

**KEAMANAN SISTEM INFORMASI (EC-5010)**

**Makalah Proyek Akhir**

**Medusa DS9 dan Penggunaannya**



Tanggal : 16 Mei 2004

Oleh

Hankky Arief (13200035)

**DEPARTEMEN TEKNIK ELEKTRO  
INSTITUT TEKNOLOGI BANDUNG**

**2004**

## ABSTRAKSI

Makalah ini membahas sebuah mekanisme *access control* bernama Medusa DS9 (versi 0.9-2), yang dapat berjalan pada sistem operasi Linux. Medusa mampu mengamati berbagai *process* (program) yang terjadi pada sistem. Melalui sebuah file konfigurasinya, dapat dibuat berbagai aturan terhadap hak akses suatu *process* yang didasarkan pada operasi-operasi yang sedang dilakukannya. Operasi-operasi tersebut, misalnya operasi pengaksesan file, pemanggilan *syscall* dan lainnya.

Pembahasan ini terdiri dari pembahasan konsep *virtual spaces* dan server otorisasi yang menjadi dasar mekanisme ini. Pembahasan dilanjutkan dengan penjelasan singkat mengenai perubahan/*patch* kernel dan server otorisasi, kemampuan-kemampuan yang ditawarkannya, tahapan pada proses otorisasi, dan cara mengkonfigurasi Medusa. Pada bagian akhir terdapat dua buah percobaan menggunakan Medusa. Percobaan pertama berupa pembatasan (*restriction*) terhadap *network-process*, yaitu proses yang melakukan pemanggilan *socket call*. Percobaan kedua akan membuat *user* root tidak dapat mengakses folder “/home”. Pada bagian lampiran akan diberikan dokumen asli yang disertakan pada paket Medusa DS9, mengenai cara mengkonfigurasi Medusa.

## DAFTAR ISI

<b>ABSTRAKSI</b>	i
<b>DAFTAR ISI</b>	ii
<b>BAB I PENDAHULUAN</b>	<b>1</b>
1.1 Latar belakang dan Rumusan Masalah.....	1
1.2 Tujuan.....	1
1.3 Pembatasan Masalah.....	1
1.4 Sistematika Penulisan.....	2
<b>BAB II Medusa DS9</b>	<b>3</b>
2.1 Konsep Dasar dan Pengantar Medusa.....	3
2.2 Paket Perangkat Lunak Medusa.....	4
2.3 Kemampuan Perangkat Lunak Medusa.....	5
2.4 Tahap - tahap Otorisasi Medusa.....	6
2.5 Cara mengkonfigurasi Medusa.....	7
<b>BAB III PERCOBAAN PENGGUNAAN MEDUSA</b>	<b>12</b>
3.1 Percobaan 1 : Membatasi hak akses <i>network process</i> .....	13
3.2 Percobaan 2 : Membuat root tidak dapat mengakses folder “/home”.....	18
3.3 Permasalahan dalam Percobaan.....	21
<b>BAB IV PENUTUP</b>	<b>22</b>
KESIMPULAN.....	22
<b>REFERENSI</b>	<b>23</b>
<b>LAMPIRAN</b>	<b>24</b>

# BAB I

## PENDAHULUAN

### 1.1 Latar Belakang dan Rumusan Masalah

Salah satu aspek keamanan sistem komputer adalah mekanisme kontrol akses (*access control*). Melalui mekanisme ini, pengguna-pengguna pada sistem memiliki batasan dalam mengakses sumber daya yang ada, contohnya: pengguna biasa (*non-root*) dibatasi dalam pengendalian komputer, pengaksesan file yang bukan miliknya, dan sebagainya. Mekanisme ini hanya menyisakan seorang pengguna, yaitu *root* atau administrator untuk dapat mengakses sistem secara penuh.

Kontrol akses pada sistem UNIX (misalnya), masih memungkinkan para *hacker* melakukan pembajakan terhadap sistem. Cara yang dilakukan bermacam-macam, seperti: melalui *buffer overflow*, penembakan/pencurian pasangan nama (*userid*) dan password, pengeksploitasian program, dan sebagainya, sehingga didapat hak akses sebagai *root*<sup>1</sup>. Jika akses sebagai *root* telah berhasil didapat (oleh *hacker*), maka mekanisme kontrol akses tidak dapat menghentikan segala tindakan oleh *hacker* tersebut.

Rumusan masalah penelitian ini adalah bagaimana bentuk/model kontrol akses yang lebih baik, sehingga memiliki pertahanan yang lebih baik dan tidak bergantung pada *root*.

### 1.2 Tujuan

Tujuan penelitian ini adalah mempelajari sebuah mekanisme kontrol akses bernama Medusa DS9 untuk memperkuat pertahanan sistem komputer terhadap serangan (*system hacking*).

### 1.3 Pembatasan Masalah

---

<sup>1</sup> Lebih jauh dapat dilihat di "Hacking Exposed", McIure, Scamvrey, Kurtz. Edisi 3

Masalah dibatasi pada pembahasan Medusa DS9 : teori dasar, penggunaan, percobaan, tanpa perbandingan dengan model kontrol akses atau hal serupa lainnya. Kemudian akan dilakukan percobaan penggunaan Medusa.

#### **1.4 Sistematika Penulisan**

Bab pertama berisi latar belakang, rumusan masalah, tujuan penelitian, dan pembatasan masalah. Pada bab kedua berisi konsep dasar Medusa, bagian-bagian paket Medusa, *feature-feature*, dan cara menggunakannya. Bab ketiga berisi dua buah percobaan dan hasilnya.

## BAB II MEDUSA DS 9

### 2.1 Konsep Dasar dan Pengantar Medusa

Medusa adalah salah satu sistem *security* yang bekerja dari level kernel. Hal yang menjadi perhatiannya adalah mekanisme kontrol akses. Mekanisme yang diterapkannya merupakan implementasi dari ZP Security Framework, yaitu sebuah kerangka untuk sistem kontrol akses yang menggunakan konsep *Virtual Spaces* dan sebuah server otorisasi – sebagai pusat pemberi keputusan. Medusa merupakan projek *open source*, dengan homepage <http://medusa.fornax.sk>.

Mekanisme kontrol akses adalah mekanisme yang mengatur perizinan pengaksesan suatu *resource* (misalnya sebuah file). Pada sebuah sistem kontrol akses, terdapat tiga hal berikut: proses (*process*)<sup>2</sup> sebagai subjek, file sebagai objek<sup>3</sup>, dan aturan-aturan subjek terhadap objek.

Pada konsep *virtual spaces*, objek-objek dan proses-proses akan ditempatkan ke sejumlah *domain* (terbatas dan diskrit) yang disebut *virtual spaces*. Kemampuan suatu proses (untuk setiap tipe akses) direpresentasikan dengan himpunan (set) *virtual spaces*. Jika pada himpunan tersebut terdapat *virtual space* objek yang akan diakses, maka akses akan diberikan. Proses memiliki tiga buah himpunan *virtual spaces* untuk tiga buah jenis tipe akses, yaitu *read*, *write*, *see*.

Berbeda dengan metoda kontrol akses lainnya, Medusa memiliki sebuah program *daemon* di *user space* yang berfungsi sebagai pusat otorisasi, namanya “Constable”. Constable akan menginisialisasi dan meng-*update* berbagai variabel (seperti *virtual spaces*) atas dasar operasi-operasi yang terjadi di sistem. Setiap operasi “penting” dapat dikonfirmasi ke Constable untuk dikontrol dan dimonitor. Constable akan menentukan apakah suatu jenis operasi boleh berjalan atau tidak berdasarkan kondisi

---

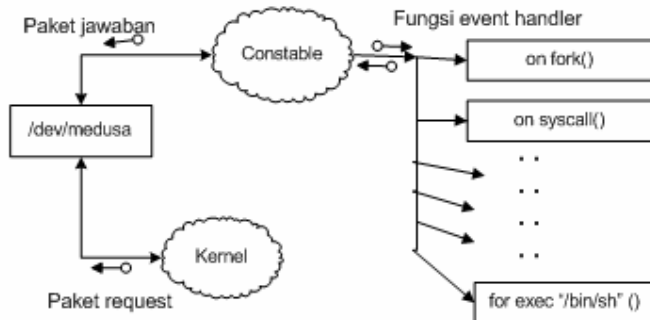
<sup>2</sup> *Process is a running instance of a program* (dari buku “Advanced Linux Programming”)

<sup>3</sup> Objek adalah sumberdaya yang diakses, sebuah proses juga dapat menjadi objek

saat itu. Melalui Constable, kita dapat membuat berbagai fungsi *handler* untuk setiap operasi yang ingin diperhatikan.

Lebih jauh mengenai ZP *security framework* dapat dilihat di [ZP].

## 2.2 Paket Perangkat Lunak Medusa



Gambar 2.2-1 Bagan perangkat lunak Medusa

Paket Medusa DS9 dapat di-*download* di <http://medusa.fornax.sk>. Penulis menggunakan versi 0.9-2.

Perangkat lunak Medusa terdiri dari : sebuah patch kernel (tersedia untuk kernel versi 2.4.18) dan program bernama Constable.

### a. Patch Kernel :

Patch kernel untuk mengimplementasikan Medusa berukuran kecil dan hanya melakukan sedikit perubahan pada bagian kernel yang asli. Patch terdiri dari tiga lapisan:

#### 1. Lapisan 1 (Penambahan kode pada kernel yang asli)

Lapisan ini akan melakukan pemanggilan fungsi-fungsi pada lapisan dua (`#include <linux/medusa.h>`) dan menginterpretasikan nilai keluarannya.

#### 2. Lapisan 2 (/medusa)

Berisikan implementasi dari fungsi-fungsi yang dipanggil lapisan satu. Fungsi-fungsi harus dijalankan di kernel space. Terdapat tiga hal yang dilakukan oleh fungsi-fungsi ini, antara lain :

- Pengecekan perizinan akses ( melihat virtual spaces, dan sebagainya).
- Pengecekan informasi konfirmasi (apakah suatu tipe akses suatu proses memerlukan konfirmasi Server otorisasi).
- Memanggil lapisan komunikasi, bila diperlukan.

#### 3. Lapisan komunikasi (medusa/comm.c)

Berfungsi untuk berkomunikasi dengan Constable yang berada pada *user-space*.

b. Constable :

Constable berjalan sebagai *daemon* pada *user space*. Program ini akan menerima paket-paket dari kernel yang berisi informasi mengenai suatu operasi yang akan dilakukan proses terhadap objek. Constable akan memanggil suatu fungsi (fungsi *handler*) untuk memproses paket tersebut. Terdapat sebuah fungsi untuk suatu tipe operasi, misal fungsi "for unlink [nama\_file] " akan dipanggil jika ada proses yang ingin menghapus file tersebut. Constable juga akan menginisialisasikan keanggotaan objek-objek pada *virtual spaces* dan variabel kontrol akses lainnya. Constable memiliki sebuah file konfigurasi yang dapat diisi dengan pendefinisian fungsi-fungsi *handler* dan nilai-nilai inisialisasi variabel-variabel.

Komunikasi antara bagian kernel dan Constabel dilakukan melalui sebuah file *device* (char mayor 111, minor 0). Paket yang dipertukarkan oleh keduanya, memiliki format tertentu, *field-field* dari paket (medusa\_packet) didefinisikan pada "/Constable/medusa.h".

Catatan :

- Menurut [XX], Medusa kemungkinan tidak akan menjadi sebuah sistem kontrol akses utama pada versi resmi kernel Linux, tetapi Medusa tetap dapat menjadi sebuah *add on* pada kernel tersebut.

### 2.3 Kemampuan Perangkat Lunak Medusa

Berikut ini kemampuan yang ditawarkan Medusa DS9, dikutip langsung dari dokumentasinya pada [Doc].

---

The current implementation of Medusa in the kernel, in cooperation with the security daemon in user space, has these features:

---

- o full access control to any file in the system (via VFS)
- o ability to redirect access to the selected file or to another one
- o complete control of signal sending/receiving
- o direct control of important process actions
- o control of execution of any *syscalls* for specified processes
- o every process or file is a member of a specified virtual subsystems and every process has assigned access rights to VS. (It is possible to completely hide processes or files from other processes this way.)
- o every process has a login uid (luid), which is set only on the first call of set{re,s}uid

- o ability to force execution of specified code for any process (similar to signal handling, see doc/Code\_forcing for details)
- o low level control of any system call

Dijelaskan pula bahwa : *low level control of any system call* dan *ability to force execution* hanya dapat berjalan di arsitektur 1386.

Berikut ini penjelasan lebih lanjut mengenai *point-point* tersebut :

- Penggunaan *virtual spaces* sebagai fasilitas kontrol akses bagi file-file pada file sistem.
- Pengontrolan langsung terhadap aktivitas proses-proses yang penting, hal ini akan berupa sebuah fungsi *handler* yang akan melakukan aksi dan otorisasi pada proses-proses tersebut.
- Dalam fungsi *handler* tersebut, selain terdapat “jawaban perizinan” untuk proses-proses, juga dapat ditambahkan perintah-perintah seperti *redirect*, *force*.
- Fungsi-fungsi tersebut akan dibangkitkan oleh pemanggilan *syscall* atau sinyal-sinyal tertentu (contohnya sinyal kill)
- Setiap proses memiliki variabel *luid* (*login user id*) dan sebetulnya masih banyak lagi (lihat lampiran).
- Perintah “force” digunakan untuk memindahkan pengekseskuan suatu proses ke program/kode yang kita buat. Hal ini dilakukan melalui pembuatan objek file dengan format ELF<sup>4</sup>. Pada [Doc] dikatakan bahwa *feature* ini memanfaatkan *stacks of user-space processes*, penulis tidak tahu apakah ini berarti justru memanfaatkan *stack* yang *executable* (hal yang justru berlawanan dengan kernel *patch* lainnya, seperti OpenWall<sup>5</sup>)

## 2.4 Tahap-tahap Otorisasi Medusa

Proses otorisasi pada Medusa akan melalui tahap-tahap yang secara terurut sebagai berikut [Doc]:

### 1) Pengecekan *Virtual Spaces*

Fase ini mengecek setiap proses yang akan mengakses objek-objek, apakah pada himpunan *virtual spaces* proses, untuk tipe akses yang akan dilakukan, setidaknya terdapat sebuah *virtual space* tempat objek berada, bila ya, lanjut ke fase dua, dan bila tidak akses tidak diberikan. Fase ini, bersama dengan fase dua dilakukan di bagian kernel.

### 2) *Confirmation status check*

---

<sup>4</sup> Lihat manual di Linux : `man ld`

<sup>5</sup> <http://www.openwall.com>

Pada fase ini dilakukan pengecekan apakah tipe akses oleh proses ini memerlukan persetujuan dari Constable, bila ya - lanjut ke fase tiga, bila tidak - lanjut ke fase empat.

3) Constable (bekerja di *user space* sebagai sebuah *daemon*)

Kernel akan mengirimkan paket yang menjelaskan operasi yang akan dilakukan ke security daemon (Constable) dan berbagai informasi yang diperlukan, kemudian menunggu jawabannya. Constable akan memproses paket tersebut pada fungsi handlernya

Bila jawaban :

- YES : Lanjut ke fase lima
- NO : Operasi dihentikan
- SKIP : ke fase enam
- ERR : ke fase empat (Constable tidak tahu jawabannya)
- OK : Ke fase empat

Constable juga dapat mengubah parameter-parameter proses dan objek (file), seperti mengubah kemampuan akses proses (mengubah set *virtual spaces* proses), menandai proses (flag), memonitor pemanggilan *system call* pada proses, dan sebagainya. Setelah selesai, Constable akan mengirimkan paket jawaban ke kernel. Aktifitas-aktifitas Constable ini akan menghasilkan sistem kontrol akses yang bersifat dinamis.

4) Pengecekan Sistem Permissi

Ini adalah pengecekan standar yang dimiliki oleh Linux, berdasarkan user ID, proses ID, dan matrik akses.

5) Pengeksekusian operasi

6) Return

Dari uraian di atas terlihat bahwa proses kontrol akses yang diberikan oleh Medusa DS9 bersifat bertingkat-tingkat. Kita dapat memberikan *virtual space* tertentu pada sebuah file sehingga hanya proses tertentu yang dapat mengaksesnya. Kita juga dapat memberikan batasan (*restriction*) lebih lanjut pada fungsi-fungsi handler Constable.

## 2.5 Cara mengkonfigurasi Medusa

File konfigurasi Constable merupakan sebuah bahasa pemrograman yang diinterpretasikan. Sintaksis program tersebut mirip dengan C, dan terdapat berbagai variabel *built-in*. Isi dari file konfigurasi ini, antara lain:

- Inisialisasi variabel-variabel bitmap *virtual spaces*, yang akan memetakan objek-objek dan kemampuan akses subjek-subjek.
- Fungsi-fungsi event handler

Fungsi *event handler* dipanggil ketika berbagai aksi (*process action* dan *filesistem action*) terjadi. Fungsi akan menerima paket-paket dari kernel, yang berisi berbagai informasi mengenai proses dan objeknya, field-field pada paket tersebut diberikan dapat dilihat pada file *header* medusa.h. Melalui pendefinisian fungsi-fungsi tersebut, kita dapat memberikan aturan-aturan kontrol akses pada berbagai proses.

#### Variabel *built-in*:

Pada file konfigurasi ini terdapat berbagai variabel-variabel *built-in*, yang akan memberikan berbagai informasi (mengenai identitas proses dan file), variabel ini antara lain: pid, uid, euid, suid, fsuid, gid, egid, sgid, fsgid, luid (uid saat login), vs, vss, vsr, vsw (variabel-variabel bitmap *virtual spaces*), flags, dan sebagainya. Untuk lengkapnya, penulis berikan pada Lampiran yang juga berisi pendefinisian sintaksis.

### 2.5.1 Cara Menginisialisasi

Berikut ini contoh skrip penginisialisasian:

```
// Set file sistem VS
recursive for set "/"
    vs=0b001;
recursive for set "/bin"
    vs=0b010;
recursive for set "/usr/bin"
    vs=0b010;
recursive for set "/home"
    vs=0b100;

/* set VS proses */
for exec "/bin/login" {
    vs = 0b100;          //ini adalah user process
    vsr= 0b111;        //dapat membaca semua file dan
    proses
    vsw= 0b100;        //hanya dapat menulis "/home"
}
```

Berikut ini penjelasan pada proses inisialisasi :

- vs adalah variabel bitmap, setiap bit '1' pada variabel ini menunjukkan keanggotaan suatu objek pada sebuah *virtual spaces*.
- recursive for set "/home" vs = 0b100, berarti : untuk folder "/home" dan file-file di dalamnya akan berada pada *virtual spaces* nomor tiga.
- Setiap proses/subjek memiliki tiga buah variabel bitmap, yaitu :

vss : bitmap *virtual spaces* yang anggotanya dapat melihat dan dilihat “si proses”

vsr : bitmap *virtual spaces* yang anggotanya dapat dibaca oleh proses ini

vsw : bitmap *virtual spaces* yang anggotanya dapat ditulisi oleh proses ini

Penginisialisasian proses dilakukan saat proses tersebut melakukan suatu aksi yang kemudian akan membangkitkan event ke Constable. Proses anaknya (fork) akan mewarisi variabel-variabel yang dimiliki oleh proses tersebut.

- Fungsi “For exec “/bin/login” “, adalah salah satu *event handler*, akan dipanggil ketika terjadi pengekseskuan file yang bernama login pada direktori /bin. Ini adalah salah satu fungsi untuk pengendalian operasi terhadap file.

### 2.5.2 Pembuatan fungsi-fungsi handler

Terdapat dua buah jenis fungsi handler, yaitu : process action handler dan filesistem handler, yaitu :

#### 1. Process action handler

Bentuk umumnya :

on <handler\_type> <command\_block>

Contoh :

```
on kill {
    if (vs ?! 0b001)
        if (target_pid == constable_pid or target_pid == 1)
            answer = MED_NOT;
}
```

Keterangan :

- “on kill“ akan dipanggil ketika suatu proses mengirim sinyal kill ke proses lain.
- (vs ?! 0b001) akan bernilai benar jika *virtual spaces* proses, yang kirim sinyal kill, tidak ‘beririsan’ dengan nilai 0b001. Pada konfigurasi ini virtual spaces 0b001 hanya ditempati oleh proses-proses lokal (misalnya tidak menggunakan socket tipe PF\_INET) .
- Fungsi ini akan menolak operasi yang akan mematikan proses Constable atau proses init, jika proses tersebut tidak lokal.

Terdapat beberapa *process handler\_type* , antara lain:

- init : dipanggil saat inisialisasi Constable (*startup*), proses yang melakukan operasi ini adalah Constable itu sendiri.
- fork : proses memanggil *system call* fork (pembuatan proses)

- `exec` : proses mengeksekusi program, variabel `info1 = argv`, `info2 = envp`
- `sexec` : proses mengeksekusi program setuid, `info1 = vs inode`
- `setuid` : proses memanggil `sys. call` `setuid`, `setreuid`, `setresuid`, `info1 = vs inode`
- `kill` : proses mengirim sinyal, *system call* `kill`, `info1 = nomor sinyal`
- `ptrace` : melakukan operasi `ptrace` pada proses
- `capable` : pengecekan kapabilitas proses, `info1 = mask kapabilitas`
- `syscall`: proses melakukan pemanggilan *system call* yang ingin kita trace, misal : kita ingin cek bila proses memanggil *socketcall* (nomor `syscall` : 102), untuk *men-trace*, "`trace_on 102`", maka setiap kali *socketcall* dipanggil akan dibangkitkan event → fungsi "`on syscall`" akan dipanggil.

## 2. Filesystem handler

Bentuk umumnya :

for `<handler_type>` `<filename>` `<command_block>`

Contoh :

```
for access "/home/hankky" {
    if (luid != 501)
        answer = NO;
}
```

Keterangan :

- fungsi ini akan memastikan hanya *user* yang login sebagai *user* dengan `uid = 501` yang dapat mengakses folder "`/home/hankky`".
- bila "`for access "/home/hankky`" " tidak diberikan, maka proses tersebut tidak akan mengetahui keberadaan folder tersebut

Terdapat beberapa tipe filesystem *handler*, antara lain:

- `set` : pada fungsi ini kita dapat menginisialisasikan nilai `bitmap` vs suatu `inode`
- `access` : dipanggil saat ada pengaksesan terhadap `inode`
- `create` : dipanggil saat pembuatan file
- `link` : dipanggil saat pembuatan *hardlink* ke file
- `unlink` : dipanggil saat penghapusan file
- `symlink` : dipanggil saat membuat link simbolik
- `mkdir` : dipanggil saat pembuatan direktori
- `rmdir`
- `mknod` : saat membuat file device
- `rename` : penamaan file
- `truncate`
- `permission`

- `exec` : pengeksekusian file

Hal-hal yang dapat dilakukan pada fungsi handler adalah sebagai berikut:

1. Pengecekan variabel-variabel keadaan, yaitu : `uid`, `pid`, `luid`, `eid`, `suid`, `fsuid`, `gid`, `egid`, `sgid`, `fsgid`, `vs`, `vss`, `vsr`, `vsw`, `flags`, `procact`, `fsact`, `ecap`, `icap`, `pcap`, `fcap`, `acap`, dan parameter-parameter *syscall* yang kita *trace*. (lihat lampiran)
2. Penginisialisasian dan peng-*update*-an nilai-nilai variabel keadaan (dari variabel `vs` sampai parameter `syscall` – lihat nomor 1).
3. Perintah untuk men-*trace* suatu *system call*.
4. Pemberian jawaban (*answer*) dan *apply* (*apply* perubahan terhadap proses ini saja atau juga pada proses parent-nya)
5. *Redirection*, yaitu mengubah objek file yang akan dikenai operasi
6. *Forcing*, yaitu menjalankan sembarang kode program yang kita tentukan, layaknya merupakan bagian dari proses tersebut.

### BAB III

#### PERCOBAAN PENGGUNAAN MEDUSA DS9

Percobaan ini berupa pembuatan file konfigurasi Constable, sehingga dihasilkan hal yang diinginkan. File konfigurasi ini sebagian diambil dari file yang disertakan pada paket Medusa dan sebagian lagi penulis buat sendiri. Terdapat dua buah percobaan yang analisisnya dilakukan mulai dari penjelasan file konfigurasi sampai dengan hasil dari file tersebut. Sewaktu penulis mencobakan Medusa ini, terdapat berbagai hal yang nampaknya tidak berjalan semestinya.

Beberapa hal yang masih tidak dapat berjalan dengan baik pada paket Medusa DS9 versi 0.9-2 (yang penulis temukan) :

1. Pen-trace-an socketcall tidak dapat memberikan nilai domain protokol yang dipakai oleh socket tersebut.
2. *Feature* “force code” tidak dapat bekerja, dideskripsikan lebih lanjut pada bagian 3.3: “Permasalahan dalam percobaan”.

Sebagai catatan penting untuk hal di atas, Medusa versi terbaru (1.0) memiliki program Constable yang sangat berbeda dengan versi 0.9-2. Oleh karenanya, mungkin hal-hal di atas telah diperbaiki.

Beberapa hal yang ingin dilakukan pada percobaan ini, antara lain:

1. Bagaimana membatasi proses-proses jaringan terhadap pengaksesan sistem
2. Bagaimana agar root tidak dapat melihat file pada direktori “home” milik pengguna biasa.

Catatan :

Perlu penulis sampaikan, bahwa penulis tidak berpikir untuk membuat sebuah solusi yang baik<sup>6</sup>, melainkan hanya untuk mencoba menunjukkan beberapa hal yang dapat dilakukan dengan Medusa.

---

<sup>6</sup> mutlak menyelesaikan masalah

### 3.1 Percobaan 1 : Membatasi hak akses *network process*

#### Tujuan :

Agar para hacker yang melakukan penyerangan tidak dapat berbuat banyak pada sistem, walaupun ia telah menjadi "root"

#### Metoda :

- Mendeteksi proses yang melakukan pemanggilan socket dengan domain internet networking (AF\_INET, AF\_UNIX) → akan diganti cara lain
- Membatasi kemampuan akses proses tersebut terhadap file-file, melalui penempatan proses dan file pada *virtual spaces* yang berbeda dan penggunaan perintah "*redirect*"
- Menghentikan operasi tertentu (dari proses tersebut), seperti pengiriman sinyal "Kill" terhadap proses "init" dan Constable

#### File konfigurasi :

1. Bagian inisialisasi *virtual spaces* file
2. Bagian pendeteksian dan pembatasan kemampuan proses jaringan
3. Fungsi-fungsi handler lainnya

#### 1. Listing bagian Inisialisasi *virtual spaces*:

```
recursive for set "/" //network sees, local has it
vs=0b0000000000000011;
recursive for set "/home"
vs=0b0000000000000111; //net can modify and
see,
//local have it
recursive for set "/tmp"
vs=0b0000000000000111;
recursive for set "/var/log"
vs=0b0000000000000111;
recursive for set "/var/run"
vs=0b0000000000000111;
recursive for set "/home/kiko"
vs=0b000000000000001;

recursive for set "/dev/{t,p}ty{,1,2,3,4,5,6,??}"
vs=0b0000000000000111;
recursive for set "/dev/{zero,null,urandom}"
vs=0b0000000000000111;
```

Listing 3.1-1 Inisialisasi *virtual spaces* file-file

#### Penjelasan :

- Penentuan nilai *bitmap virtual spaces* ini berhubungan erat dengan letak *virtual space* proses. Untuk proses-jaringan variabel *virtual spaces*-nya akan diberi nilai sebagai berikut :

vss = 0b110 (hanya 3 bit karena bit yang lain *don't care*)  
vsr = 0b110 atau vsr \=0b001 (ekivalen, karena bit lain *don't care*)  
vsw = 0b100 atau vsw \=0b011

Dengan demikian, user dari internet :

- tidak dapat melihat dan membaca file pada folder `"/home/kiko"`
- hanya dapat menulis file pada folder `/home` (kecuali `/home/kiko`), dan file device terminal dan sebagainya (lihat listing 3.1-1)
- hanya dapat membaca file-file lainnya.

Ketiga hal ini dibatasi lagi dengan kontrol akses standar linux : *permission file* yang dimiliki *user* tersebut.

- Penulis mendapatkan bahwa sintaks `"/dev/{t,p}ty{,1,2,3,4,5,6,??}"` tidak dapat dikenali, karenanya penulisan dilakukan satu per satu (saat percobaan)

### 3. Bagian pendeteksian proses-jaringan

Pada saat awal, penulis mengatakan : pendeteksian dilakukan melalui pendeteksian socket call dan kemudian melihat parameter domainnya (apakah PF\_INET atau PF\_UNIX, misalnya). Hal ini adalah cara yang dicontohkan pada contoh konfigurasi di paket medusa DS9 versi 0.9.2. Namun, pengecekan parameter tersebut tidak dapat memberikan hasil yang benar (bahkan tidak berhasil sama sekali), berikut adalah listing tersebut:

```
on syscall {
  if (action == 102) { /* socketcall */
    if ( trace1 == 1          /* SYS_SOCKET */
        and lpeek trace2 $x /* verify_area() */
        and $x == 2         /* PF_INET */
    ) { // it opens an inet socket
      if (flags ?& 0b10) /* login hack? */
      {
        answer = MED_NOT;
        log "login hack";
      }
      // otherwise, if it is not trusted process
      // and if it still is `local', restrict it.
      else if (flags ?! 0b01 && vs ?& 0b01) {
        log "Process " pid " is restricted now.";
        restrict;
      }
    }
  } else /* we are not interested in other system calls */
    trace_off action;
}
```

Listing 3.1-2 Fungsi pendeteksi pemanggilan *socket call* (dari contoh konfigurasi pada paket Medusa DS9)

Penjelasan :

- Agar *socket call* dapat dimonitor, perlu pemanggilan perintah “trace\_on 102;” pada bagian lain, misal saat terjadi *fork* (terbentuknya proses baru).
- “restrict” adalah sebuah fungsi, lihat listing 3.1-3.
- “lpeek trace2 \$x “ tidak berjalan dengan baik. Variabel *trace2* yang seharusnya merupakan pointer ke parameter domain socket call, bernilai negatif (tidak benar)
- Hal ini pernah ditanyakan pada milis proyek ini, namun belum ada yang menjawab pertanyaan ini.

```
function restrict {
    if (pid==1) /* don't restrict init */
        return 1;

    fsact |= FS_UNLINK; /* watch the unlinks */

    vs = 0b110;
    vss = 0b110;
    vsr /= 0b001; /* can read only files that have at least
                   one of the first two virtual spaces set. */
    vsw /= 0b011; /* can write only files that have the first
                   virtual space set. */

    ecap/=CAP_SETPCAP|CAP_LINUX_IMMUTABLE|CAP_NET_RAW
          |CAP_IPC_OWNER|CAP_SYS_MODULE
          |CAP_SYS_RAWIO|CAP_SYS_PTRACE|CAP_SYS_BOOT
          |CAP_SYS_NICE|CAP_SYS_RESOURCE|CAP_SYS_TIME;

    procact|=P_KILL;
}

```

Listing 3.1-3 Fungsi “restrict” untuk membatasi kemampuan akses proses-jaringan (dari contoh konfigurasi yang disertakan pada paket Medusa DS9)

Penjelasan :

- Fungsi restrict dipanggil saat terjadi *socket call* PF\_INET (listing 3.1-2)
- Penjelasan ada di penjelasan listing 3.1-1
- ecap : effective POSIX capabilities

Karena cara pendeteksian di atas tidak dapat dilakukan, maka penulis menggunakan lain: pendeteksian terhadap pengekseskuan *daemon* telnet, yang dilanjutkan dengan perintah “sh”, sehingga semua akses yang masuk lewat telnet akan dibatasi. Hal ini tidak sebaik cara pendeteksian sebelumnya, karena proses-proses yang terhubung dengan pengguna luar tidak hanya menggunakan telnet. Berikut listingnya :

```
for exec "/usr/sbin/in.telnetd" {
    flags |= 0b100; //tanda untuk menandai,
                  //digunakan saat exec /bin/bash
    procact|=P_EXEC;
}

```

```

    }

for exec "/bin/bash" {
    log "/bin/bash";
    if (flags ?& 0b100){ // bila ya maka ini proses telnet
        vs = 0b110;
        vss = 0b110;
        vsr /= 0b001;
        vsw /= 0b011;
        ecap/=CAP_SETPCAP|CAP_LINUX_IMMUTABLE|CAP_NET_RAW
            |CAP_IPC_OWNER|CAP_SYS_MODULE
            |CAP_SYS_RAWIO|CAP_SYS_PTRACE|CAP_SYS_BOOT
            |CAP_SYS_NICE|CAP_SYS_RESOURCE|CAP_SYS_TIME;

        procact|=P_KILL;
        log "PID telnet restrict "pid;
        trace_on 102;
    }
}

```

Listing 3.1-4 Listing untuk mendeteksi dan membatasi proses yang mengakses sistem melalui telnet

Penjelasan :

- Proses-prose yang mengakses sistem melalui telnet (dan diikuti dengan pemanggilan “/bin/bash” akan diberi batasan akses.
- Listing ini hanya untuk melihat kemampuan Medusa, terdapat banyak kelemahan, yaitu: tidak semua proses-jaringan yang harus melewati *daeomon* telnet dan tidak mesti melakukan pengekseskuan perintah “/bin/bash”.

#### 4. Fungsi-fungsi lain

```

on kill { //`network' user cannot kill constable or init
    // process
    if (vs ?! 0b001)
        if ( target_pid==constable_pid or
target_pid==1)
            answer=MED_NOT;
}
//-----
for access "/etc/shadow"
    if(vs ?! 0b001)
        redirect "/etc/shadow.net";
//-----
for access "/etc/passwd"
    if(vs ?! 0b001)
        redirect "/etc/passwd.net";
//-----
for exec "/bin/ps"
    if (vs ?! 0b110) {
        vs = 0b111;
    }
}

```

```

        vss = 0b1111;
    } else
        redirect "/usr/bin/pwd";
//-----
for unlink "/home/root2/a" //force Tidak berhasil
    force "/usr/src/medusa-0.9.2/Constable/Mlibc/f_exit"
0;

```

Listing 3.1-5 Penggunaan perintah “redirect” dan “force”

Penjelasan :

Ini adalah fungsi-fungsi yang menggunakan perintah “redirect” dan “force”, serta penggunaan salah satu *process action handler* ( on kill )

Percobaan dan hasilnya :

Berikut ini adalah perintah-perintah yang penulis berikan dan jawaban yang diberikan Linux.

```

[root@localhost root]# telnet 127.0.0.1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Red Hat Linux release 9 (Shrike)
Kernel 2.4.18 on an i686
login: kiko
Password:
Last login: Mon Jun 14 08:24:02 from local
id: cannot find name for user ID 507
-bash-2.05b$ ps
/home/kiko
-bash-2.05b$ ls
ls: .: Permission denied
-bash-2.05b$ ps
/home/kiko
-bash-2.05b$ cat /etc/passwd
root:x:x:x:xxx:xxx:/bin/xxxx
nama:x:s:sss23:s:/bin/ls
-bash-2.05b$ cat /etc/shadow
Fatal ERROR !! Turn Off your computer!!
riturnmvmvbnbityitiy3455fgnmgw8
rhtklerjoiu67893r9890012562506
45yrth84568954680oj nlr450
prioj[0400ojerghr3603-
50095i7-97-4fgklfhji45u57-
....3.45..6.....688.-bash-2.05b$
-bash-2.05b$

```

Penjelasan :

- Telnet dilakukan ke komputer sendiri: lokal (127.0.0.1).
- Eksekusi perintah “ps” dirubah ke eksekusi “pwd”.

- Perintah "ls" tidak berhasil karena variabel *virtual spaces* proses-ls (vss, vsw, dan vsr) tidak beririsan dengan variabel vs file-file pada folder "/home/kiko".
- Pengaksesan file "/etc/passwd" diarahkan ke file "/etc/passwd.net" (yang ditampilkan di layar adalah isi file "/etc/passwd.net", hal yang sama untuk file "/etc/shadow".
- "id: cannot find name for user ID 507", disebabkan pada saat mengeksekusi "/bin/bash", proses ini tidak dapat membaca file "/etc/passwd" (diarahkan ke file passwd.net). Jika fungsi "restrict" dilakukan sejak pengekseskuan *daemon* telnet, maka proses *login* tidak bisa berhasil.

### 3.2 Percobaan 2 : Membuat root tidak dapat mengakses folder "/home"

Tujuan : Root tidak dapat membaca file-file pengguna lain.

Metoda :

- Menggunakan variabel *uid* dan *luid* (variabel *built-in* Constable).
- File-file pada folder /home tersebut diberi "password" untuk dapat mengaksesnya. Jadi untuk akses diperlukan bahwa *luid*, *uid*, dan *password* yang sesuai.
- Kemudian membuat hanya proses Constable yang dapat mengakses file konfigurasi
- Membuat Constable berjalan sejak sistem *booting* dan membuat sistem menjadi "mati" atau "reboot" saat Constable dihentikan (keduanya termasuk *feature* pada kernel yang telah di-*patch* dengan Medusa).

File konfigurasi :

```

for access "/usr/src/linux-2.4.20-8/crypto/ciphers/gen-
ctl.h"{
    flags = 0x524ffe16;
    log "flags "flags;
    apply = A_PARENT;
}
recursive for access "/home/kiko/privat"{
    if(uid == 507 and luid == 507){
        if(flags != 0x524ffe16)
            answer = NO;
    }
    else
        answer = NO;
}
for access "/etc/constable.conf"

```

```

        if(pid != constable_pid){
            answer = NO;
            apply = A_PARENT;
        }
    for access "/home/kiko/.bash_history"
        if(uid != 507 or luid != 507)
            answer = NO;

```

Listing 3.2 File konfigurasi untuk membuat folder “/home/kiko/privat” tidak dapat dilihat oleh root

Penjelasan :

Terlihat bahwa file konfigurasi ini tidak menggunakan fasilitas *virtual spaces*, namun hanya memanfaatkan kemampuan *authorization* Constable. Hal ini menunjukkan kemampuan kontrol akses yang dimiliki Medusa bersifat bertingkat-tingkat (lihat bab 2.3 dan 2.4).

Jika file konfigurasi ini dijalankan, maka:

- Untuk masuk ke folder “/home/kiko/privat” diperlukan dua buah hal, yaitu: *user* harus login dengan menggunakan uid = 507 (uid milik kiko) dan mengeksekusi perintah pengaksesan (misalnya ls dan cat) terhadap sebuah file “rahasia” .
- Variabel uid oleh Constable diisi dengan nomor id *user* kiko, sedangkan luid adalah uid saat login (tidak dapat diubah), sehingga salah satu syarat untuk dapat mengakses folder itu adalah *user* login dengan uid 507.
- File rahasia ini dapat file apa saja, ini akan menjadi semacam *password* yang cukup baik (tergantung pemilihan file). Untuk file ini, penulis memilih “/usr/src/linux-2.4.20-8/crypto/ciphers/gen-ctl.h”, yang sebetulnya adalah file palsu (mirip dengan file “gen-ctr.h” yang memang ada). Pembaca bayangkan jika *user* kiko tersebut dapat mengakses komputer itu secara langsung dan kemudian file rahasia tersebut ada pada sebuah CD-R.
- Ketiga hal di atas tidak berguna bila *root* dapat melihat file konfigurasi tersebut, oleh karenanya diperlukan dua hal, yaitu: file konfigurasi harus boleh diakses oleh Constable (lihat penggunaan variabel constable\_pid) dan Constable harus sudah jalan saat pengguna manapun *login* ke sistem (hal ini dapat dilakukan melalui konfigurasi pada kernel)
- Root juga tidak boleh melihat file “. bash\_history” milik kiko.

Percobaan dan hasilnya :

Berikut ini adalah percobaan pengaksesan folder “/home/kiko/privat” oleh root :

```

[root@localhost kiko]# constable /etc/constable.conf
[root@localhost kiko]# cd /home/kiko

```

```

[root@localhost kiko]# ls
Desktop  privat
[root@localhost kiko]# cd privat
bash2: cd: privat: Operation not permitted
[root@localhost kiko]# ls /usr/src/linux-2.4.20-8/crypto/ciphers/gen-ctl.h
/usr/src/linux-2.4.20-8/crypto/ciphers/gen-ctl.h
[root@localhost kiko]# cd /home/kiko/privat
bash2: cd: /home/kiko/privat: Operation not permitted
[root@localhost kiko]#

```

Terlihat bahwa “root” tidak dapat mengakses folder tersebut, walaupun ia telah mengakses file “password”.

Berikut ini adalah percobaan pengaksesan oleh *user* kiko :

```

[kiko@localhost kiko]$ ls
Desktop  privat
[kiko@localhost kiko]$ cd privat
bash: cd: privat: Operation not permitted
[kiko@localhost kiko]$ ls "/usr/src/linux-2.4.20-8/crypto/ciphers/gen-ctl.h"
/usr/src/linux-2.4.20-8/crypto/ciphers/gen-ctl.h
[kiko@localhost kiko]$ cd privat
[kiko@localhost privat]$ ls
bs      hankky  login   loginlog  pexit   shtrc   telnet
xmms
exit   hankky2  logincall  password  rologin  simpen  telnetf
[kiko@localhost privat]$

```

Pada percobaan di atas, terlihat bahwa *user* kiko dapat mengakses folder “privat”, setelah ia mengeksekusi perintah yang mengakses file gen-ctl.h.

Hal-hal Lain :

- Untuk membuat Medusa dan Constable berjalan sejak waktu *booting*, terdapat pilihan yang dapat diaktifkan pada saat meng-compile kernel yang telah di-*patch*.
- Selain itu juga terdapat pilihan agar jika Constable berhenti (dimatikan), maka sistem akan otomatis mati atau *reboot*.
- Cara pengaksesan seperti di atas dapat dikembangkan menjadi metoda yang lebih baik, misalnya :
  - “Pengaksesan hanya diberikan ketika terjadi pengaksesan terhadap misalnya 10 buah file “rahasia” secara berurut dan tidak boleh diselingi oleh file lain.”
- Pada dasarnya “password” ini dapat dibuat dari berbagai *event* yang dapat ditangkap oleh Constable, yang kemudian atas *event* tersebut Constable akan diberikan *flags* terhadap proses tersebut. *Flags* ini akan digunakan oleh

### 3.3 Permasalahan dalam percobaan

#### Deskripsi permasalahan perintah “force” :

Pada permasalahan perintah “force code”, pesan-pesan error terjadi sejak meng-*compile* file Mlibc.c, yaitu terjadi *conflict* pada cara penggunaan (letak pendeklarasian) *lstat*, *fstat*, dan *stat* terhadap pendeklarasian file tersebut di Mlibc.h. (lihat kembali bagian 2: “Kemampuan perangkat lunak Medusa”). Jika ketiga hal tersebut penulis beri “//” (dijadikan komentar), maka peng-*compile*-an dapat dilakukan, namun pesan *error* kembali terjadi saat Constable dijalankan, berupa : “invalid file”<sup>7</sup> terhadap file objek ELF yang dihasilkan. Saat perintah ini dipaksa dijalankan (lihat listing 3.1-5), kembali terjadi pesan *error*.

#### Permasalahan variabel-variabel penyerta suatu perintah :

Penulis tidak tahu cara menggunakan variabel-variabel berikut ini: *trace2*, *data*, *dentry*, *info1*, *info2*. Tidak terdapat manual yang lengkap untuk cara penggunaannya (mungkin harus melihat *source code*-nya atau belum ada dokumentasi dalam bahasa Inggris). Jika seandainya variabel “*dentry*” dapat digunakan (manual yang ada menyebutkannya sebagai penyimpan nama file), maka pembuatan “password” pada bagian 3.2 dapat dilakukan dengan lebih baik. Dokumentasi bahasa pemrograman untuk mengkonfigurasi Constable ada pada lampiran.

#### Permasalahan peng-*compile*-an Medusa :

Sebetulnya penulis belum mengecek apakah pilihan untuk menjalankan Constable sejak *booting* dapat berjalan. Pada saat pertama kali penulis menggunakan Medusa, penulis mengaktifkan pilihan ini, namun hal ini mengakibatkan sistem tidak dapat berjalan (*kernel panic*). Hal tersebut mungkin disebabkan oleh file konfigurasi yang penulis berikan saat itu (dari contoh file konfigurasi) masih mengandung *syntax error*. Namun sampai saat ini pun, penulis belum kembali mencobanya.

#### Medusa versi terbaru : (versi 1.0)

Penulis tidak mendapatkan versi terbaru karena harus menggunakan CVS untuk men-*download*-nya, sedangkan akses tersebut tidak didapat.

---

<sup>7</sup> Hal ini pernah ditanyakan di milis Medusa.

## BAB IV PENUTUP

### Kesimpulan

1. Medusa DS9 merupakan metoda kontrol akses yang memanfaatkan konsep *virtual spaces* dan server otorisasi.
2. Server otorisasi tersebut berupa program *daemon* yang memiliki berbagai fungsi-fungsi yang akan dipanggil saat terjadi operasi-operasi terhadap file dan pemanggilan *system call*.
3. Sistem kontrol akses Medusa terdiri dari dua buah tingkatan, yaitu pada tingkat kernel, berupa pengecekan *virtual spaces*, dan pada tingkat *user space* berupa fungsi-fungsi *handler* (milik server otorisasi) yang akan mengatur jalannya suatu proses (*process*). Kedua hal ini akan digabungkan dengan sistem akses kontrol standar Linux (jadi ada tiga tingkatan).
4. Fungsi handler tersebut dapat kita definisikan dengan menggunakan sebuah bahasa pemrograman mirip C yang memiliki beberapa perintah khusus dan berbagai variabel *built-in* untuk membantu dalam melakukan pengidentifikasian dan pengontrolan jalannya suatu proses.
5. Terdapat dua buah *feature* pada Medusa DS9 versi 0.9-2 yang tidak bekerja dengan baik, yaitu : perintah *force* dan pengambilan parameter-parameter *syscall (trace)* dan parameter *file operation* (misalnya nama file).
6. Oleh karena tidak didapatnya parameter-parameter tersebut, pengidentifikasian yang baik atas suatu proses (misal *network process*) akan sangat sulit.
7. Akan terdapat banyak cara (yang kreatif) dalam menerapkan suatu *security policies* dengan menggunakan Medusa.

## **REFERENSI**

[ZP] Zelem, Marek dan Pikula, Milan “ZP SECURITY FRAMEWORK”

[Doc] Zelem, M., Pikula, M., Ockajak, M. “Package Documentation”

## **HOMEPAGE**

<http://medusa.fornax.sk/>

## **LAMPIRAN**

### **Pengkonfigurasian Constable**

## Configuration of Constable

To achieve better flexibility and a richer configuration, Constable interprets its configuration file as a kind of program. More precisely, it is a set of handlers (functions). Each handler corresponds to a certain type or subtype, depending on the actual data, of query from the kernel, and, depending on the data from the kernel, permits or forbids the operation. If there is more than one handler that match the request, all execute. If no handler for some type of query exists, then Constable returns an error, and the operation is executed normally, as it would if Medusa did not exist. The configuration language is very similar to standard C. There are only few things different and some features added \_ some special keywords, etc.. A description of the configuration language follows:

### BASICS

Programs consist of commands, which end with a semicolon, and of blocks of commands, which are marked with '{' and '}'. Any string between '/\*' and '\*/', or after '//' to the end of the line, is considered to be a comment. Strings have to be between two '"'. The format of integer values is 'VALUE' for decimal, '0bVALUE' for binary and '0xVALUE' for hexadecimal. These are all implemented exactly as in C.

### FUNCTION DEFINITION

The definition of functions is different. Functions are more similar to macros than to functions in C. Each function definition begins with the keyword 'function' followed by the function name and the body of the function in a command block. Functions can return a value using the keyword 'return', which exits the function with the specified return value. Here is an example:

```
function admin_sees {
    if (ecap ?& CAP_SYS_ADMIN) {
        vss = 0b11111111;
        return 0;
    }
    return 1;
}
```

### VARIABLES

All variables are of integer type with lengths of 16 or 32 bits. This also includes pointers. Pointers to strings are a special type of pointer. A variable's type is automatically determined on variable initialization. There are two groups of variables: predefined and local. There are several predefined variables that have special meanings, and their identifiers are explicitly specified. For example, 'answer'. Most predefined variables are related to current data being processed such as various entries in the current process' task\_struct, actual inode information, etc.. They provide some additional information about the subject of an actual request from the kernel, allowing you to fine tune your system security model. All predefined variables except pid, inode\_vs, luid, action, info1 and info2 can be changed (but at your own risk; there are no checks). These changes are then reported back to the kernel and executed. Local variables that have identifiers beginning with \$ (for example '\$foo') are static and do not need to be declared, just as in bash or perl.

## Description of predefined variables and their values

pid	process id of the current process
uid	user id of the current process
euid	effective user id of the current process
suid	saved user id of the current process
fsuid	filesystem user id of the current process
gid	group id of the current process
egid	effective group id of the current process
sgid	saved group id of the current process
fsgid	filesystem group id of the current process
luid	login user id of the current process (this one is set to current->uid only at first call of set{r,e}uid for each process)
vs	bitmap of virtual spaces whose member is current object
vss	bitmap of virtual spaces whose members can see current process
vsr	bitmap of virtual spaces from which current process can read
vsw	bitmap of virtual spaces to which current process can write (all of these are currently 32 bit)
flags	user defined flags of the current process
procact	process actions for the current process which need confirmation from Constable
	P_FORK                process fork
	P_EXEC               execute program
	P_SEXEC              execute set uid program
	P_EXIT               process exit
	P_SETUID             setuid, setreuid, setresuid
	P_KILL               send signal
	P_FSACT              filesystem actions (see below)
	P_CAP                process POSIX capabilities check
	P_PTRACE             process tracing
fsact	filesystem actions for the current process which need confirmation from Constable
	FS_ACCESS           inode access
	FS_CREATE           create file
	FS_LINK             create hard link
	FS_UNLINK           unlink inode (delete)
	FS_SYMLINK          create symbolic link
	FS_MKDIR            create directory
	FS_RMDIR            delete directory
	FS_MKNOD            create special device file
	FS_RENAME           rename file
	FS_TRUNCATE         truncate file
	FS_PERMISSION       file permission check
	FS_EXEC             execute file
ecap	effective POSIX capabilities of the current process (current file in "for set")
icap	inherited POSIX capabilities of the current process (current file in "for set")
pcap	permitted POSIX capabilities of the current process for capabilities constants see /usr/include/linux/capability.h (current file in "for set")
fcap	pcap (for filesystem use only)
acap	icap (for filesystem use only)
data	additional data following standard Medusa packet (usually filenames etc.)
answer	answer code for the kernel

```

ERR          error, indicates security daemon does not know
             how to handle request, so everything will
             proceed without change as though Medusa
             did not exist.
YES          permit operation
NO           forbid operation
SKIP        forbid operation, but return success
OK          permit operation, but proceed with
             standard system permission check if any
             (so permission could be still denied
             by standard system security mechanisms)
             this is the default

apply       information about processes for which to apply actual
             operation
             A_CURRENT   apply operation to current process
                       (this is the default)
             A_PARENT    apply operation to current and
                       parent process
             A_FOR_PARENT  apply operation to parent process and
                       all its children recursively
             A_FOR_LOGIN  apply operation to login process with
                       current luid and all its children
                       recursively

action      system call number
info1       additional information on filesystem or process action
info2       additional information on filesystem or process action
trace1      first parameter of the traced syscall
trace2      second parameter of the traced syscall
trace3      third parameter of the traced syscall
trace4      fourth parameter of the traced syscall
trace5      fifth parameter of the traced syscall
target_pid  process id of the target process (used in operations which
             have some target process, for example kill)
target_uid  user id of the target process
target_euid effective user id of the target process
target_suid saved user id of the target process
target_fsuid filesystem user id of the target process
target_gid  group id of the target process
target_egid effective group id of the target process
target_luid login user id of the target process (this one is set to
             current->uid only at first call of set{r,e}uid for each process)
target_vs   bitmap of virtual spaces whose member include the target object
target_vss  bitmap of virtual spaces whose members can see target process
target_vsr  bitmap of virtual spaces from which target process can read
target_vsw  bitmap of virtual spaces to which target process can write
target_flags user defined flags of the target process
             for process flags constants see /usr/include/linux/sched.h

target_procact  process actions for the target process which need
                confirmation from Constable
target_fsact    filesystem actions for the target process which need
                confirmation from Constable
target_ecap     effective POSIX capabilities of the target process
target_icap     inherited POSIX capabilities of the target process
target_pcap     permitted POSIX capabilities of the target process
inode_uid      actual file owner user id (actual file stands for file's
                inode VFS entry)

inode_gid      actual file owner group id

```

inode\_mode actual file mode  
for file mode constants see /usr/include/linux/stat.h or  
/usr/include/sys/stat.h  
inode\_vs bitmap of virtual subsystems to which actual file belongs  
(changes to this variable are not applied back to the kernel)  
inode\_fsact file actions for the actual file which need confirmation  
from Constable  
constable\_pid process id of the Constable

## OPERATORS

Operators are very similar to those used in C, but there are some changes and new operators. The C equivalent (marked <==>) will follow the description of each operator. All operators except 'not' are binary and associate left to right.

### Description of operators

==	equal	a == b	<==>	a == b
!=	not equal	a != b	<==>	a != b
>	greater	a > b	<==>	a > b
<	less	a > b	<==>	a < b
>=	greater or equal	a >= b	<==>	a >= b
<=	less or equal	a <= b	<==>	a <= b
?&	intersection	a ?& b	<==>	(a & b) != 0
?!	disjunction	a ?! b	<==>	(a & b) == 0
?=	superset	a ?= b	<==>	(a & b) == b
+	plus	a + b	<==>	a + b
-	minus	a - b	<==>	a - b
&	bitwise AND	a & b	<==>	a & b
	bitwise OR	a   b	<==>	a   b
^	bitwise XOR	a ^ b	<==>	a ^ b
=	assignment	a = b	<==>	a = b
+=	assign sum	a += b	<==>	a += b
-=	assign difference	a -= b	<==>	a -= b
=	assign bitwise OR	a  = b	<==>	a  = b
/=	assign bitwise complement	a /= b	<==>	a &= ~b
~=	assign bitwise XOR	a ~= b	<==>	a ^= b
>>=	assign right shift	a >>= b	<==>	a >>= b
<<=	assign left shift	a <<= b	<==>	a <<= b
and	logical AND	a and b	<==>	a && b
or	logical OR	a or b	<==>	a    b
not	logical NOT	not a	<==>	!a

## FURTHER KEYWORDS AND COMMANDS

### Description of basic commands:

function begins the definition of a function. It is followed by function name and body (see above)  
two parameters: function <name> <command block>  
return exits function with return value  
one parameter: return <value>  
if begins a conditional statement. If the expression between '(' and ')' (these go after 'if') is non-zero (true), then the following command block is executed.  
else placed after the end of the 'if' command block and before command block which is executed, if the 'if' expression evaluates to zero (false).  
'if' and 'else' are implemented exactly the same way as in C

redirect redirects any type of access to a given file to another file  
one parameter: redirect <filename>  
(this does not work for directories yet)  
example:

```

for exec "/bin/ping" {
    if (vs ?= 0b1010 )
        redirect "/usr/local/bin/ping-pong"; /* :-) */
}

```

trace\_on enables tracing of a specified syscall for current process  
one parameter: trace\_on <syscall\_number>

trace\_off disables tracing of a specified syscall for current process  
one parameter: trace\_off <syscall\_number>

lpeek reads data from an address in the virtual address space of  
the current process, storing the result in a given variable  
two parameters: lpeek <address> <variable>

lpoke writes a given value to an address in the virtual address space  
of the current process  
two parameters: lpoke <address> <value>

log log messages through the kernel logging facility  
This command can have multiple parameters.  
These are variables or strings to be logged.  
example:

```

on syscall {
    log "uid = " uid "gid = " gid "number = " action;
}

```

log\_inode this logs information about inode  
no parameters

log\_proc similar to log but it automatically logs information about  
the current process

log\_vproc similar to log\_proc but it logs more information about the  
current process

on process action handler followed by the type of handler  
and the handler body  
two parameters: on <handler\_type> <command\_block>  
example:

```

on exec {
    if ( (uid == 111) and (action == 1) ) {
        flags |= 0x8;
        ecap |= CAP_SYS_MODULE;
    }
}

```

This will give any process with certain user defined flags  
that is successfully executed under uid 111 the effective  
capabilities of admin

for filesystem handler followed by the type of handler and  
the handler body  
three parameters: for <handler\_type> <filename> <command\_block>  
example:

```

for unlink "/etc/profile" {
    if ( pid > 234 ) {
        ecap = 0;
        log_fs "someone tried to delete me";
        procact = P_EXEC;
        fsact = FS_CREATE;
        vsw /= 0b1000000000000000;
        answer = NO;
    }
}

```

```
}
This will deny and log an attempt of any process with process
id greater than 234 to delete /etc/profile. It will also lower
the process' effective capabilities, disallow writing to virtual
space 31 for this process and enable watching what files this
process executes and creates.
```

force force execution of code by the current process  
The dynamic loader loads the object file, which has to be  
in ELF format and cannot use any dynamically linked libraries,  
because the loader does not support dynamic linking.  
Code is executed by calling symbol "main".  
Only direct numeric parameters are allowed (no addresses -  
you probably will not be able to dereference them).  
one or more parameters: force "<elf\_obj\_file>" [arg0] [arg1] ...  
You can learn more about code forcing and programming  
functional code for it in code\_forcing.doc

recursive apply recursively for all subdirectories  
example:

```
recursive for set "/var/log"
inode_vs = 0b00000000000000001;
```

this will set virtual space 0 for all files and directories  
in /var/log  
this directive is used only for filesystem handlers

recur same as above

Variables info1 and info2 have different meanings for each type of handler.  
Their meanings will be described for all handler types.

#### Descriptions of the filesystem handler types

The filename parameter of a filesystem handler has to be a string, but it can  
contain something similar to extended regular expressions, so more  
files can be matched even without using keyword 'recursive'. File stands  
for the current file (inode) being processed.

A few examples:

```
"/etc/*.conf"
"/bin/(bash|tcsh|ash|ksh|zsh)"
"/dev/{t,p}ty{,1,2,3,4,5,6,??}"
```

set receive inode information from the Constable to the kernel,  
(vs stands for inode\_vs in this type of handler, linux  
implementation of Medusa allows you to set file capabilities  
here, too)

access file access  
create file creation, info1 = dentry, info2 = mode  
link creation of hard link to the file, info1 = dentry  
unlink file deletion, info1 = dentry  
symlink creation of symbolic link to the file, info1 = dentry  
mkdir directory creation, info1 = dentry, info2 = mode  
rmdir directory deletion, info1 = dentry

mknod	special device file creation, info1 = dentry, info2 = device number
rename	file renaming, info1 = dentry
truncate	file truncation, info1 = dentry, info2 = length
permission	file access permission check, info1 = file access mask
exec	file execution, info1 = argv, info2 = envp, action = 0 when attempting to execute the file, 1 when exec was successful (yes, this handler is called twice during exec operation)

#### Description of the process action handler types

Everything is related to the current process unless stated otherwise.

init	special action executed only at Constable startup (current process is constable here)
fork	fork system call, info1 = clone flags
exec	execution of program, info1 = argv, info2 = envp
sexec	execution of set uid program, info1 = inode vs
setuid	setuid, setreuid and setresuid system calls, info1 = uid
kill	sending signals, kill system call, info1 = signal number
ptrace	performing ptrace operations on target process
capable	process capability check, info1 = capability mask
syscall	any traced syscall (see above: action, trace)

#### MISCELLANEOUS

Constable and its configuration language absolutely are not stupid-proof. You can get an error when reading the configuration file and some runtime errors, but that is all. If you want to do some really stupid things, you can do them, and face the consequences. A partial reason for this is that you can alter some data structures in kernel and there is not much we can do with this. And if it would prevent you from doing many stupid things, it would also prevent you from doing many clever things. But if you build your configuration file carefully, it should be all right. The best way to do this is to keep in mind that anything you enable, you will have to handle later. Have fun.