Jika ada sebuah kelas fungsi hash *H*, agar memiliki kekuatan kriptografi yang baik, kemungkinan penyerang menebak hasil pesan ringkas *d* dari pesan palsu *f* setelah menghalau sebuah pesan *a* dengan hasil pesan ringkas *c* harus sangat kecil. Berikut persamaannya:

$$Pr_{h \in H}[h(f) = d | h(a) = c] \quad (1)$$

Nilai peluangnya dari persamaan tersebut harus kecil, umumnya nilai yang dipakai adalah 1/|D|.

2.2 UMAC

Kurang lebih langkah UMAC adalah sebagai berikut:
Pihak pengirim pesan:

1. Membuat pesan ringkas dengan menggunakan fungsi hash yang tersedia tergantung dari masukan yang telah disepakati. Masukan yang disepakati bisa berupa kunci untuk enkripsi, pemetaan langsung, dsb.
2. Mengenkripsi dengan fungsi enkripsi yang telah ditentukan pada algoritma UMAC dengan masukan kunci untuk otentikasi. Kunci masukan adalah kunci yang telah disepakati sebelumnya oleh pengirim dan penerima.
3. Mengirim pesan berikut nilai UMAC yang telah dikonkatenasi.

Pihak penerima pesan:

1. Memisahkan pesan dengan nilai UMAC yang diterima.
2. Membuat pesan ringkas dengan fungsi hash yang sama ketika saat mengirim pesan. Bisa berupa kesepakatan atau diturunkan dari nilai lainnya.
3. Mendekripsi nilai UMAC yang diterima dengan kunci yang telah disepakati pengirim dan penerima.
4. Membandingan nilai pesan ringkas dengan hasil dekripsi nilai UMAC, jika hasilnya sama, maka otentikasi pesan berhasil dan integritas data terjaga.

2.3 Contoh Algoritma UMAC

Algoritma UMAC sangat mungkin bervariasi, hal ini dikarenakan jenis dan kelas fungsi hash yang diimplementasikan bisa apapun, dan kombinasinya pun bisa bermacam-macam. Berikut ini salah satu contoh algoritma UMAC yang sederhana:

```
void UHash24 (uchar *msg, uchar *secret, int
len, uchar *result)
 {
   uchar r1 = 0, r2 = 0, r3 = 0, s1, s2, s3,
byteCnt = 0, bitCnt, byte;

   while (len-- > 0) {
     if (byteCnt-- == 0) {
       s1 = *secret++;
       s2 = *secret++;
       s3 = *secret++;
       byteCnt = 2;
     }
     byte = *msg++;
     for (bitCnt = 0; bitCnt < 8; bitCnt++) {
       if (byte & 1) { /* msg not divisible
by x */
         r1 ^= s1; /* so add s * 1 */
         r2 ^= s2;
         r3 ^= s3;
       }
       byte >>= 1; /* divide message by x */
       if (s3 & 0x80) { /* and multiply
secret with x, subtracting
           the polynomial when necessary to
keep it's order under 24*/
         s3 <<= 1;
         if (s2 & 0x80) s3 |= 1;
         s2 <<= 1;
         if (s1 & 0x80) s2 |= 1;
         s1 <<= 1;

         s1 ^= 0x1B; /* x^24 + x^4 + x^3 + x
+ 1 */
       }
       else {
         s3 <<= 1;
         if (s2 & 0x80) s3 |= 1;
         s2 <<= 1;
         if (s1 & 0x80) s2 |= 1;
         s1 <<= 1;
       }
     } /* for each bit in the message */
   } /* for each byte in the message */
   *result++ ^= r1;
   *result++ ^= r2;
   *result++ ^= r3;
 }
```

3. IMPLEMENTASI PADA PIDGIN

Untuk implementasi pada Pidgin, Pidgin menyediakan sebuah library yang dapat dipakai untuk memudahkan pembuatan plugin, terutama masalah *user interface* dan pemanfaatan koneksi, percakapanan, penyimpanan akun, menu dan sub-menu, dsb. Library yang dimaksud adalah *libpurple*. Library ini didesain sebagai *front-end* dari Gtk+ yaitu, lingkungan pengembangan Pidgin.

3.1 Penanaman Kode UMAC pada Percakapan Pidgin

Pertama-tama untuk menginisialisasi plugin pada Pidgin, harus menuliskan kode sebagai berikut:

```
PURPLE_INIT_PLUGIN(pidgin_plugin,
init_plugin, info);
```

Secara singkat, prosedur tersebut adalah prosedur yang menjadi bagian dari *libpurple* yang digunakan untuk menginisialisasi plugin Pidgin, `pidgin_plugin` di atas adalah nama pluginnya, `init_plugin` adalah fungsi yang digunakan untuk inisialisasi plugin, `info` adalah struktur data yang digunakan untuk menyimpan data plugin, dari data biasa sampai fungsi-fungsi yang diterapkan ketika plugin di-load dan di-unload.

Untuk menanam kode UMAC pada percakapan Pidgin, harus dicantumkan sebuah kode yang menangkap *event* ketika percakapan dilakukan. Proses penangkapan *event* ini harus di-inisialisasi pada saat plugin di-load. Proses inisialisasi *event handler* dapata

dilakukan dengan menuliskan kode sebagai berikut:

```
purple_signal_connect(conv_handle, "receiving-im-msg", h, PURPLE_CALLBACK(PE_got_msg_cb), NULL);

purple_signal_connect(conv_handle, "sending-im-msg", h, PURPLE_CALLBACK(PE_send_msg_cb), NULL);
```

Potongan kode di atas digunakan untuk menginisialisasi *event handler* untuk tiap dilakukan percakapan, pada potongan kode pertama, *event handler* yang ditangani adalah ketika menerima pesan, sedangkan potongan kode berikutnya menangani pengiriman pesan. Argumen PURPLE_CALLBACK adalah fungsi yang akan dipanggil ketika *event* tersebut terjadi.

Di dalam fungsi yang digunakan sebagai *event handler* dapat ditangkap struktur data `message` yang berisi pesan yang diterima atau yang akan dikirim. Pada saat inilah pengembang dapat mengubah dan memodifikasi pesan yang dikirim.

Strategi implementasi UMAC adalah sebagai berikut:

1. Sebelum memulai percakapan yang aman (integritas data dan otentikasi terjamin) kedua belah pihak memasukkan kunci yang telah disepakati untuk digunakan dalam percakapan.
2. Setelah mode aman dijalankan, setiap pesan yang melewati Internet akan memiliki header dan nilai UMAC. Pesan yang masuk maupun keluar akan selalu melalui aplikasi UMAC.
3. Pihak penerima akan mengecek otentikasi dengan cara UMAC yang telah disebutkan di atas.
4. Pihak pengirim juga akan mengirimkan nilai UMAC setelah pesan mentah siap dikirim.

3.2 Eksperimen Plugin UMAC Pidgin

Berikut ini eksperimen-eksperimen yang telah di coba pada plugin UMAC Pidgin yang telah berhasil dikembangkan:

1. Memasukkan kunci pada awal percakapan untuk masukkan UMAC dengan cara menekan tombol secure pada window percakapan.

Gambar 1 Antarmuka Pemasukkan Kunci

2. Menunggu lawan bicara memasukkan kunci agar bisa memulai percakapan yang aman.

Gambar 2 Antarmuka Menunggu Konfirmasi

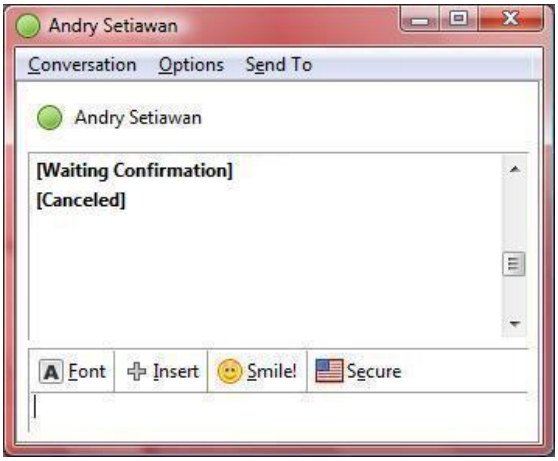
3. Setelah pihak lawan bicara memasukkan kunci, konfirmasi dianggap telah berhasil. Dan setiap lawan bicara menuliskan pesan, hasil uji otentikasi akan muncul sebelum pesan ditulis.

Gambar 3 Antarmuka Pesan Berhasil

4. Jika kunci yang dimasukkan tidak valid, atau pesan mengalami perubahan di saat pengiriman, maka akan muncul pesan gagal otentikasi setiap pesan diterima. Pada gambar di bawah, kegagalan otentikasi diakibatkan karena kesalahan kunci masukkan. Untuk saat ini, pengujian terhadap pesan yang berubah saat pengiriman belum bisa dilakukan.

Gambar 4 Antarmuka Otentikasi Gagal

5. Kasus lain adalah jika lawan bicara menolak pembicaraan secara aman. Akibatnya konfirmasi akan dihitung gagal. Berikut ini hasil tampilannya



Gambar 5 Antarmuka Konfirmasi Ditolak

3.3 Analisis Eksperimen

Setelah dilakukan eksperimen, ada beberapa hal yang perlu diperhitungkan, antaranya:

1. Integritas data dan otentikasi pesan dapat dilakukan walaupun pesan yang dikirim masih bisa diketahui orang, tetapi perubahan terhadap pesan dapat diketahui dengan kemampuan otentikasi yang dimiliki UMAC
2. Kecepetan percakapan seharusnya terganggu karena adanya proses yang lebih ketika percakapan terjadi. Namun, karena (mungkin) diuji pada spesifikasi komputer dengan *clock* 1.5 GHz, proses tidak terlalu terasa.
3. Jumlah data yang ditransfer bertambah, jadi untuk pesan yang sangat singkat (di bawah ukuran pesan ringkas), redundansi transmisi data cukup terasa. Namun karena ukuran pesan cenderung kecil, peningkatan ukuran pesan relatif bisa ditanggulangi.

4. KESIMPULAN

Penggunaan otentikasi pesan dan penjagaan integritas data bagus untuk diterapkan pada aplikasi *client messaging*. Walaupun ada sisi bermasalahnya yaitu, penurunan performa dan kecepatan. Namun, penurunan performa ini dapat ditangani dengan perkembangan teknologi saat ini. Di antaranya adalah perkembangan kecepatan *processing* dan jaringan.

DAFTAR REFERENSI

[1] Munir, Rinaldi, Diktat Kuliah IF5054 Kriptografi, Institut Teknologi Bandung, 2007.
[2] http://developer.pidgin.im/wiki/CHowTo.
[3] http://fastcrypto.org/umac/