“Studi mengenai penggunaan kriptografi nirsimetri pada aplikasi OpenSSH”

Aditya Nurcholis Hakim – NIM : 13503107

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13107@students.if.itb.ac.id

Abstrak

Sampai pada akhir tahun 1975, para kriptografer masih mengenkripsi pesan dengan kriptografi simetri. Setelah itu muncul Diffie dan Hellman yang mengusulkan kriptografi nirsimteri yang memungkinkan pengguna berkomunikasi secara aman tanpa perlu berbagi kunci rahasia. Dari sini muncullah berbagai macam algoritma yang berbasis kriptogragi nirsimetri seperti RSA, DSA, dan sebagainya. Banyak arsitektur program yang menggunakan algortima nirsimteri ini salah satu nya ialah SSH(Secure Shell). SSH merupakan satu set standard network protokol untuk membangun *secure channel* antara local computer dan remote computer. Dari berbagai program SSH yang ada, OpenSSH merupakan salah satu kakas implmentasi dari SSH yang menggunakan teknik RSA/DSA untuk mengautentikasi *user*-nya. Pada makalah ini penulis mencoba melakukan studi mengenai penggunaan RSA/DSA pada OpenSSH untuk pembangunan *secure channel* selain itu akan dibahas pula bagaimana cara untuk membangun passwordless connection.

Kata Kunci: Kriptografer, *RSA*, DSA, kunci publik, kunci privat, *OpenSSH*, kriptografi nirsimetri.

Pendahuluan

Pada komputer, Secure shell atau biasa disingkat dengan SSH adalah satu set standard dan terasosiasi network protokol yang memungkinkan untuk membangun sebuah ‘jaluir aman’ (secure channel) antara lokal dengan remote komputer. SSH ini menggunakan prinsip kriptografi kunci publi untuk mengautentifikasi remote komputer dan (juga) memungkinkan remote komputer untuk mengautentifikasi user. SSH menyediakan kerahasiaan dan integritas data yang dipertukarkan antar kedua komputer (lokal dan remote komputer) dengan menggunakan enkripsi dan Message Authentification Codes (MAC). SSH bukan hanya digunakan untuk log in pada remote machine dan mengeksekusi perintah – perintah, tetapi juga mensupport tunneling, memforward port – port arbitrary TCP dan koneksi X11; juga dapat mentransfer file – file dengan menggunakan protokol SFTP dan SCP. Sebuah server SSH, secara default menggunakan standard port TCP 22.

Bagaimana ssh menggunakan kriptografi kunci publik (dengan analogi)

Pertama – tama, sepasang kunci kriptografi dibangkitkan. Yang pertama ialah kunci privat sedangkan yang satu lainnya merupakan kunci publik. Sebagai analogi, mereka dapat dianggap sebagai sepasang kunci privat dan gembok publik. Gembok publik diinstal pada remote komputer dan digunakan ssh untuk mengautentifikasi user yang mempunyai padanan kunci privatnya. Sebagai user dari sistem tersebut, kita tidak peduli siapa yang dapat melihat atau meng-copy gembok tersebut, sebab hanya satu kunci privat yang cocok dengan gembok tersebut. Kunci privat merupakan bagian yang kita simpan pada ‘kotak aman’ (*secure box*) yang hanya dapat dibuka dengan ‘sandi lewat’ (*passphrase*) yang benar. Ketika user akan

mengakses remote sistem, ia membuka secure box tersebut dengan passphrasem dan menggunakan kunci privat tersebut untuk mengautentifikasi dirinya dengan gembok yang terdapat pada remote komputer. Baik passphrase maupun kunci privat tidak tertinggal pada mesin user. Namun, user juga harus mempercayai lokal mesin bahwa lokal mesin tidak mengambil passphrase ataaupun meng-copy kunci privat user tersebut ketika dibuka dari secure box.

Manajemen kunci pada Open SSH

Banyak dari kita menggunakan OpenSSH untuk mengganti telnet dan rsh command. Salah satu fitur ssh yang membanggakan ialah kemampuannya untuk mengautentifikasi user dengan menggunakan protokol autentifikasi RSA dan DSA, yang berdasar pada sepasang kunci numerikal komplementary. Salah satu daya tariknya ialah autentifikasi RSA dan DSA menjanjikan untuk pembangunan koneksi ke remote sistem tanpa harus memberikan sebuah password. Meskipun demikian, banyak pengguna baru OpenSSH yang sering mengkonfigurasi RSA/DSA dengan cara cepat dan kotor, menghasilkan passwordless login, tetapi juga membangun lubang keamanan besar pada pelaksanaannya.

Apakah autentifikasi RSA/DSA itu?

SSH, lebih spesifiknya OpenSSH (salah satu implementasi gratis dari SSH) merupakan suatu kakas yang sangat luar biasa. Seperti telnet atau rsh, ssh client dapat digunakan untuk melakukan log in pada suatu remote mesin. Yang dibutuhkan untuk menjalankannya pada remote ssh ialah sshdm sebuah server ssh pemroses. Tidak seperti telnet protokol ssh sangatlah aman. Ia menggunakan algoritma sepesial untuk menenkripsi aliran data, memastikan integritas aliran data dan bahkan melakukan autentifikasi dengan cara yang aman (baik dari segi pemrosesannya/internal maupun dari segi keamanan eksternal).

Meskipun ssh sangatlah hebat, ada beberapa komponen dari fungsionalitas ssh yang sering diabaikan, bahkan lebih berbahaya lagi salah digunakan, atau salah dimengerti. Komponen ini adalah OpenSSH's RSA/DSA key authentification sistem, cara alternatif dari sistem autentifikasi standar secure password yang digunakan OpenSSH secara default.

Protokol autentifikasi RSA/DSA pada OpenSSH berdasarkan pada sepasang kunci yang dibangkitkan oleh generator kunci, yang disebut dengan kunci publik dan kunci privat. Keuntungan dari menggunakan sistem autentifikasi berbasis kunci ialah dalam banyak kasus kita dapat menciptakan suatu secure connection tanpa harus mengetikkan password secara manual.

Meskipun protokol berbasis kunci ini relatif aman, masalah muncul ketika user menggunakan *shorcut - shortcut* atas nama kemudahan, tanpa memperhatikan implikasi sekuritas. Selanjutnya kita akan melihat bagaimana menggunakan protokol RSA dan DSA dengan benar tanpa mengekepos diri kita dengan resiko keamanan yang tidak perlu.

Bagaimana cara kerja kunci RSA/DSA

Selanjutnya akan diperlihatkan cara kerja dari kunci RSA/DSA secara umum. Kita mulai dengan skenario hipotesis dimana kita menggunakanautentifikasi RSA untuk memperbolehkan lokal linux workstation (yang dinamakan *localbox*) untuk membuka remote shell pada *remotebox*. Pada saat kita akan mencoba untuk terhubung dengan remotebox dengan ssh client akan muncul prompt sebagai berikut :

```
% ssh drobbins@remotebox
drobbins@remotebox's password:
```

Disini kita melihat sebuah contih bagaimana cara ssh menangani autentifikasi secara default. Ia menanyakan password akun drobbin pada remotebox. Jika kita menuliskan password kita untuk remotebox maka ssh akan menggunakan protokol secure password untuk mengirim pasword tersebut untuk diverifikasi. Tidak seperti apa yang dilakukan telnet, disini password kita dienkripsi sehingga tidak dapat diintersep oleh *sniffer* (seseorang yang mendengarkan koneksi data kita. Ketika remotebox mencocokan password yang kita berikan dengan database yang dimilkinya, jika cocok maka kita boleh untuk log on pada sistem dan disambut dengan shell prompt remotebox. Meskipun metode default yang digunakan untuk autentifikasi user sudah cukup aman, RSA dan DSA datang dengan beberapa kemungkinan baru. Tidak seperti autentifikasi ssh secure password, autentifikasi RSA membutuhkan beberapa inisial konfigurasi. Kita cukup melakukan konfigurasi ini sekali. Setelah itu autentifikasi RSA antara localbox dan remotebox akan 'tidak terasa'. Untuk mengatur autentifikasi RSA, pertama kita harus membangkitkan

pasangan kunci, kunci publik dan kunci privat. Kedua kunci ini mempunyai atribut yang menarik. Kunci publik hanya dapat digunakan untuk mengenkripsi dan kunci privat hanya dapat digunakan untuk mengenkripsi pesan yang dkodekan oleh kunci publik pasangannya. Protokol autentifikasi RSA (dan DSA) menggunakan sifat unik ini untuk melakukan secure authentication, tanpa harus mengirimkan informasi rahasia apapun melalui jaringan. Untuk menjalanakan autentifikasi RSA atau DSA, kita melakukan cukup melakukan konfigurasi sekali pada waktu pertama kali. Kita tanamkan kunci publik kita pada remote box. Kunci ini disebut "publik" karena satu alasan, karena kunci ini hanya bisa melakukan enkripsi pesan untuk kita, kita tidak perlu mengkhawatirkan bahwa kunci ini jatuh ke pihak yang salah. Setelah kita menanamkan kunci tersebut pada remotebox dan menempatkannya pada file khusus (~/.ssh/authorized_keys) sehingga sshd dapat menemukannya, kita sudah siap untuk melakukan autentifikasi RSA untuk *log on* pada remotebox.

Untuk melakukannya, kita cukup mengetikkan ssh drobbins@remotebox pada konsole localbox. Pada saat itu, ssh memberitahu sshd remotebox bahwa kita akan menggunakan autentfikasi RSA. Apa yang terjadi selanjutnya akan lebih menarik. Sshd pada remotebox akan membangkitkan sebuah bilangan acak, dan kemudian mengenkripsinya dengan menggunakan kunci publik yang sudah kita tanamkan sebelumnya. Kemudian dia akan mengirimkan bilangan yang sudah terenkripsi tersebut ke ssh pada localbox. Selanjutnya, ssh kita dengan menggunakan kunci privat mendekripsi pesan ini dan mengirimkan kembali ke remotebox, pesan tersebut kira-kira berisi sebagai berikut :

"Lihat, saya benar-benar mempunyai kunci privat pasangannya; saya dengan sukses telah mendekripsi pesan Anda!".

Akhirnya sshd menyimpulkan bahwa kita diperbolehkan untuk login pada remotebox tersebut, karena kita mempunyai pasangan kunci privatnya. Fakta bahwa kita mempunyai kunci privat pasangannya memberikan kita akses kedalam remotebox.

Dua Observasi

Terdapat dua observasi penting pada autentifikasi RSA dan DSA. Pertama kita cukup membangkitkan satu pasang kunci. Kemudian kita dapat menanamkannya pada seluruh remote mesin yang akan kita akses dan mereka (seluruh remote mesin tersebut) akan cocok dengan satu-satunya pasangan yakni kunci privat yang kita miliki. Dengan kata lain, kita tidak membutuhkan sepasang kunci untuk setiap sistem yang kita ingin akses. Satu pasang cukup untuk semuanya.

Pengamatan yang kedua ialah bahwa kunci privat kita tidak boleh jatuh ke tangan yang salah. Kunci privat merupakan satu-satunya hal yang memungkinkan kita untuk mengakses ke seluruh remote sistem kita, dan semua orang yang memilki kunci privat kita akan mempunyai diberikan hak yang sama dengan kita. Seperti halnya kita tidak ingin orang asing mempunyai kunci rumah kita, kita harus menjaga kunci privat kita dari unauthorized use. Pada dunia bit – bit dan byte – byte, hal ini berarti tidak seorangpun dapat membaca ataupun meng-copynya.

Tentu saja, pengembang ssh ini sadar akan pentingnya kunci privat, dan telah membangun beberapa pengaman pada ssh dan sshkey-gen sehingga kunci privat kita tidak dapat disalahgunakan. Pertama, ssh telah dikonfigurasi untuk mencetak pesan peringatan yang besar jika permissions file tempat kita meletakkan kunci dapat dilihat semua orang. Kedua, ketika kita membuat pasangan kunci publik/privat dengan menggunakan ssh-keygen, ssh-keygen akan meminta kita untuk memasukkan passphrase. Jika kita melakukannya, maka kunci privat kita akan dienkripsi dengan menggunakan passphrase tersebut, jadi meskipun kunci tersebut dicuri, kunci tersebut akan menjadi tidak berguna jika orang tersebut tidak mengetahui passphrase tersebut. Dengan berbekal pengetahuan diatasm kita dapat melihat bagaimana mengkonfigurasi ssh untuk menggunakan protokol autentifikasi RSA dan DSA.

Lebih lanjut dengan ssh-keygen

Langkah pertama dalam mengatur autentifikasi RSA dimulai dengan membangkitkan pasangan kunci publik/privat. Autentifikasi RSA merupakan bentuk original dari autentifikasi kunci dari ssh, jadi RSA dapat bekerja pada versi OpenSSH manapun, walaupun lebih disarankan pada versi yang paling baru. Bangkitkan pasangan kunci RSA seperti berikut ini :

```
% ssh-keygen
Generating public/private rsa1 key
pair.
Enter file in which to save the key
(/home/drobbins/.ssh/identity): (hit
enter)
Enter passphrase (empty for no
```

```
passphrase): (enter a passphrase)
Enter same passphrase again: (enter it
again)
Your identification has been saved in
/home/drobbins/.ssh/identity.
Your public key has been saved in
/home/drobbins/.ssh/identity.pub.
The key fingerprint is:
a4:e7:f2:39:a7:eb:fd:f8:39:f1:f1:7b:fe
:48:a1:09 drobbins@localbox
```

Ketika ssh-keygen menanyakan lokasi default dari kunci tersebut, kita akan menekan enter untuk menerima defaultm dari /home/drobbins/.ssh/identity. Ssh-keygen akan menyimpan kunci privat pada path tersebut, dan kunci publik akan disimpan juga di sebelahnya pada file identity.pub.
Perhatikan juga bahwa ssh-keygen meminta kita untuk memasukkan passphrase (tujuh atau lebih karakter-sulit-dipredeksi). Ssh-keygen kemudian mengenkripsi kunci privat kita (~/.ssh/identity) dengan passphrase tersebut sehingga akan menjadi tidak berguna bagi orangyang tidak mengetahui passphrase tersebut.

Ketika kita menyebutkan passphrase, hal tersbut juga memperbolehkan ssh-keygen untuk mengamankan kunci privat kita dari penggunaan yang tidak benar, namun juga menyebabkan ketidaknyamanan. Sekarang, setiap kali kita akan mencoba terhubung dengan akun drobbins@remotebox dengan menggunakan ssh, ssh akan meminta kita untuk memasukkan passphrase yang dibutuhkan untuk mendekripsi kunci privat kita. Setelah kunci privat kita terdekripsi, ssh client akan mengurus sisanya. Meskipun mekanisme antara penggunaan remote password dan RSA passphrase sangat berbeda, namun dalam prakteknya kita tetap akan diminta untuk menuliskan “secret phrase” pada ssh.

```
# ssh drobbins@remotebox
Enter passphrase for key
'/home/drobbins/.ssh/identity': (enter
passphrase)
Last login: Thu Jun 28 20:28:47 2001
from localbox.gentoo.org

Welcome to remotebox!

%
```

Disini banyak orang salah mengerti dalam mengambil jalan pintas. Orang-orang membuat kunci privat mereka dalam keadaan tidak terenkripsi sehingga mereka tidak perlu lagi mengetikkan password. Dengan cara tersebut, mereka cukup mengetikan perintah ssh, dan mereka akan langsung terautentifikasi via RSA (atau DSA) dan kemudian sudah *logged in*.

```
# ssh drobbins@remotebox
Last login: Thu Jun 28 20:28:47 2001
from localbox.gentoo.org

Welcome to remotebox!

%
```

Meskipun menjadi lebih nyaman, sebaiknya Anda tidak menggunakan pendekatan ini apabila tidak mengerti implikasi dari segi keamanannya. Dengan kunci privat yang tidak terenkripsi, jika seseorang meng-hack ke localbox, mereka otomatis dapat mengakses remotebox dan sistem laiinya yang sudah dikonfigurasi dengan kunci publik.
Lebih lanjut akan dijelaskan bagaimana keuntungan paswordless authentification tanpa harus mengorbankan keamanan kunci privat.

Pembangkitan pasangan kunci RSA

Untuk membuat pasangan kunci pada autentifikasi RSA kita harus melakukan hal ini sekali saja. Ketikkan pada konsol :

```
% ssh-keygen
```

Terima lokasi default kunci ketika diminta (biasanya ~/.ssh/identity dan ~/.ssh/identity.pub untuk kunci publik), dan sediakan ssh-keygen dengan passphrase yang aman. Sekali hal ini dilakukan , Anda telah mempunyai kunci publik dan kunci privat yang terenkripsi dengan passphrase tersebut.

Instalasi kunci publik RSA

Sekarang kita harus mengkonfigurasi remote sistem dengan menggunakan sshd untuk menggunakan kunci publik kita untuk autentifikasi. Biasanya, hal ini dilakukan dengan meng-copykan kunci publik ke remote sistem sebagai berikut :

```
% scp ~/.ssh/identity.pub
drobbins@remotebox:
```

Karena autentifikasi RSA belum sepenuhnya terset, kita akan diminta untuk memasukkan password pada remotebox. Lakukan. Lalu log in pada remotebox dan kemudian append kunci

publik tersebut ke file ~/.ssh/authorized_keys seperti dibawah ini :

```
% ssh drobbins@remotebox
drobbins@remotebox's password: (enter
password)
Last login: Thu Jun 28 20:28:47 2001
from localbox.gentoo.org

Welcome to remotebox!

% cat identity.pub >>
~/.ssh/authorized_keys
% exit
```

Sekarang, dengan autentifikasi RSA yang sudah terkonfigurasi, kita akan diminya untuk memasukkan RSA passphrase (bukan password kita) ketika kita akan mencoba untuk terhubung dengan remotebox dengan menggunakan ssh

```
% ssh drobbins@remotebox
Enter passphrase for key
'/home/drobbins/.ssh/identity':
```

Konfigurasi autentifikasi RSA telah selesai. Jika Anda tidak diminta passphrase ada beberapa cara yang dapat dilakukann. Pertama, coba login dengan mengetikkan ssh -1 drobbins@remotebox. Ini akan memberitahu ssh hanya untuk mengunakan versi 1 dari protokol ssh dan mungkin juga dibutuhkan jika ada beberapa alasan bahwa remote sistem melakukan default autentifikasi DSA. Jika tetap tidak berjalan, pastikan bahwa tidak ada tulisan RSAAuthentification no pada /etc/ssh/ssh_config. Jika ya komentari baris tersebut dengan menambahkan tanda pagar '#' pada awal baris. Atau, tanyakan remotebox sistem administrator bahwa ia telah menyalakan autentifikasi RSA dan telah mengkonfigurasi nya pada /etc/ssh/sshd_config.

Pembangkitan kunci DSA

Sementara kunci RSA digunakan untuk versi 1 pada protokol ssh, kunci DSA digunakan pada protokol level 2, versi baru dari protokol ssh. OpenSSH model terbaru dapat menggunakan keduanya, kunci RSA dan DSA. Membangkitkan kunci DSA dengan OpenSSH ssh-keygen dapat dilakukan seperti pada RSA, seperti dibawah ini:

```
% ssh-keygen -t dsa
```

Lagi, kita akan diminta untuk memasukkan passphrase. Masukkan. Kita juga akan diminta untuk memasukkan lokasi tempat penyimpanan kunci DSA. Secara default akan ditempatkan pada ~/.ssh/id_dsa dan ~/.ssh/id_dsa.pub. setelah membangkitkan pembangkitan kunci selesai, maka selanjutnya kita menginstall kunci publik DSA kita pada remote sistem.

Instalasi kunci publik DSA

Sekali lagi, instalasi kunci publik DSA identik dengan RSA. Untuk DSA, kita meng-copy file ~/.ssh/id_dsa.pub ke remotebox dan meng-appendnya ke ~/.ssh/authorized_keys2 pada remote box. Perhatikan bahwa file ini berbeda dengan file RSA authorized_keys. Setelah selesai kita menkonfigurasi, kita dapat log in ke remotebox dengan mengetikkan passphrase dan bukan password kita sebenarnya pada remotebox.

Pengenalan ssh-agent

Ssh-agent, yang juga sudah terdapat pada distribusi OpenSSH, merupakan suatu program yang dirancang untuk RSA dan DSA agar aman dan menyenangkan. Ssh-agent, tidak seperti ssh, merupakan suatu long-running daeomn yang bertujuan untuk men-cache kunci privat yang sudah terdekripsi.

Ssh, termasuk built-innya yang memungkinakan ssh untuk berkomunikasi dengan ssh-agent, memugkinkan ssh untuk mendapatkan kunci privatyang terdekripsi tanpa harus meminta Anda memasukkan passphrase untuk setiap koneksi baru. Dengan ssh-agent, Anda cukup menggunakan ssh-add untuk memasukkan kunci privat Anda ke cache ssh-agent. Itu merupakan proses sekali saja; setelah menggunakan ssh-add; ssh akan mengambil kunci privat Anda dari ssh-agent.

Menggunakan ssh-agent

Mari kita lihat cara kerja keseluruhan dari sistem cache dari ssh-agent. Ketika ssh-agent dijalankan, ia memberikan beberapa variabel-variabel lingkungan penting sebelum terlepas dari shell dan kemudian berjalan pada proses background. Ini merupakan contoh keluaran yang dibangkitkan oleh ssh-agents ketika mulai :

```
% ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-
XX4LkMJS/agent.26916; export
SSH_AUTH_SOCK;
SSH_AGENT_PID=26917; export
SSH_AGENT_PID;
echo Agent pid 26917;
```

Seperti yang dapat dilihat, keluaran ssh-agent merupakan beberapa perintah pada command bash; yang jika dieksekusi, perintah-perintah ini akan mengeset beberapa varriabel lingkungan, SSH_AUTH_SOCK dan SSH_AGENT_PID. Akibat dari *include* nya perintah export, variabel-variabel lingkungan (*environment variables*)ini akan dapat digunakan untuk tambahan perintah (belakangan/tambahan) lainnya. Semua tersebut terjadi jika baris – baris ini dievaluasi oleh shell, namun sekarang mereka hanya di tampilkan ke layar. Untuk memperbaiki hal ini, kita dapat memanggil ssh-agent dengan cara berikut ini :

```
eval `ssh-agent`
```

Perintah ini akan memberitahu bash bahwa untuk menjalankan ssh-agent dan kemudian mengevaluasi keluaran ssh-agent. Memanggil dengan cara ini (dengan kutip belakang `, bukan kutip biasa ‘), variabel SSH_AUTO_SOCK dan SSH_AGENT_PID get set dan dieksport oleh shell, sehingga variabel tersedia untuk setiap proses baru yang dilakukan pada waktu sesi login(*session login*).
Cara terbaik untuk menjalankan ssh-agent ialah dengan dengan menambahkan garis atas pada ~/.bash_profile; dengan cara tersebut, semua program yang mulai pada sesi login Anda akan melihat variabel – variabel lingkungan, dapat menemukan ssh-agent dan men-*query* kuncinya jika diperlukan. Salah satu variabel lingkungan yang penting ialah SSH_AUTO_SOCK; SSH_AUTO_SOCK berisi path ke UNIX domain socket yang mana ssh dan scp gunakan untuk membangun komunikasi dengan ssh-agent.

Menggunakan ssh-add

Ssh-agent start-up dengan cache yang masih kosong (dari beberapa kunci privat yang telah didekripsi). Sebelum kita benar – benar menggunakan ssh-agent. Pertama – tam kita harus menambahkan kunci privat kita ke cache ssh-agent dengan perintah ssh-add. Pada contoh berikut, digunakan perintah ssh-add untuk menambahkan ~/.ssh/identity kunci privat RSA ke cache ssh-agent :

```
# ssh-add ~/.ssh/identity
Need passphrase for
/home/drobbins/.ssh/identity
Enter passphrase for
/home/drobbins/.ssh/identity
(enter passphrase)
```

Seperti yang terlihat, ssh-add meminta passphrase sehingga kunci privat, dapat didekripsi dan di simpan pada cache ssh-agent, dapat digunakan. Sekali menggunakan ssh-add untuk menambahkan kunci privat ke cache ssh-agent, maka SSH_AUTO_SOCK didefinisikan pada current shell, kemudian Anda dapat menggunakan scp dan ssh untuk membangun koneksi dengan remote sistem tanpa harus memberikan passphrase Anda.

Keterbatasan dari ssh-agent

Ssh-agent sangatlah berguna, namun dengan konfigurasi defaultnya maka hal tersebut meninggalkan beberapa inconvinience. Perhatikan hal berikut ini.
Pertama, dengan eval `ssh-agent` pada ~.bash_profile, duplikasi baru dari ssh-agent diluncurkan setiap sesi login; tidak saja membuang-buang resource namun kita juga harus menambahkan kunci privat ke dalam setiap duplikasi baru tersebut. Jika hanya membuka satu terminal atau satu konsol mungkin hal ini bukanlah masalah, namun banyak dari kita membuka beberapa terminal dan sehingga harus mengetikkan passphrase setiap kita membuka konsol baru. Sebenarnya tidak ada alasan mengapa hal tersebut harus dilakukan bahwa satu proses ssh-agent saja seharusnya sudah mencukupi.
Problem lain dengan konfigurasi default ialah ssh-agent tidak *compatible* dengan *cron jobs*. Karena cron jobs dimulai dengan cron process, mereka tidak akan menurunkan variabel SSH_AUTO_SOCK dari lingkungan mereka, jadi tidak mengetahui bahwa proses ssh-agent sedang berjalan atau tidak tahu bagaimana memanggilnya.

Memasukkan keychain

Untuk mengatasi masalah ini Danniel Robbins menulis handy bash-based ssh-agent front-end yang disebut dengan *keychain*. Apa yang membuat keychain spesial ialah fakta bahwa ia dapat membuat kita dapat menggunakan proses ssh-agent per sistem, bukan hanya per sesi login. Ini berarti kita cukup sekali melakukan ssh-add tiap kunci privat. Selanjutnya, keychain bahkan turut membantu optimasi proses ssh-add hanya dengan cara mencoba menambahkan kunci – kunci privat yang belum ada pada cache ssh-agent. Berikut ini adalah cara kerja keychain.

Ketika memulai dari ~/.bash_profile, ia akan mengecek apakah ssh-agent sudah berjalan. Jika belum, ia akan menjalankannya dan menyimpan variabel SSH_AUTO_SOCK dan SSH_AGENT_PID pada file ~/.ssh-agent untuk penyimpanan yang aman dan penggunaan belakangan lainnya. Berikut adalah cara memulai keychain; seperi menggunakan ssh-agent awal, kita melakukan seting pada ~/.bash_profile :

```
#!/bin/bash
#example ~/.bash_profile file
/usr/bin/keychain ~/.ssh/id_rsa
#redirect ~/.ssh-agent output to
/dev/null to zap the annoying
#"Agent PID" message
source ~/.ssh-agent > /dev/null
```

Seperti yang terlihat kita mengambil langsung file ~/,ssh-agent daripada mengevaluasi keluaran seperti yang dilakukan pada saat menggunakan ssh-agent secara langsung. Bagaimanapun hasilnya tetap sama – SSH_AUTO_SOCK didefinisikan, dan ssh-agent berjalan dan siap untuk digunakan. Dan karena SSH_AUTO_SOCK disimpan pada ~/.ssh-agent, maka script shell dan cron jobs dapat terhubung dengan ssh-agent dengan mengambil dari file ~/.ssh-agent. Keychain pun juga mendapat keuntungan; kita ingat bahwa ketika keychain dijalankan, ia melihat apakan keychain sudah dijalankan, jika ya, ia menggunakan file ~/.ssh-agent untuk mendapatkan setting SSH_AUTO_SOCK yang cocok, jadi menggunakan ssh-agent yang lama daripada membuat ssh-agent yang baru. Keychain akan memulai proses baru ssh-agent hanya jika file ~/ssh-agent stale (titik dimana non-existent ssh-agent) atau jika ssh-agent itu sendiri tidak eksis.

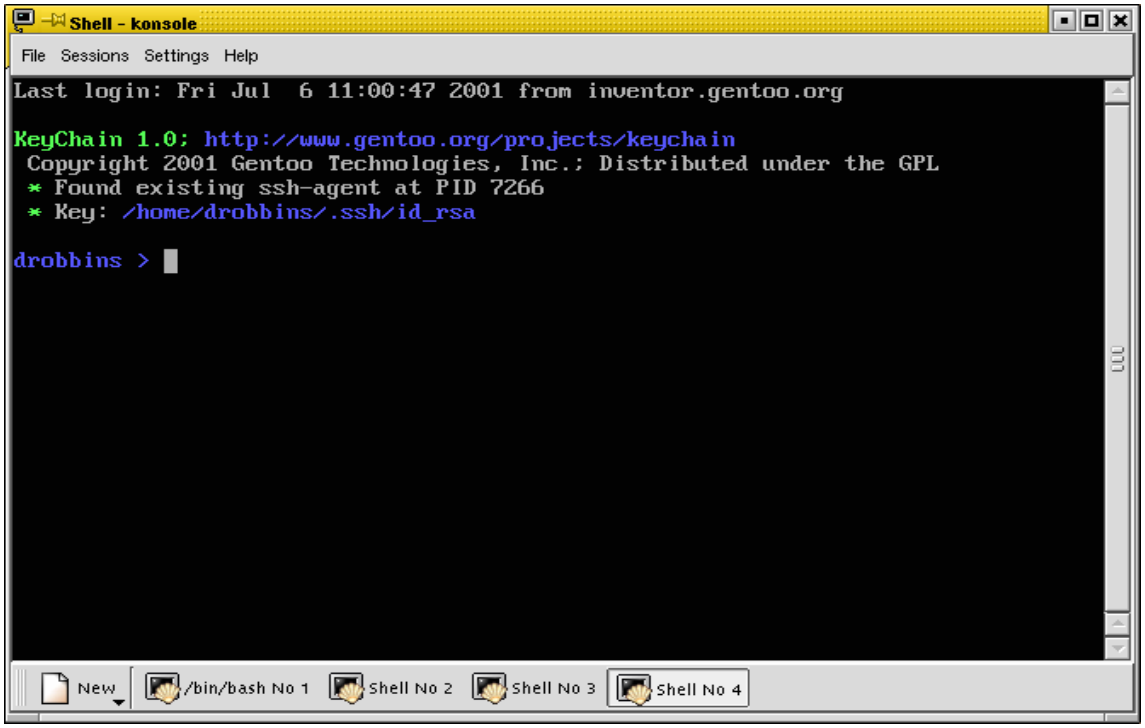
Instalasi keychain
Proses instlasinya sangatlah mudah. Tuju situs keychain dan ambil versi terbarunya. Kemudian diinstal dengan cara seperti berikut :

```
# tar xzvf keychain-1.0.tar.gz
# cd keychain-1.0
# install -m0755 keychain /usr/bin
```

Sekarang keychain berada pada /usr/bin, tambahkan path tersebut pada ~/.bash_profile, beri path kunci privat sebagai argumen.

```
#!/bin/bash
#on this next line, we start keychain
and point it to the private keys that
#we'd like it to cache
/usr/bin/keychain ~/.ssh/id_rsa
~/.ssh/id_dsa
source ~/.ssh-agent > /dev/null
#sourcing ~/.bashrc is a good thing
source ~/.bashrc
```

Keychain in action
Konfigurasi ~/.bash_profile untuk memanggil keychain pada saat login, logout, dan logback. Ketika dijalankan, keychain akan menjalankan ssh-agent, menyimpan setting variabel lingkungan pada ~/.ssh-agent, dan meminta passphrase untuk setiap kunci privat yang tertera pada keychain command line pada ~/.bash_profile.



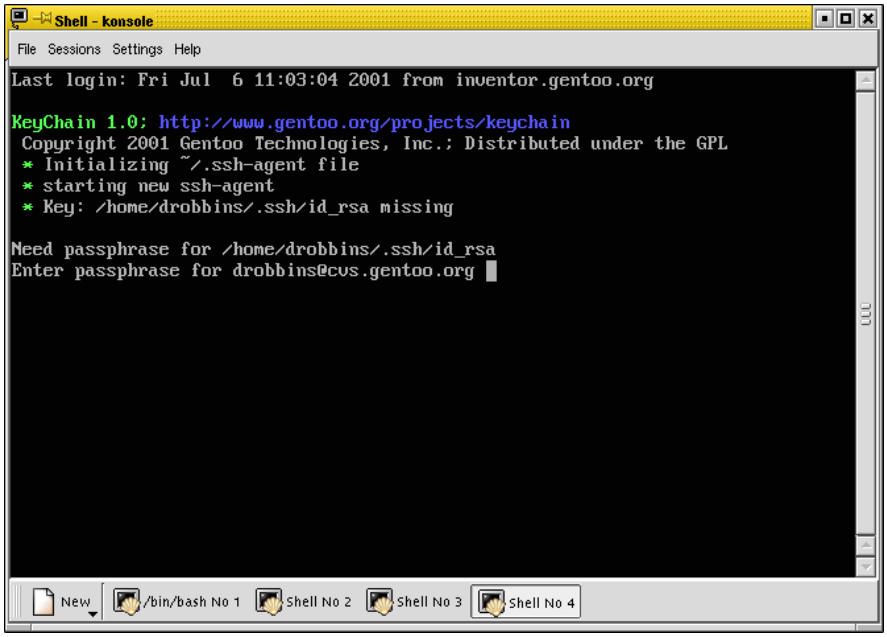
Setelah memasukkan passphrse, kunci – kunci privat akan dicache, kemudian keychain akan keluar. Lalu, ~/.ssh-agent akan di sourced, untuk menginisialisasi sesi login untuk digunakan dengan ssh-agent. Sekarang jika Anda logout atau logback, terlihat bahwa keychain akan menemukan proses ssh-agent; ia tidak akan mati

Sekarang Anda login dan dapat melakukan ssh dan scp ke remote sistem, tidak perlu menggunakan ssh-add setelah login, dan juga ssh dan scp tidak meminta passpharse. In fact, selama proses inital ssh-agent tetap berjalan, Anda dapat membangun sebuah koneksi tanpa memberikan password. Dan proses ssh-agent tersebut akan terus berjalan hingga mesin tersebut di reboot, karena setting ini berjalan pada sistem linux, kemungkinan Anda tidak perlu untuk memasukkan password untuk beberapa bulan.
Lanjutkan dan buat beberapa sesi login baru, terlihat bahwa keychain tetap akan “tersangkut” pada proses ssh-agent yang sama. Anda juga jika logout. Tambahan, keychain akan memvervikasi kunci – kunci privat yang tertera pada cache ssh-agent. Jika tidak, Anda akan diminta passpharse kembali, namun jika semua berjalan dengan lancar, Anda tidak akan diminta password untuk login.

dapat menyangkutkan cron jobs dan script pada proses ssh-agent yang sama. Untuk menggunakan perintah ssh atau scp dari shell scripts atau cron jobs, lakukan source ke file ~/.ssh-agent :

```
source ~/.ssh-agent
```

Sekarang setiap perintah ssh atau scp dapat menemukan ssh agent-yang-sedang-berjalan dan dapat membangun koneksi parwordless seperti pada shell.



Keychain options

Setelah keychain berjalan, beri argumen --help untuk melihat option lebih lanjut. Salah satu opsinya ialah --clear.

Seperti yang sudah diutarakan sebelumnya, penggunaan kunci-privat-yang-tidak-terenkripsi sangatlah berbahay, sebab hal tersebut memungkinkan seseorang mencuri kunci privat Anda dan menggunakannya untuk login ke akun remote Anda dari sistem lain tanpa memberikan password. Dengan keychain hal ini tentu saja dapat diatasi namun terdapat celah keamanan dari prinsip kerja keychain itu sendiri yakni bahwasanya keychain menyangkutkan dirinya pada proses ssh-agent-yang-berjalan. Apa yang dapat terjadi jika entah bagaimana seseorang mengetahui passpharse atau password sehingga dapat login ke lokal sistem Anda? Jika entah bagaimana ia dapat login dengan akun Anda dan kemudian keychain memberikan akses ke kunci-privat-anda-yang-terdekirpsi.

Mari kita tinjau dari berbagai sisi. Jika ada *malicious* user yang bisa masuk sebagai Anda maka keychain akan memberikan akses ke akun remote Anda. Namun begitu, untuk mencuri kunci-privat-yang-terdekripsi pun tetap sulit karena disimpan pada tempat yang terenkripsi. Juga untuk mendapatkan akses ke kunci privat juga membutuhkan login sebagai Anda sepenuhnya, tidak cukup hanya dengan membaca file dari directory. Jadi mengekssploitasi ssh-agent lebih susah dari pada mengambil kunci-privat-yang-tidak-terenkripsi, yakni hanya membutuhkan akses ke file ~/.ssh, apakah login sebagai Anda atau bukan. Jadi jika ada seseorang yang login sebagai diri Anda ia akan dapat menimbulkan kerusakan dengan kunci-privat-Anda-yang-tidak-terdekripsi. Jika Anda menggunkan keychain pada server yang tidak terlalu sering Anda login maka aktifkan --clear untuk menambahkan layer pengamanan.

Opsi --clear akan menganggap semua login baru ke akun Anda akan dianggap sebagai ancaman sekuritas yang potensial. Ketika keychain dijalankan dengan opsi --clear, maka ia akan mem-flush seluruh kunci privat Anda yang ada pada cache ssh-agent, sebelum menjalanakan tugasnya. Oleh sebab itu jika Anda seorang penyudup maka Anda akan diminta untuk memasukkan password, bukan mendapatkan akses ke cache kunci Anda. Tentu saja hal ini seperti melakukan ssh-agent biasa tanpa menggunakan keychain. Meskipun meningkatkan tingkat sekuriti namun ini berarti menyebabkan adanya ketidaknyamanan. Makin tinggi tingkat keamanan maka (kemungkinan besar) makin tinggi pula ketidaknyamanan.

Dibalik itu, penggunaan keychain tetap menguntungkan sebab dengan opsi --clear, cron jobs dan script Anda tetap dapat membuat koneksi yang passwordless sebab keychain melakukan flushing pada saat login bukan logout. Penggunaan opsi ini cocok dilakukan pada server – server yang jarang diakses seperti server backup, firewall, dan routers.

Meningkatkan keamanan ssh

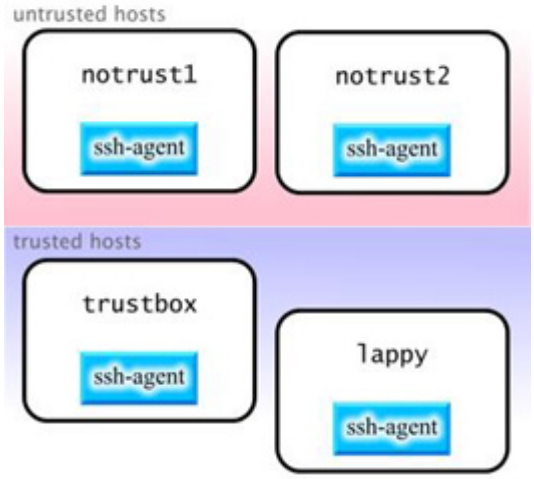
Menjalankan ssh-agent pada untrusted-machines cukup berbahaya. Jika seseorang mempunyai akses ke root pada sistem tersebut, maka kunci yang terdekripsi dapat diekstrak dari ssh-agent. Namun demikian, mengekstrak kunci tersebut sulit dilakukan, dibutuhkan skill profesional cracker. Dengan fakta tersebut bahwa pencurian kunci privat sangat mungkin terjadi maka kita harus memprioritaskan hal tersebut.

Untuk memformulasikan strategi untuk melindungi kunci privat, kita harus menkategorikan mesin ke dalam salam satu dari dua kategori ini. Jika sebuah host well-secured atau isolated – melakukan eksploitasi dengan root hampir tidak mungkin – maka host tersebut dikategorikan sebagai *trusted host*. Sebaliknya jika mesin tersebut digunakan oleh banyak orang da kemungkinan untuk dibobol besar maka kategorikan mesin tersebut sebagai untrusted host. Untuk melinduni kunci privat Anda dari ekstraksi, ssh-agent dan keychain oleh karena itu tidak boleh berjalan pada untrusted host. Dengan cara itu, meskipun keamanan sistem tersebut bobol, maka tidak ada ssh-agent yang dapat digunakan oleh penyusup untuk mengekstraksi kunci – kunci Anda.

Namun hal ini menjadi masalah. Jika kita tidak dapat menjalankan ssh-agent pada untrusted host lantas bagaimana kita membangun suatu koneksi-yang-tidak-berpassword dari sistem ini? Jawabannya ialah dengan memasangkan ssh-agent dan keychain pada trusted host. Dengan kata lain, melakukan pengalihan tugas dari sesi ssh pada suatu ssh-agent yang berjalan pada trusted sistem.

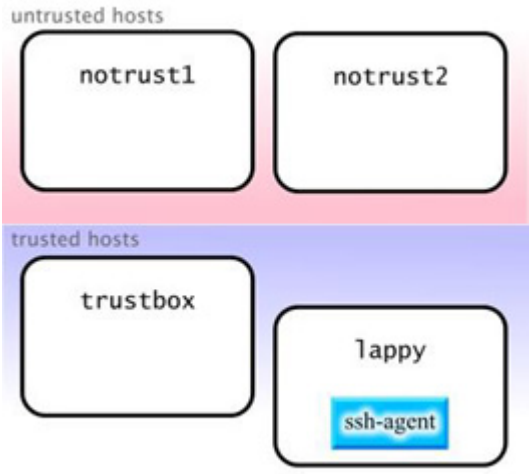
Autentikasi agent forwarding

Untuk mengerti bagaimana cara kerja autentifikasi forwarding, ambil contoh misalkan terdapat user drobbins yang mempunyai trusted laptop, disebut dengan lappy, trusted serer yang disebut dengan trusbox, dan dua untrusted sistem yang disebut dengan notrust 1 dan notrust 2. Ia menggunakan ssh-agent pada keempat sistem tersebut seperti tampak pada gambar berikut :



ssh-agent running on trusted and untrusted machines

Masalah yang terjadi dengan hal ini ialah jika seseorang mempunyai akses root pada notrust 1 dan no trust 2 maka tentu saja sangat mungkin orang tersebut untuk mengekstrak kunci dari proses ssh-agent. Untuk mengatasi masalah ini drobbins mengehentikan proses ssh-agent yang berada pada notrust1 dan notrust 2. Bahkan agar lebih berhati-hati drobbins hanya menggunakan ssh-agent pada lappy saja. Pembatasan exposure pada kunci-privat-yang-terdekripsi ini, melindungi dia dari pencurian kunci privat.



ssh-agent running only on lappy; a more secure configuration

Tentu saja, dengan pendekatan seperti ini, drobbins sekarang hanya dapat membangun koneksi passwordless dari lappy. Selanjutnya bagaimana kita memjalankan autentifikasi forwarding dan mengatasi masalah ini.
Misalkan semua sistem menggunakan OpenSSh versi terbaru maka masalah ini dapat diatasi dengan melakukan autentifikasi forwarding. Autentifikasi forwarding memungkinkan proses ssh-agent untuk memanggil ssh-agent yang berjalan pada lokal trusted mesin. Hal ini memungkinkan Anda untuk menjalankan ssh-agent pada satu mesin, dan berarti seluruh koneksi ssh yang berasal (baik itu langsung ataupun tidak langsung) dari mesin ini akan menggunakan ssh-agetn lokal.
Untuk meng-enable autentifikasi forwarding ini, tambahkan baris dibwah ini pada lappy dan trustbox /etc/ssh/ssh_config. Perhatikan bahwa file ini untuk konfigurasi ssh (ssh_config), bukan ssh daemon (sshd_config) :

Listing 1. Add this line to your /etc/ssh/ssh_config

```
ForwardAgent Yes
```

Dengan autentifikasi forwarding maka drobbins dapat terhubung dari lappy ke trustbox, dan dari trusbox ke notrust 1 tanpa harus menyediakan passphrase untuk semua koneksi. Kedus proses ssh "tap in" ke ssh-agent yang berjalan pada lappy :

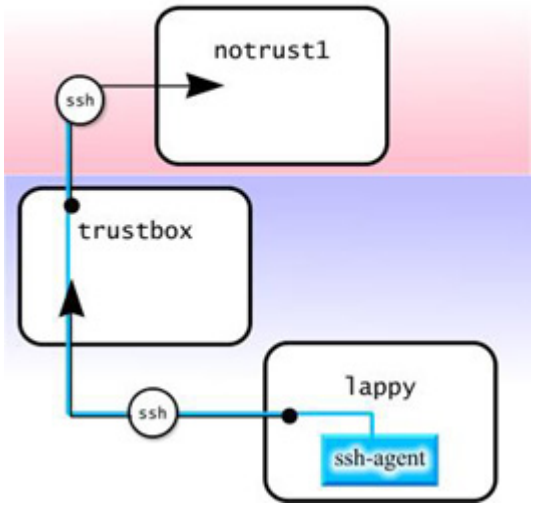
Listing 2. Tapping lappy

```
$ ssh drobbins@trustbox
Last login: Wed Sep 26 13:42:08 2001
from lappy

Welcome to trustbox!
$ ssh drobbins@notrust1
Last login: Tue Sep 25 12:03:40 2001
from trustbox

Welcome to notrust1!
$
```

Jika agent forwarding tidak bekerja, tambahkan -A untuk menyatakan secara eksplisit bahwa meng-enable autentifikasi forwarding. Berikut ini gambar yang terjadi saat drobbins login ke trustbox dan notrust 1 dengan autentifikasi forwarding :



Agent forwarding in action

Seperti yang terlihat, ketika ssh terhubung dengan trustbox, ia tetap menjaga hubungannya dengan ssh-agent yang berjalan pada lappy. Ketika koneksi ssh dibangun dari trustbox ke notrust 1, proses baru ssh ini menjaga koneksi autentifikasi dari ssh sebelumnya, sehingga memperpanjang rantai. Rantai ini dapat terus diperpanjang tergantung dari setting pada notrust 1 /etc/ssh/ssh_config. Sepanjang agent forwarding di-enable maka selruh mata rantai akan dapat melakukan autentifikasi dengan ssh-agent yang berjalan lappy.

Keuntungan dari agent connection forwarding

Adapun keuntungan dari yang diberikan oleh agent connection forwarding adalah :

1. Kunci privat hanya disimpan pada trusted mesin. Hal ini mencegah malious user untuk mengambil kunci-yang-terenkripsi di disk dan usaha untuk meretasnya.
2. Ssh-agent hanya berjalan pada trusted mesin. Hal ini mencegah penyusup untuk melakukan memory dump dari proses ssh-agent dan kemudian mengekstrasi kunci-privat-yang-terdekripsi dari hasil dump tersebut.
3. Karena hanya mengetikan passphrase pada trusted mesin maka mencegah pencurian passphrase dengan menggunakan keystroke loggers yang umumnya tertanam pada untrusted mesin.

Agent connection forwarding ini tidak memecahkan masalah dari memperbolehkan cron jobs untuk menggunakan autentifikasi RSA/DSA. Untuk mengatasi masalah ini ialah dengan cara mengeset semua cron jobs sehingga mereka semua mengeksekusinya dari trusted mesin pada suatu LAN. Bahkan jika perlu cron jobs dapat menggunakan ssh untuk terhubung ke remote sistem untuk melakukan backup otomatis, sinkronisasi file, dsb.

X11 forwarding

Protocol ssh juga mampu memforward koneksi X11, memungkinkan anda untuk secara aman menampilkan remote X11apps secara lokal. Sekali lagi, opsi ini secara default masih belum nyala karena alasan keamanan. Hal ini juga membutuhkan server end yang mempunyai xauth binary accesible untuk mengeset autentifikasi MIT-MAGIC-COOKIE 1 untuk server X Anda. Berikut ini cara konfigrasi filenya :

```
source ~/.ssh-agent
ssh -X hachi xclock
[djm@roku ssh-tutorial]$ cat
~/.ssh/config
Host trustedhost
ForwardX11 yes
```

Port forwarding

Salah satu pengunaan fleksibel dari protokol SSH adalah port forwarding, dimana ssh memungkinkan untuk memforward sesi arbitrary TCP. Karena koneksi ini dibungkus oleh saluran ssh maka mereka fully encrypted.

Hal ini membuat port-forwarding berguna untuk menambahkan keamanan pada protokol tradional biasa. Ssh mensupport port-forwarding dari server ke client (alias lokal) dan dari client ke server (alias remote).

Lokal port forwarding memungkinkan Anda untuk memforward port pada mesin klient melalui koneksi ssh ke host dan port dimana remove ssh yang akan terhubung. Lokal port forwarding dijalankan dengan menambahkan argumen –L:localport:remotehost:remoteport. Sebagai contoh perintah ini akan membuat port 8000 terhubung dengan remote host 10.88.45.12 port 80

```
ssh -L8000:10.88.45.12:80 somehost
```

Hal dibawah ini juga dapat dilakukan untuk pada konfigurasi file :

```
Host fw.somedomain.com.au
LocalForward 8000
somehost.int.somesomain.com.au:80
```

Hal ini berguna untuk mengurus mesin yang berada di belakang firewall.

Ketika menggunakan lokal port forwarding kelakuan defaultnya hanya memungkinkan koneksi dari lokalhost ke forwarded ports. Untuk meng-enable alamat lain untuk terhubung dengan forwarded port, Anda harus menyebutkan opsi GatewayPorts. Hal ini dilakukan dengan cara mengkonfigurasi file konfigurasi klient sebagai berikut : -o gatewaysports=yes.

Sedangkan untuk remote port forwarding ialah kebalikannya. Contoh dibawah ini akan menyebabkan koneksi ke port 2500 pada remote end dan terhubung dengan 10.34.54.12 port 25 pada lokal end :

```
ssh -R2500:10.34.54.12:25 somehost
```

Dynamic port forwarding

OpenSSH juga mensupport mode yang memnugkinkan untuk melakukan dynamic port forwarding. Pada konfigurasi ini OpenSSH bekerja sebagai proxy SOCKS4 pada port tertentu. Mode ini berguna untuk menggali firewalls. Dynamic port forwarding disetup dengan –D port flag, dimana port tersebut merupakan port yang ssh client gunakan untuk me-listen SOCKS4 yang direkuest.

Perbaikan kompatibilitas shell

Seiring dengan penigkatan fungsionalitas, banyak perbaikan dilakukan dalam hal kompatibilitas shell. Pada keychain 1.0 dibutuhkan bash sedangkan pada versi selanjutnya dapat digunakan pada sh-compatible shell. Perubahan ini memungkinkan keychain untuk bekerja diluar-kotak pada hampir semua sistem UNIX, termasuk Linux, BSD, Solaris, IRIX, dan AIX.

Terdapat dua macam kompatibilitas yang harus diperbaiki. Pertama, keychain harus menggunakan built-in, ekspresi, dan operator yang disupport oleh seluruh implementasi sh, termasuk sh shell populer baik itu yang bersifat free mapun yang komeersial, zsh, dab bash versi 1 dan 2. Berikut ini listing program yang diterapkan pada keychain. Kaarena sh shell ada yang tidak mensuppor ~ sebagai user home, maka baris yang menggunakan ~ diganti dengan $HOME :

Listing 3. Making it $HOME

```
hostname=`uname -n`
pidf=${HOME}/.ssh-agent-${hostname}
cshpidf=${HOME}/.ssh-agent-csh-
${hostname}
```

Selanjtunya semua references ke source diganti dengan **.** untuk memastikan kompatibilitas dengan NerBSD /bin/sh, yang mana tidak mensupport perintah source sama sekali

Listing 4. Humoring NetBSD

```
if [ -f $pidf ]
then
    . $pidf
else
SSH_AGENT_PID="NULL"
fi
```

Kemudian, diterapkan pula beberapa perbaikan kinerja. Daripada “menyentuh” file dengan mengetikan touch foo, dapat dilakukan seperti ini :

Listing 5. Touching files

```
> foo
```

Dengan menggunakan sintaks built in daripada menggunakan eksternal binary, sebuah fork dihindari dan script tersebut menjadi lebih efisien. > foo dapat bekerja dengan sh-compatible shell manapun.

Masalah platform executable

Membuat script bekerja pada berbagai macam jenis UNIX operating sistem membutuhkan lebih dari *adhering* ke sintaks sh. Karena banyak script yang menggunakan eksternal command, seperti grep, awk, ps, dan lainnya, perintah – perintah ini harus dipanggil denfan cara se-standard mungkin. Sebagai contoh, echo yang dikenali oleh hampir seluruh UNIX mengenali opsi –e, namun solaris tidak, ia hanya menampilkan –e ke stdout ketika dipanggil. Jadi untuk mengatasi hal ini keychain melakukan auto-detects apakah opsi –e bekerja :

Listing 6. Sniffing out Solaris

```
if [ -z "`echo -e`" ]
then
    E="-e"
fi
```

Diatas E diset menjadi –e jika –e escaping disupport. kemudian echo dapat dipanggil sebagai berikut :

Listing 7. Better echo

```
echo $E Usage: ${CYAN}${0}${OFF} [
${GREEN}options${OFF} ]
${CYAN}sshkey${OFF} ...
```

Dengan menggunakan echo $E daripada echo –e, opsi –e dapat secara dinamis dienable ataupun didisable sesuai kebutuhan.

Pidof, ps

Perbaikan kompatibilitas yang paling signifikan ialah bagaimana keychain mendeteksi adanya proses ssh-agent yanng sedang bekerja. Sebelumnya digunakan perintah pidof, namun hal tersebut harus di-remove karena beberapa sistem tidak mensupport pidof.

Karena ketidak *reliable*an yang dimiliki oleh pidof maka diganti dengan pipng ps output untuk grep dan awk untuk mengambil pid proses tersebut.

Listing 8. Piping better than pidof

```
mypids=`ps uxw | grep ssh-agent | grep
-v grep | awk '{print $2}'`
```

Pipeline diatas akakn mengeset variabel mypids ke nilai – nilai dari seluruh proses ssh-agent yang dimiliki oleh current user.

Grep –v perintah grep merupakan bagian dari pipeline untuk memastikan bahwa proses grep ssh-agent tidak menjadi bagian dari list PID.

Meskipun pendekatan ini baik secara konsep, penggunaan ps membuka celah keamanan sebab opsi ps bukan merupakan standar antar various BSD dan turunan System V UNIX. Sebagai contoh ps uxw bekerja pada linux namun tidak bekerja pada IRIX. Sementara itu ps –u username –f bekerja pada Linux, IRIX, dan solaris namun tidak bekerja pada BSD, yang mana hanya mengenali opsi ps BSD-style. Untuk mengatasi maslah ini, keychain mengecek apakah current sistem ps tersebut bekerja dengan sintaks BSD atau Sistem Vsebelum mengeksekusi pipeline ps :

Listing 9. Detecting BSD vs. System V

```
psopts="FAIL"
ps uxw >/dev/null 2>&1
if [ $? -eq 0 ]
then
psopts="uxw"
else
ps -u `whoami` -f >/dev/null 2>&1
if [ $? -eq 0 ]
then
psopts="-u `whoami` -f"
fi
fi
if [ "$psopts" = "FAIL" ]
then
echo $0: unable to use \"ps\" to scan
for ssh-agent processes.
Report KeyChain version and echo
system configuration to
drobbins@gentoo.org.
exit 1
fi

mypids=`ps $psopts 2>/dev/null | grep
"[s]sh-agent" | awk '{print $2}'` >
/dev/null 2>&1
```

Untuk memastikan bahwa kita bekerja dengan kedua sistem Sistem V dan BSD-style ps commans, script melakukan “dry run” dari ps uxw, throw away semua output. Jika kode error dari perintah tersebut zero maka kita tahu bahwa ps uxw bekerja dan kita set nilai psopts dengan sesuai.

Jika ps uxw mengembalikan nilai non-zero error code maka kita melakukan dryrun dari ps –u `whoami` -f, sekali lagi membuang semua output. Pada titik ini diharapkan ditemukan varian ps baik itu dari BSD atau Sistem V yang dapat kita gunakan. Jika tidak kita cetak error

dan kemudian keluar. Namun tampaknya salah satu jenis ps bekerja dimana kita eksekusi bari akhir pada kode snippet di atas. Dengan menggunakan variabel ekspansion $psopts langsug setelah ps, kita dapat meneruskan opsi yang benar untuk perintah ps tersebut.
Pipeline ps juga berisi true gep gem. Perhatikan bahwa grep –v grep bukan lagi bagian dari pipeline; melainkan, telah dibuang dan grep "ssh-agent" telah dirubah menjadi grep "[s] sh-agent".

Listing 10. Neat grep trick

```
mypids=`ps $psopts 2>/dev/null | grep
"[s]sh-agent" | awk '{print $2}'` >
/dev/null 2>&1
```

Ketika menggunakan grep "[s] sh-agent", Anda merubah bagaimana perintah grep muncul pada ps proses list. Dengan melakukan hal tersebut, Anda mencegah grep dari matching dirinya sendiri sebab string [s] sh-agent berbeda dengan expresi reguler [s] sh-agent.

Daftar Pustaka

[1]. Munir, Rinaldi (2006). Diktat Kuliah IF5054 Kriptografi Program Studi Teknik Informatika Institut Teknologi Bandung.
[2]. http://en.wikipedia.org/wiki/Ssh
[3]. http://www-128.ibm.com/developerworks/linux/library/l-keyc1/#resources
[4]. http://www-128.ibm.com/developerworks/linux/library/l-keyc2/#resources
[5]. http://www-128.ibm.com/developerworks/linux/library/l-keyc3/#resources
[6]. http://www.openssh.com/
[7]. http://www.gentoo.org/projects/keychain.html
[8]. http://www.socks.nec.com/protocol/socks4.protocol