MEMPERKECIL CELAH KEAMANAN DENGAN PEMPROGRAMAN JAVA

R. Rhoedy Setiawan

Jurusan Sistem Informasi, Teknik Univesitas Muria Kudus
Gondang Manis Po B0x 53, Bae Kudus
e-mail: rhoedy_05@yahoo.co.id

ABTRAKSI

Konten dieksekusi adalah ide mengirim sekitar data yang sebenarnya kode yang akan dieksekusi. Mengapa ide konten dieksekusi begitu menarik? Javabannya cukup sederhana. Kekuatan konten dan ekspresi penggunaan World Wide Web telah meledak selama beberapa tahun terakhir, seiring dengan pertumbuhan ini ada banyak usaha untuk memperbaiki lubang aplikasi untuk Web. Sementara Web telah diadaptasi untuk memungkinkan menggunakan lebih menarik melalui bentuk dan skrip yang berjalan di server, metode ini sangat membatasi. Kemampuan untuk memiliki user lokal menjalankan program yang ditulis dalam bahasa pemrograman penuh memungkinkan aplikasi untuk digunakan secara langsung melalui Web. Sampai Apple melihat cocok untuk memecahkan masalah ini, bagaimana Anda melindungi diri sendiri. Satu-satunya solusi yang nyata untuk saat ini adalah untuk menonaktifkan Java di browser Anda.

ABTRACTION

Executable content is the idea of sending some data the actual code to beexecuted. Why the idea of executable content so compelling? Answer the quite simple. The power of content and expression using the World Wide Web hasexploded over the last few years, in line with this growth there are many attempts to repair the hole for the Web application. While the Web has been adapted toallow use of more interesting through the forms and scripts running on the server this method is very limiting. The ability to have a local user to run programs written in a full programming language allows the application to be used directlyover the Web. Until Apple sees fit to solve this problem, how do you protect yourself. The only real solution for now is to disable Java in your browser.

PENDAHULUAN

Java Runtime Environment (JRE) dan *Java Development Kit*(JDE), terkait hal ini, sebagai platform ini meluncurkan *patch* terbaru untuk menambal beberapa celah keamanan yang ada. Dua celah keamanan pertama kali ditemukan di mesin virtual milik Java Runtime Environment yang memungkinkan sebuah *applet* atau aplikasi independen (yang ditulis dengan Java) melakukan penambahan komponen program

yang diambil langsung dari Internet. Hak akses yang melebihi ketentuan pada lubang keamanan ini bukan tidak mungkin dapat dimanfaatkan *cracker* untuk mengambil alih sistem tanpa sepengetahuan yang mempunyai web

Celah lainnya juga sempat ditemukan pada badan komponen dari Java Runtime Environment yang memproses transformasi XSLT. Akibat yang ditimbulkannya hampir sama dengan 2 celah di JRE. Cuma saja, yang ini memiliki efek tambahan yang memungkinkan *cracker* melakukan serangan Denial of Service (DoS) sehingga menyebabkan JRE menjadi crash.

METODE PENELITIAN

Dalam penelitian ini penulis menggunakan metode pemodelan *Spiral* (Pressman,1982), diantaranya meliputi:

1. **Studi Literatur**

   Studi literatur mengenai *semantic web*, RDF/OWL dan situs-situs web yang relevan, studi kasus pencarian pada beberapa situs-situs edukasi melalui *digital library* dan mempelajari jenis-jenis properti untuk selanjutnya menjadi pertimbangan dalam memperkecil celah keamanan berbasis java.

2. **Perencanaan (*Planning*)**

   Menggambarkan mengenai sumber daya (*resource*), alokasi waktu (*timeline*) yang diperlukan untuk menyelesaikan pembuatan sistem dan informasi-informasi lainnya terkait dengan pengembangan sistem

3. ***Risk Analisis***

   Pada bagian ini merupakan analisa terhadap kekurangan-kekurangan yang terjadi pada sistem dengan tujuan untuk memperoleh masukan dan perbaikan sistem kedepan

4. **Implementasi (*Engineering*)**

   Bangun dengan tool *protege_3.4*, kemudian aplikasi keamanan diimplementasikan mengunakan *Java Server Pages* dan Jena API 2.5.6.

5. ***Construction and Release***

   Pengujian dilakukan untuk validasi apakah sistem yang dibangun sudah berjalan sesuai yang diharapkan dan memberikan hasil yang relevan.

HASIL DAN PEMBAHASAN

Penelitian ini mengevaluasi isu-isu keamanan yang diangkat oleh bahasa Java dan menggunakan yang dimaksud dalam Java Web browser diaktifkan dan solusi yang diusulkan Java. Setelah diskusi singkat tentang latar belakang konten dieksekusi, makalah ini bergerak untuk mendiskusikan risiko keamanan potensial dari konten dieksekusi, apa solusi Java diusulkan, dan akhirnya analisis efektivitas mereka solusi. Tentang keamanan Java seseorang harus memiliki pemahaman tentang potensi masalah yang diangkat oleh konten dieksekusi. Keuntungan dari konten dieksekusi berasal dari peningkatan kekuatan dan fleksibilitas yang disediakan oleh program perangkat lunak. Peningkatan daya dari applet Java (istilah Java untuk konten dieksekusi) juga potensi masalah. Bila pengguna adalah berselancar di Web, mereka tidak perlu khawatir bahwa applet dapat menghapus file atau mengirimkan informasi pribadi.

Suatu bagian penting untuk menciptakan lingkungan yang aman untuk sebuah program untuk berjalan di adalah mengidentifikasi sumber daya dan kemudian memberikan beberapa jenis akses terbatas ke sumber daya. Tabel 1 menyediakan sebagian daftar sumber daya host khas bersama dengan klasifikasi dari beberapa jenis serangan yang dapat dikaitkan dengan ketersediaan sumber daya itu. Empat jenis serangan. Pengungkapan informasi tentang pengguna atau mesin host penolakan serangan layanan membuat sumber daya tersedia untuk tujuan yang sah (yaitu mengisi sistem file) merusak atau memodifikasi data, hal ini dapat mencakup data yang digunakan oleh program lain atau oleh sistem file gangguan serangan seperti menampilkan gambar cabul di layar pengguna. Perhatikan bahwa tabel tidak dimaksudkan untuk menjadi lengkap dalam hal kemungkinan jenis serangan, tetapi hanya memberikan contoh dari jenis masalah yang terkait dengan sumber daya yang diberikan. Sebagai contoh, untuk program spoofing (program yang muncul kepada pengguna untuk menjadi program yang berbeda) mungkin keinginan untuk memanfaatkan semua sumber daya yang diberikan untuk serangan sejak itu akan muncul kepada pengguna untuk menggunakan sumber daya yang sama seperti program asli.

Ketersediaan	Sumber daya	Pengungkapan	Integritas	Kerusakan
File sistem	x	x	x	x
Jaringan		x		x
Memori acak	x	x	x	x
Perangkat Keluaran (CRT, Speaker, dll)				x
Input Device (Keyboard, Microphone, dll)			x	x
Proses Kontrol		x		x
Pengguna Lingkungan		x	x	x
Sistem Panggilan	x	x	x	x

Tabel 1: Sumber Daya *host*

Beberapa sumber daya yang diberikan adalah jelas lebih berbahaya untuk memberikan akses penuh ke daripada yang lain. Sebagai contoh sulit untuk membayangkan kebijakan keamanan di mana program yang tidak diketahui harus diberikan akses penuh ke sistem file. Di sisi lain, kebijakan keamanan yang paling tidak akan membatasi program dari akses hampir penuh ke layar (dengan asumsi program ini terbatas dalam cara lain).

Java adalah bahasa berorientasi objek dengan sintaks mendekatkati dengan C++. Fitur penting dari bahasa dari sudut pandang keamanan adalah penggunaan kontrol akses untuk variabel-variabel dan metode dalam kelas, keamanan sistem tipe, kurangnya pointer sebagai tipe data bahasa, penggunaan pengumpulan sampah (dealokasi memori otomatis) , dan penggunaan paket dengan ruang nama yang berbeda.

Java, seperti C++, memiliki fasilitas untuk mengontrol akses ke variabel dan metode obyek. Kontrol ini memungkinkan akses objek yang akan digunakan oleh non-terpercaya kode dengan jaminan bahwa mereka tidak akan digunakan tidak semestinya. Sebagai contoh, perpustakaan Java berisi definisi untuk objek File. Obyek Berkas memiliki metode umum (callable oleh siapapun) untuk membaca dan metode tingkat rendah swasta (hanya callable dengan metode objek) untuk membaca. Panggilan dibaca publik pertama melakukan pemeriksaan keamanan dan kemudian panggilan

membaca pribadi. Bahasa Java memastikan bahwa kode non-terpercaya aman dapat memanipulasi objek File, hanya menyediakan akses ke metode publik. Dengan demikian, fasilitas kontrol akses memungkinkan programmer untuk menulis perpustakaan yang dijamin oleh bahasa yang akan aman dengan benar menentukan kontrol akses perpustakaan.

Sebuah fasilitas kedua untuk menyediakan kontrol akses adalah kemampuan untuk menyatakan kelas atau metode sebagai akhir. Hal ini menyediakan kemampuan untuk mencegah programmer berbahaya dari subclassing kelas perpustakaan kritis atau meng-override metode kelas. Jadi, bahasa menjamin bahwa metode aktual yang dipanggil pada suatu objek adalah metode selesai yang ditulis untuk jenis objek waktu kompilasi. Hal ini memberikan jaminan bahwa bagian-bagian tertentu dari perilaku obyek belum diubah.

Bahasa Java ini juga dirancang untuk menjadi bahasa jenis-aman. Ini berarti bahwa waktu kompilasi jenis dan tipe *runtime* dari variabel yang dijamin akan kompatibel. Hal ini memastikan bahwa *gips* (operasi yang memaksa tipe *runtime* untuk jenis kompilasi diberikan waktu) diperiksa baik di waktu kompilasi atau *runtime* untuk memastikan bahwa mereka sah. Hal ini untuk mencegah penempaan akses ke obyek untuk berkeliling kontrol akses.

Fitur lain keselamatan adalah penghapusan pointer sebagai tipe data. Ini berarti bahwa pointer tidak dapat secara langsung dimanipulasi oleh kode pengguna (tidak ada pointer arithmetic). Hal ini untuk mencegah penyalahgunaan baik berbahaya dan disengaja dari pointer (menjalankan dari ujung array misalnya). Sekali lagi menggunakan contoh file kita, hal ini mencegah kode berbahaya dari sekadar mengakses metode pribadi langsung dengan menggunakan pointer arithmetic dimulai dengan pointer objek File itu.Jelas jenis-keselamatan adalah bagian penting dari fasilitas kontrol akses objek, mencegah penempaan.

Bahasa Java menggunakan pengumpulan sampah untuk memulihkan memori yang tidak terpakai bukan mengandalkan dealokasi pengguna eksplisit. Ini tidak hanya menghilangkan kelas yang sangat umum dari bug, tapi menghilangkan lubang keamanan potensial. Sebagai contoh, jika Java dealokasi manual, ini bisa menyediakan cara berputar-putar secara ilegal casting. Pertama, kode berbahaya menciptakan objek

baru dari myfile jenis, dan kemudian deallocates memori yang digunakan oleh objek itu, menjaga pointer. Kemudian, kode berbahaya segera membuat objek File yang kebetulan memiliki ukuran yang sama. Akhirnya, bahasa Java menggunakan paket (mirip dengan modul dalam Modula-3, atau paket di *Common Lisp*) untuk menyediakan *enkapsulasi namespace*. Dari sudut pandang keamanan, paket berguna karena mereka memungkinkan kode *download* untuk dapat dengan mudah dibedakan dari kode lokal. Secara khusus, hal ini mencegah kode *download* dari kode membayangi sistem perpustakaan dengan kode jahat. Bahasa Java menjamin bahwa ketika kelas direferensikan sistem yang pertama terlihat dalam namespace lokal, dan kemudian di namespace dari kelas referensi. Ini juga menjamin bahwa kelas lokal tidak dapat sengaja referensi kelas download. getInCheck Tentukan apakah pemeriksaan keamanan sedang berlangsung checkCreateClassLoader Periksa untuk mencegah instalasi classloaders tambahan. Periksa checkAccess untuk melihat apakah thread atau kelompok benang dapat memodifikasi kelompok benang.

Urutan Metode *SecurityManager* :

1. CheckExit Cek jika perintah Exit dapat dieksekusi.
2. CheckExec Cek jika perintah sistem dapat dieksekusi.
3. Cek checkRead jika file dapat dibaca dari.
4. Cek checkWrite jika file dapat ditulis.
5. CheckConnect Cek jika koneksi jaringan dapat dibuat.
6. Cek checkListen jika port jaringan tertentu dapat mendengarkan untuk koneksi.
7. CheckAccept Cek apakah koneksi jaringan dapat diterima.
8. CheckProperties Cek jika sifat Sistem dapat diakses.
9. Cek apakah checkTopLevelWindow jendela harus memiliki peringatan khusus.
10. Cek checkPackageAccess jika paket-paket tertentu dapat diakses.
11. CheckPackageDefinition Cek jika kelas baru dapat ditambahkan ke paket.
12. CheckSetFactory Periksa apakah Applet yang dapat mengatur pabrik objek jaringan-terkait.

SecurityManager menyediakan mekanisme yang sangat fleksibel dan kuat untuk kondisional memungkinkan akses ke sumber daya. SecurityManager metode yang yang

memeriksa akses dilewatkan argumen yang diperlukan untuk menerapkan kebijakan akses bersyarat, serta memiliki kemampuan untuk memeriksa tumpukan eksekusi untuk menentukan apakah kode telah dipanggil oleh kode lokal atau di-download.

Metafora standar untuk menciptakan kode perpustakaan untuk sumber daya sistem yang berpotensi berbahaya adalah untuk hanya memberikan akses ke operasi yang tidak berbahaya, dan untuk membungkus pemeriksaan keamanan (melalui SecurityManager itu) sekitar panggilan bahwa akses diberikan kepada secara terbatas.

Contoh pemeriksaan keamanan.

```
public boolean mkdir (String path) throws IOException {
SecurityManager security = System.getSecurityManager ();
jika (keamanan = null) {
security.checkWrite (path);
}
kembali mkdir0 ();
}
```

Contoh pemeriksaan keamanan.

```
BEGIN
IMPORTING string lemmas; identifiers
ClassLoader : TYPE+
Class : DATATYPE
BEGIN
resolved(name : string; references : list[string]; loader : ClassLoader; linked :
list[Class]) :
resolved
unresolved(name : string; references : list[string]; loader : ClassLoader) : unresolved
END Class
ClassID : TYPE = Ident
ClassList : TYPE = list[Class]
ClassIDMap : TYPE = FUNCTION[ClassID ! Class]
```

```
ClassDB : TYPE = [ClassID;ClassIDMap]
ClassTable : TYPE = [list[[string; ClassLoader; list[ClassID]]];ClassDB]
Object : TYPE+ = [#cl : Class#]
primordialClassLoader : ClassLoader
mkClass((nm : string); (refs : list[string]); (ldr : ClassLoader)) :
Class = unresolved(nm; refs; ldr)
bogusClass : Class = mkClass( "" ; null; primordialClassLoader)
emptyClassTable : ClassTable = (null; (initialID; _ (id : ClassID) : bogusClass))
FindClassIDs((ct : ClassTable); (nm : string); (cldr : ClassLoader)) :
RECURSIVE list[ClassID] = CASES PROJ 1(ct) OF
null : null;
cons(hd; tl) :
LET tab = PROJ 1(ct); db = PROJ 2(ct)
IN IF
PROJ 1(hd) = nm^
PROJ 2(hd) =
cldr
THEN PROJ 3(hd)
ELSE
FindClassIDs((tl; db); nm; cldr)
ENDIF
ENDCASES
MEASURE length(PROJ 1(ct))
FindClass((ct : ClassTable); (nm : string); (cldr : ClassLoader)) :
ClassList = map(PROJ 2(PROJ 2(ct)); FindClassIDs(ct; nm; cldr))
InsertClass((ct : ClassTable); (nm : string); (cldr : ClassLoader); (cl : Class)) :
ClassTable =
LET old = FindClassIDs(ct;nm; cldr);
newID = GetNextID(PROJ 1(PROJ 2(ct)));
newMap = PROJ 2(PROJ 2(ct)) WITH [newID := cl]
8
```

```
IN (cons((nm; cldr; cons(newID; old)); PROJ 1(ct)); (newID; newMap));
ReplaceClass((ct : ClassTable); (cl; newCl : Class); (cldr : ClassLoader)) : ClassTable =
LET classDB = PROJ 2(PROJ 2(ct));
id = PROJ 1(PROJ 2(ct));
tab = PROJ 1(ct);
clID = FindClassIDs(ct; name(cl); cldr)
IN CASES clID OF cons(hd; tl) : (tab; (id; classDBWITH [hd := newCl])); null :
ctENDCASES
define((ct : ClassTable); (nm : string); (refs : list[string]); (cldr : ClassLoader)) :
[Class; ClassTable] = LET cl = mkClass(nm; refs; cldr) IN (cl; InsertClass(ct; nm; cldr;
cl))
findSysClass((ct : ClassTable); (nm : string)) :
ClassList = FindClass(ct; nm; primordialClassLoader)
foo : list[string] = cons( "foo" ; null)
Input : (cons?[string])
loadClass((ct : ClassTable); (nm : string); (cldr : ClassLoader)) : [Class;ClassTable] =
LET local = findSysClass(ct;nm); loaded = FindClass(ct; nm; cldr)
IN IF null?(local) THEN IF cons?(loaded) THEN (car(loaded); ct)
ELSE define(ct; nm; Input; cldr)
ENDIF
ELSE (car(local); ct)
ENDIF;
linkClass((ct : ClassTable); (cl : Class); (cldr : ClassLoader)) :
RECURSIVE [Class;ClassTable] = LET getClass = (_(n: string) : loadClass(ct; n; cldr))
IN CASES references(cl) OF
null :
IF unresolved?(cl)
THEN (resolved(name(cl); null; loader(cl); null);ct)
ELSE (cl; ct)
ENDIF;
cons(hd; tl) :
```

```
LET (res; newCt) = getClass(hd);
newCl = CASES cl OF
unresolved(name;
references;
loader) :
resolved(name; tl;
loader;
cons(res; null));
resolved(name;
references;
loader; linked) :
resolved(name; tl;
loader;
cons(res; linked))
ENDCASES
IN linkClass(newCt; newCl; cldr)
ENDCASES
MEASURE length(references(cl))
resolve((ct : ClassTable); (cl : Class); (cldr : ClassLoader)) : ClassTable =
LET (newCl; newCt) = linkClass(ct; cl; cldr) IN ReplaceClass(newCt; cl; newCl; cldr);
forName((ct : ClassTable); (nm : string); (cldr : ClassLoader)) : [Class;ClassTable] =
CASES FindClass(ct; nm; cldr) OF cons(hd; tl) : (hd; ct); null : loadClass(ct;nm; cldr)
ENDCASES
newInstance((clss : Class)) : Object = (#cl := clss#)
getClassLoader((cl : Class)) : ClassLoader = loader(cl)
getName((cl : Class)) : string = name(cl)
jlObjectClass : Class =
mkClass( "java.lang.Object" ; null; primordialClassLoader)
jlClassClass : Class = 9
mkClass( "java.lang.Class" ;
cons( "java.lang.Object" ; null); primordialClassLoader)
```

```
jlClassLoaderClass : Class =
mkClass( "java.lang.ClassLoader" ;
cons( "java.lang.Object" ;
cons( "java.lang.Class" ; null));
primordialClassLoader)
sysClassTable : ClassTable =
InsertClass(InsertClass(InsertClass(emptyClassTable;
"java.lang.Object" ;
primordialClassLoader;
jlObjectClass);
"java.lang.Class" ;
primordialClassLoader; jlClassClass);
"java.lang.ClassLoader" ;
primordialClassLoader; jlClassLoaderClass)
ct : VAR ClassTable
nm : VAR string
cldr : VAR ClassLoader
cl : VAR Class
MapPreservesLength : LEMMA
(8 (f : FUNCTION[ClassID ! Class]); (l : list[ClassID]) :
length(map(f; l)) = length(l))
proj1 FindClassIDs : LEMMA
(8 (ct : ClassTable); (nm : string); (cldr : ClassLoader); (classdb : ClassDB) :
FindClassIDs((PROJ 1(ct); classdb);nm; cldr) = FindClassIDs(ct; nm; cldr))
Add : CONJECTURE
(9 (cll : ClassList) :
FindClass(InsertClass(ct; nm; cldr; cl);nm; cldr) = cons(cl; cll))
Resolve : CONJECTURE
(8 (cl : Class); (ct : ClassTable); (cldr : ClassLoader) :
references(PROJ 1(linkClass(ct; cl; cldr))) = null)
Safe((ct : ClassTable)) : bool =
```

(8 (nm : string); (cldr : ClassLoader) :
LET cll = length(FindClass(ct; nm; cldr)) IN cll _ 1)
safe proj : LEMMA
(8 ct; (mapping : ClassIDMap) :
Safe(ct) _ Safe(PROJ 1(ct); (PROJ 1(PROJ 2(ct));mapping)))
forName inv : THEOREM (8 ct;nm; cldr : Safe(ct) _ Safe(PROJ 2(forName(ct; nm; cldr))))
Initial Safe : THEOREM Safe(sysClassTable)
loadClass inv : THEOREM
(8 ct; nm; cldr : Safe(ct) _ Safe(PROJ 2(loadClass(ct;nm; cldr))))
linkClass inv : THEOREM
(8 ct; cl; cldr : Safe(ct) _ Safe(PROJ 2(linkClass(ct; cl; cldr))))
resolve inv : THEOREM (8 ct; cl; cldr : Safe(ct) _ Safe(resolve(ct; cl; cldr)))
END Types

Contoh ini menunjukkan metafora dasar: mkdir metode umum memeriksa SecurityManager sistem (yang akan melemparkan eksepsi jika memeriksa tidak lulus) dan kemudian memanggil mkdir0 tingkat metode rendah pribadi(contoh1). Pada THEOREM akand dibadingkan secara acak dengan perhitungan berjenjang atau berkelas yang diwakili dengan kelaskelas tertentu bila yang dibandingkan cocok, maka akan berlanjut proses berikutnya(contoh 2)..

Web browser sendiri memainkan peran besar dalam keamanan sistem. Web browser mendefinisikan dan menerapkan kebijakan keamanan untuk menjalankan kode Java download. Web browser Java akan mencakup Java interpreter dan runtime perpustakaan bersama dengan kelas-kelas tambahan untuk menerapkan SecurityManager dan berbagai classloaders. Dari sudut pandang keamanan, pelaksanaan browser Web dari SecurityManager jauh lebih penting daripada pelaksanaan classloaders. Beberapa diskusi tentang bagaimana meningkatkan keamanan melalui *ClassLoader* ini dibahas dalam berikutnya.

SecurityManager mengontrol akses ke sumber daya sistem kritis. Hal ini memungkinkan penulis browser Web untuk menerapkan kebijakan keamanan tertentu

dengan *subclassing SecurityManager* dan override metode tertentu, dan kemudian menginstal versi baru sebagai *SecurityManager* sistem. Sejak *SecurityManager* subclassed menerapkan kebijakan keamanan, sangat penting bahwa versi browser Web dari *SecurityManager* diimplementasikan dengan benar. Dalam ekstrim, jika browser Web Java tidak menginstal SecurityManager sistem, applet akan memiliki akses yang sama sebagai aplikasi *Java* lokal.

Kebijakan keamanan *browser Web* dapat dibuat sewenang-wenang kompleks sejak SecurityManager kait menyediakan antarmuka yang fleksibel. Setiap kebijakan yang dapat diprogram dapat digunakan. Sebagai contoh, kebijakan dapat memiliki permintaan *SecurityManager* pengguna dengan informasi mengenai setiap akses diminta tertentu. Java keamanan model ini hampir seluruhnya didasarkan pada kemampuan untuk memverifikasi bytecode download, kemampuan untuk menentukan dan menulis perpustakaan yang mencegah akses yang tidak diinginkan ke sumber daya, dan kemampuan pengembang *Web browser* untuk menentukan dan menulis kode yang mengimplementasikan kebijakan keamanan yang baik.

Keputusan untuk mentransfer program *Java* dalam bentuk *bytecode* dikompilasi adalah keputusan dipertanyakan dari sudut pandang keamanan.Sistem ini mungkin akan lebih aman jika kode sumber *Java* yang digunakan, bukan *bytecode*. Penggunaan *bytecode* memerlukan proses verifikasi *bytecode* untuk memastikan bahwa *bytecode* tidak melanggar persyaratan dari bahasa Java. Jika kode sumber Java bukan digunakan, keamanan kode dapat dijamin dengan menginterpretasikan sumber langsung atau kompilasi kode *Java* secara lokal dengan kompilator terpercaya. Ini akan menjadi pendekatan yang lebih sederhana, karena kompiler Java sudah dipercaya (sistem runtime Java tidak melakukan verifikasi pada *bytecode* lokal).Penambahan program yang memeriksa apakah program-program dalam bahasa lain (*bytecode*) kompatibel menambahkan titik lain kegagalan untuk sistem.

Ada beberapa kemungkinan alasan untuk menggunakan *bytecode*. Pertama, ia menyediakan kebingungan tertentu, mencegah reverse *engineering* dari program Java. Sementara ini memang benar, bytecode tidak mungkin untuk mendekompilasi, dan ada teknik kebingungan lain yang memanipulasi kode sumber secara langsung. Kedua, representasi *bytecode* mungkin agak kecil.Saat ini, hal ini tidak

benar. Dalam setiap contoh saat ini, termasuk source browser HotJava seluruh sumber dan *bytecode* hampir identik dalam ukuran. Ketiga, proses verifikasi *bytecode* mungkin lebih cepat dari proses kompilasi. Dengan demikian, proses download dan menjalankan program-program komputasi secara intensif kurang dan memberikan kinerja rendah. Argumen ini cukup menarik, meskipun tim Java telah menyarankan bahwa mereka akhirnya mungkin menggunakan Hanya pada saat compiler untuk program-program komputasi secara intensif, menunjukkan kompilasi tidak mahal. Argumen terakhir dapat dibuat bahwa *bytecode* verifier adalah program kurang kompleks daripada sebuah kompiler penuh, sehingga verifikasi kebenaran adalah proses sederhana. Bahkan, compiler tidak harus dipercaya karena semua *bytecode* dapat diverifikasi dengan bytecode verifier terpercaya (kode lokal dapat diverifikasi hanya sekali ketika dikompilasi untuk tujuan efisiensi).

Skenario

Setiap serangan integritas disebutkan dengan mudah dapat dicegah dengan kemampuan akses kontrol. Modifikasi berbahaya dari file, memori, dan benang dapat dicegah. Ketersediaan serangan ketersediaan adalah jauh lebih sulit untuk mencegah. Seperti disebutkan sebelumnya, tidak ada batasan saat ini untuk mencegah alokasi semua memori yang tersedia ke Java atau penciptaan ribuan jendela. Java tidak memiliki kemampuan untuk menempatkan kontrol beberapa pada penciptaan benang prioritas tinggi.

Pengungkapan setiap serangan pengungkapan disebutkan dengan mudah dapat dicegah dengan kemampuan akses kontrol. Java menyediakan mekanisme yang baik untuk mencegah applet dari mengakses informasi sensitif, serta mencegah pembentukan saluran untuk mengirimkan data. Karena salah satu dari ini akan cukup untuk menghentikan serangan pengungkapan, kombinasi cukup. Sejak grafis dan audio saat ini tidak mungkin untuk layar berdasarkan konten, serangan tidak dapat dicegah tanpa mengambil posisi ekstrem yang tidak ada data yang didownload akan ditampilkan atau didengar. Java menyediakan alternatif tertentu ini (jangan menggunakannya untuk men-download sesuatu), tetapi tidak memberikan sesuatu yang lebih flekibel. Analisis yang diberikan menunjukkan bahwa Java efektif untuk mencegah jenis serangan lebih

berbahaya. Perlu dicatat bahwa serangan yang disebutkan hanya sebagaimana berlaku bagi browser Web saat ini yang tidak menggunakan Java. Masalah penolakan serangan layanan juga cukup sulit untuk mencegah sepenuhnya. Satu bisa membayangkan kebijakan keamanan yang mencegah penciptaan lebih dari 10 jendela, atau mencegah penggunaan lebih dari 100*Kbytes* memori, tetapi ini jenis pembatasan tampak sewenang-wenang. Sebaliknya, akan diinginkan untuk memiliki *Web browser* yang memungkinkan pengguna untuk secara eksplisit membunuh applet dan semua sumber daya yang menggunakan. Semoga mekanisme tersebut akan dilaksanakan. Penggunaan tanda tangan digital sebagai mekanisme untuk memverifikasi kode yang berasal dari sumber yang terpercaya dapat memainkan bagian penting dalam masa depan keamanan Java. Saat ini tidak ada built in mekanisme untuk memungkinkan kode yang diverifikasi telah datang dari sumber terpercaya memiliki akses khusus ke sumber daya. Mekanisme keamanan Java saat ini tampaknya cukup fleksibel untuk memungkinkan penambahan applet digital ditandatangani. Kelas *ClassLoader* dapat *subtyped* untuk membuat *SignedClassLoader* yang pertama melakukan verifikasi tanda tangan digital, dan kemudian melakukan pemuatan sebenarnya dari kelas.Berbagai metode *SecurityManager* kemudian dapat memeriksa apakah panggilan tersebut dalam lingkup dinamis *SignedClassLoader* dalam rangka untuk menentukan apakah akses seharusnya diizinkan atau ditolak. Dengan demikian, mekanisme saat ini tentu memungkinkan para penulis browser Web untuk menambahkan akses khusus untuk kode digital ditandatangani.

KESIMPULAN

Beberapa pembahasan tentang keamanan Java perlu dimasukkan ke dalam perspektif. Alasan bahwa Java diinginkan di tempat pertama adalah bahwa itu memberikan daya yang meningkat dan fleksibilitas. Ada *trade off* antara ini tak terelakkan kenaikan daya dan risiko keamanan sistem menggunakan Java. Langkah-langkah keamanan di Java memberikan kemampuan untuk memiringkan keseimbangan ini dengan cara apapun adalah lebih baik.

DAFTAR PUSTAKA

Nathaniel S. Borenstein, *Email With a Mind of Its Own: The Safe-Tcl Language for Enabled Mail*. In *Proceedings of ULPAA* (1994).

Drew Dean and Dan S. Wallach, *Security Flaws in the HotJava Web Browser*, November 3, 1995. Available via ftp://ftp.cs.princeton.edu/reports/1995/501.ps.Z

General Magic, Inc. Available via *An Introduction to Safety and Security in Telescript*. Available via http://cnn.genmagic.com/Telescript/TDE/security.html

James Gosling and Henry McGilton, *The Java Language Environment: A White Paper*, Sun Microsystems, May 1995. Available via ftp://java.sun.com/docs/JavaBook.ps.tar.Z

Sun Microsystems, *HotJava(tm): The Security Story*. Available via http://java.sun.com/1.0alpha3/doc/security/security.html

Sun Microsystems, *The Java Language Specifications: Version 1.0 Beta*. Available via http://java.sun.com/JDK-beta/psfiles/javaspec.ps

Frank Yellin, *Low Level Security in Java*. Available via http://www.w3.org/pub/Conferences/WWW4/Papers/197/40.htm