```
#!/bin/sh
# the next line restarts using tclsh \
exec wish "$0" "$@"
#
# Partie client de l'outil d'exploitation
#
# Eric BURGHARD 28/12/2001

package require templates 1.0
package require BWidget 1.3.1
package require dp 4.0

# charge les données de configuration locales du client
source clicfg.tcl
proc NodeSelected { tree node } {
        # redirige la procédure de selection d'un noeud vers la procédure
de selection de l'objet associé
        global of1 mf1

        # récupère l'objet associé au noeud
        set inst [$tree itemcget $node -data]

        # vide la barre des buttons
        set frame [$of1 getframe]
        foreach widget [winfo children $frame] {
         destroy $widget
        }
        # affichage de l'inspector lié à l'objet
        $inst Inspector $frame

        # vide la barre des indicateurs
        foreach widget [winfo children $mf1.status.indf] {
         destroy $widget
        }
        # affiche la barre d'états lié à l'objet
        foreach lib_i [$inst Stat] {
         $mf1 addindicator -text "$lib_i"
        }

        # appelle la procédure de selection lié à l'objet
        Node CurrentNode $inst
        $inst Select
}
proc NodeOpened node {
        # redirige la procédure d'ouverture d'un noeud vers la procédure
d'ouverture de l'objet associé
        global t1

        # récupère l'objet associé au noeud
```

```
        set inst [$t1 itemcget $node -data]

        # appelle la procédure de selection
        $inst Open
}
proc NodeClosed node {
        # redirige la procédure de fermeture d'un noeud vers la procédure
de fermeture de l'objet associé
        global t1

        # récupère l'objet associé au noeud
        set inst [$t1 itemcget $node -data]

        # appelle la procédure de selection de l'objet
        $inst Close
}
# Node est la superclasse de tous les elements de l'arbre d'exploitation
TClass subclass Node

Node Templates IntrTemplate

Node private currentnode ""

Node classmethod CurrentNode {inst} {
 set [privatevar Node currentnode] $inst
}

# constructeur de noeuds
# -> label: label du noeud
#    type: [static|dynamic]
#
Node method init {label type} {
 $object private label $label
 $object private type $type
 $object private tree ""
 $object private node ""
 $object private opened 0

 return [super init]
}

Node method destroy {} {
 return [super destroy]
}

# méthode pour insérer une instance de type Node dans un arbre
Bwidget::Tree
# <- tree: instance BWidget::Tree
#    parent: position
#    kind: {static, dynamic}, type d'initialisation
#
Node method InitTree {tree parent kind} {
 $object SetNode $tree $parent
 $object InsertChildren $kind
```

```
}

Node method InsertChildren {kind} {
 private $object tree type

 # l'objet ne doit pas etre une feuille et son type doit etre conforme au
type de l'initialisation
 if {(! [$object IsTerminal]) && "$type" == "$kind"} {
  foreach child_i [$object Children] {
   # appel récursif à InitTree pour chaque descendant. L'initialisation
dynamique se fait étape par étape
   $child_i InitTree $tree [$object Node] static
  }
 }
}

# méthode qui retourne la liste des descendants a initialiser dans l'arbre
# <- racine: racine de l'arbre à partir de laquelle les descendants seront
rajoutés
#
Node method Children {} {
}

# méthode permettant de creer un ensemble de widgets pour controler le
noeud
# -> chemin du container dans la hierachie des widgets
#
Node method Inspector {cont} {}

# méthode énumératrice d'états
# <- liste d'états a rajouter dans la barre
#
Node method Stat {} {
 private $object tree node opened

 set stat ""
 if {$opened} {
  set stat [list "[llength [$tree nodes $node]] éléments"]
 }
 return $stat
}

# méthode appellée a la sélection du noeud
#
Node method Select {} {}

# méthode appellée à l'ouverture du noeud (Initialisation dynamique de la
branche)
#
Node method Open {} {
 private $object type opened

 if {$type == "dynamic"} {
  $object InsertChildren dynamic
```

```
 }
 set opened 1
}

# méthode appellée à la fermeture d'un noeud (Vider la branche si son
contenu est dynamique)
#
Node method Close {} {
 private $object type opened

 # detruit tous les descendants pour une branche de type "dynamic"
 if {$type == "dynamic"} {
  $object DeleteChildren
 }
 set opened 0
}

# méthode appellée pour supprimer tous les objets descendants de $object
dans $tree
#
Node method DeleteChildren {} {
 private $object tree

 foreach node_i [$tree nodes [$object Node]] {
  # récupère l'objet associé au noeud
  set inst [$tree itemcget $node_i -data]
  # Efface récursivement l'arbre (parcours en profondeur)
  $inst DeleteChildren
  # détruit l'instance
  $inst destroy
  # efface le noeud de l'arbre
  $tree delete $node_i
 }
}

# méthode determinant si le noeud a des descendants dans l'arbre
# <- [0,1]
Node method IsTerminal {} {
 # noeud terminal par défaut
 return 1
}

Node method Label {} {
 return [$object private label]
}

# méthode permettant de lier $object à une position dans un arbre $tree
# -> tree: instance de BWidget::Tree
#    parent: position dans l'arbre $tree
#
Node method SetNode {t parent} {
 private $object tree node

 set tree $t
```

```
 # Nomme l'objet dans l'arbre à partir du nom du père $args et du nombre de
ses descendants directs
 set node $parent.[llength [$tree nodes $parent]]
 $tree insert end $parent $node -drawcross [expr {[$object IsTerminal] ?
"never" : "allways"}] \
                                -text [$object Label] \
                                -data $object
 return $node
}

# accésseur à la variable node
#
Node method Node {} {
 return [$object private node]
}

# méthode retournant le père du noeud
#
Node method Parent {} {
 set node [$object Node]
 set idx [string last . $node]
 incr idx -1
 return [string range $node 0 $idx]
}

# méthode pour ajouter une barre de bouttons dans le container $cont
# -> cont: conteneur
#    libproc: libellé boutton, procédure, (*)
#
Node method AddButtonBar {cont libproc} {
 set i 0
 foreach {lib_i proc_i} $libproc {
  button $cont.b$i -text $lib_i -command $proc_i
  pack $cont.b$i -fill x -padx 2 -pady 2
  incr i
 }
}
Node subclass Site

# Un object Site est lié à un objet Host
Site method init {label host} {
 $object private host $host
 return [super init $label static]
}

Site method Children {} {
 # Les descendants sont les bases a gérer dans l'arbre
 private $object host

 set res ""
 set i 0
 foreach {base_sid_i site_i} [$host RPC GetSites] {
  lappend res [Base new $object.$i $base_sid_i $base_sid_i $site_i $host]
  incr i
```

```
 }
 return $res
}

Site method Inspector {cont} {
 # créer la liste des opérations possibles pour les noeuds de type site
 private $object host

 if {[$host IsConnected]} {
  return [$object AddButtonBar $cont [list "Tout traiter" "$object DoAll"
"Planifier" "$object PlanAll"]]
 } else {
  return [$object AddButtonBar $cont [list "Se connecter" "$host Connect"]]
 }
}

Site method IsTerminal {} {
 return 0
}
Node subclass Bals

Bals method init {label base host} {
 $object private base $base
 $object private host $host
 return [super init $label static]
}

Bals method Children {} {
 # retourne la liste des types des boites aux lettres
 private $object base host

 set base_sid [$base GetSID]
 return [list \
  [BalsDep new $object.0 départ [$host RPC GetVal BALDEP $base_sid] $host
{}] \
  [BalRec new $object.1 arrivée [$host RPC GetVal BALREC $base_sid] $host
{}] \
 ]
}

Bals method Inspector {cont} {
 # retourne la liste des opérations possibles pour les noeuds de type boite
aux lettres
 private $object base

 return [$object AddButtonBar $cont [list constituer "$object Constitution"
integrer "$object Integration" purger "$object Purge"]]
}

Bals method IsTerminal {} {
 return 0
}
Node subclass Listeners
```

```
Listeners method init {label base host} {
 $object private base $base
 $object private host $host
 return [super init $label static]
}

Listeners method Children {} {
 # retourne la liste des listeners pour une base
 private $object base host

 set base_sid [$base GetSID]
 return [list \
  [Listener new $object.0 ${base_sid}_l1 ${base_sid}_l1 $host] \
  [Listener new $object.1 ${base_sid}_l2 ${base_sid}_l2 $host] \
 ]
}

Listeners method Inspector {cont} {
 # retourne la liste des opérations possibles pour les noeuds de type
regroupement de listeners
 private $object base host

 return [$object AddButtonBar $cont [list lancer "$object RunLists" arreter
"$object StopLists"]]
}

Listeners method IsTerminal {} {
 return 0
}
Node subclass Exploitation

Exploitation method init {label base host} {
 $object private base $base
 $object private host $host
 return [super init $label static]
}

Exploitation method Children {} {
 # retourne la liste des types des boites aux lettres
 private $object base host

 set base_sid [$base GetSID]
 Exploit new $object.0 exploit $base $host
 Prints new $object.1 imprimés [$host RPC GetVal PRINTS $base_sid] $host {}

# $object.0 Register

 return [list \
  $object.0 \
  $object.1 \
 ]
}

Exploitation method Inspector {cont} {
```

```
 # retourne la liste des opérations possibles pour les noeuds de type boite
aux lettres
 private $object base

 return [$object AddButtonBar $cont [list constituer "$object Constitution"
integrer "$object Integration" purger "$object Purge"]]
}

Exploitation method IsTerminal {} {
 return 0
}
# Classe des services (executable lancé par l'exploit susceptible de
générer des sorties écran)
# les sorties sont redirigées vers une zone appropriée de l'interface
graphique
# les sous-classe doivent surcharger Run
#
Node subclass Service

Service Templates IdxTemplate

# méthode de classe permettant d'envoyer toutes les secondes des messages
Flash à toutes les instances
# de toutes les sous-classes de Service
# <- handler after
#
Service classmethod Flash {} {
 foreach obj_i [[super info class] allchildren] {
  $obj_i Flash
 }
 # recommence toutes les secondes
 return [after 1000 Service Flash]
}

Service method init {label id host} {
 $object private id $id
 $object private host $host
 $object private activity 0

 return [super init $label static]
}

Service method destroy {} {
 return [super destroy]
}

Service method Inspector {cont} {
 # retourne la liste des opérations possibles pour les noeuds de type
Service
 private $object id host

 if {[$host RPC [$object info class] IsRunning $id]} {
  set switch "arreter"
 } else {
```

```
  set switch "démarrer"
 }

 return [$object AddButtonBar $cont [list $switch "$object Switch"]]
}

# méthode qui retourne les états a afficher pour le service
#
Service method Stat {} {
 private $object id host

 set stat [super Stat]
 if {[$host RPC [$object info class] IsRunning $id]} {
  lappend stat "actif"
 } else {
  lappend stat "arreté"
 }
 return $stat
}

# méthode a surcharger qui lance le service sur le serveur en asynchrone
# l'activité du service nous arrive sur la connexion d'écoute sous la forme
# log {$objet} {données}
#
Service method RunA {} {
 private $object id host

 # lance le service identifié par la meme clef sur le serveur
 $host RPC [$object info class] RunA $id
}

# méthode permettant de parametrer un effet visuel répétitif (1s) sur le
noeud
#
Service method Flash {} {
 private $object tree node activity

 set c [$tree itemcget $node -fill]
 if {$c != "black"} {
  $tree itemconfigure $node -fill black
 } else {
   $tree itemconfigure $node -fill [expr {$activity ? "green" : "red"}]
 }
}
Service subclass Base

Base method init {label base_sid site host} {
 $object private base_sid $base_sid
 $object private site $site
 $object private host $host
 return [super init $label $base_sid $host]
}

Base method Children {} {
```

```
 # retourne la liste des fonctionalités a gérer par bases
 private $object base_sid host

 Listeners new $object.0 listeners $object $host
 Scrut new $object.1 scrutateur $object $host
 Exploitation new $object.2 exploitation $object $host
 Bals new $object.3 boites $object $host
 Sauvegardes new $object.4 sauvegardes [$host RPC GetVal SAUV $base_sid]
$host {}
 Traces new $object.5 traces [$host RPC GetVal BATCH_LOG $base_sid] $host
{}

 $object.1 Register

 return [list \
  $object.0 \
  $object.1 \
  $object.2 \
  $object.3 \
  $object.4 \
  $object.5 \
 ]
}

Base method Inspector {cont} {
 # retourne la liste des opérations possibles pour les noeuds de type base
 private $object base_sid site host

 if {[$host RPC [$object info class] IsRunning $base_sid]} {
  set switch "arreter"
 } else {
  set switch "démarrer"
 }
 return [$object AddButtonBar $cont [list $switch "$object Switch" "lancer
les traitements" "$object Exploit"]]
}

Base method IsTerminal {} {
 return 0
}

# accésseur à la variable base_sid
Base method GetSID {} {
 return [$object private base_sid]
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Base method Stat {} {
 return [super Stat]
}
Service subclass Listener
```

```
Listener method init {label lis host} {
 $object private host $host
 return [super init $label $lis $host]
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Listener method Stat {} {
 return [super Stat]
}
Service subclass ServiceLog

ServiceLog private currentlog ""

ServiceLog method init {label id host} {
 $object private sock ""
 $object private log ""
 return [super init $label $id $host]
}

ServiceLog method destroy {} {
 $object Unregister
 return [super destroy]
}

# méthode pour s'enregistrer auprès du serveur de manière à recevoir
l'activité
# du service
#
ServiceLog method Register {} {
 private $object id host sock

 if {$sock == ""} {
  set sock [$host UserConnect "SERVICE [$object info class] $id" $object
ShowLog]
 }
}

ServiceLog method Unregister {} {
 private $object sock host

 if {$sock != ""} {
  $host UserDisconnect $sock
 }
}

ServiceLog method Disconnected {_sock} {
 puts "$object Disconnected"
 set
}

ServiceLog method Inspector {cont} {
 # retourne la liste des opérations possibles pour les noeuds de type
```

```
service
 private $object id host

 if {! [$host RPC [$object info class] IsRunning $id]} {
  return [$object AddButtonBar $cont [list démarrer "$object RunA"]]
 }
}

ServiceLog method Select {} {
 private $object log
 private ServiceLog currentlog
 global txt2

 # efface le contenu de la fenetre
 if {$currentlog != "$object"} {
  $txt2 config -state normal
  $txt2 delete 1.0 end
  foreach line $log {
   $txt2 insert end $line\n
  }
  $txt2 config -state disabled
  set currentlog $object
 }
}

# méthode permettant l'affichage de l'activité des taches lancées sur
# sur le serveur en asynchrone dans la zone prévue à cet effet
# -> message a afficher
#
ServiceLog method ShowLog {msg} {
 private $object log
 private ServiceLog currentlog
 global txt2

 lappend log $msg
 # afficher si le log est visile
 if {$currentlog == "$object"} {
  $txt2 config -state normal
  $txt2 insert end $msg\n
  $txt2 config -state disabled
 }
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
ServiceLog method Stat {} {
 return [super Stat]
}
ServiceLog subclass Scrut

Scrut method init {label base host} {
 return [super init $label [$base GetSID] $host]
}
```

```
# méthode pour lancer le scrutateur en asynchrone sur le serveur
#
Scrut method RunA {} {
 private $object host id

 $object ShowLog "lancement du scrutateur $id"
 return [super RunA]
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Scrut method Stat {} {
 return [super Stat]
}
ServiceLog subclass Exploit

Exploit method init {label base host} {
 return [super init $label [$base GetSID] $host]
}

# méthode pour lancer l'exploitation en asynchrone sur le serveur
#
Exploit method RunA {} {
 private $object host id

 $object ShowLog "lancement des traitements sur $id"
 return [super RunA]
}

# méthode pour attirer l'attention sur l'etat de l'exploitation
#
Exploit method Flash {} {
 private $object tree node activity

 # ne clignoter que si l'exploit est active
 if {$activity} {
  super Flash
 }
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Exploit method Stat {} {
 return [super Stat]
}
# Classe des noeuds de type Fichier (leur contenu s'affiche dynamiquement à
l'ouverture du
# noeud dans l'arbre, si c'est un répertoire)
#
Node subclass File
```

```
# variable comptant le nombre de fenetre de visualisation ouvertes
File private winbr 0

File classmethod Stat {host path} {
 return [$host RPC Stat $path]
}

File method init {label path host stget} {
 $object private path $path
 $object private host $host
 $object private txt1 ""
 $object private sock ""
 $object private stget [expr {$stget != {} ? $stget : [File Stat $host
$path]}]
 return [super init $label dynamic]
}

File method Children {} {
 private $object path host

 # retourne le contenu des répertoires
 if {[$object Type] == "directory"} {
  set i 0
  set res ""
  foreach file_i [$host RPC GetDirContents $path] {
   set child_i [$object NewChild $object.$i $file_i]
   if {$child_i != ""} {
    lappend res $child_i
   }
   incr i
  }
  return $res
 }
}

# méthode qui retourne un objet représentant le fichier $file a rajouter
dans les descendants,
# et {} pour ne rien rajouter
# -> path: chemin absolu du fichier
#    name: nom de l'objet a instancier
# <- object instancié de classe File ou descendante | {}
#
File method NewChild {name path} {
 private $object host

 # par défaut, instancie tous les descendants comme étant de type File
 return [new File $name [file tail $path] $path $host]
}

File method Type {} {
 private $object stget

 array set st $stget
 return $st(type)
```

```
}

File method Disconnected {_sock} {
 puts "$object Disconnected"
 set [privatevar $object sock] ""
}

# méthode permettant de visualiser le fichier representé par $object dans
une fenetre
#
File method Watch {} {
 private $object path label host txt1 sock
 private File winbr

 set win [toplevel .t$winbr]
 incr winbr
 eval [subst {
  wm protocol $win WM_DELETE_WINDOW {
   # effacer l'association de l'objet avec la fenetre de visualisation
   set txt1 ""
   # fermer la connexion (ne pas évaluer [])
   $host UserDisconnect \[$object GetSock\]
   # ferme la fenetre
   destroy $win
  }\
 }]
 wm title $win $label

 set sw1 [ScrolledWindow $win.sw1 -auto vertical -scrollbar both]
 pack $sw1 -expand true -fill both -side top
 set txt1 [text [$sw1 getframe].txt1 -state disabled -wrap none]
 $sw1 setwidget $txt1

 # envoie l'ordre de lecture de fichier sur le serveur
 set sock [$host UserConnect "GET $path" $object View]
}

# méthode permettant de télécharger le fichier
# -> chemin de destination
#
File method Download {dest} {
 private $object sock

 set sock
}

# méthode permettant de visualiser des données dans la fenetre ouverte
# préalablement par un appel à Watch
# -> données
#
File method View {data} {
 private $object txt1

 if {$txt1 != ""} {
```

```
  $txt1 config -state normal
  $txt1 insert end $data\n
  $txt1 config -state disabled
 }
}

File method IsTerminal {} {
 # tout fichier qui n'est pas un répertoire est un noeud terminal de
l'arbre
 return [expr {[$object Type] != "directory"}]
}

File method Inspector {cont} {
 if {[$object Type] == "file"} {
  return [$object AddButtonBar $cont [list Visualiser "$object Watch"]]
 }
}

File method Stat {} {
 private $object stget

 set stat [super Stat]
 array set st $stget
 if {$st(type) == "file"} {
  lappend stat "taille $st(size)" "date [clock format $st(mtime) -format
{%d/%m/%Y %H:%M}]"
 }
 return $stat
}

File method GetSock {} {
 return [$object private sock]
}

File subclass BalsDep

BalsDep method NewChild {name path} {
 private $object host

 set stget [File Stat $host $path]
 array set child_st $stget
 if {$child_st(type) == "directory"} {
  return [BalDep new $name [file tail $path] $path $host $stget]
 }
}

BalsDep method Inspector {cont} {
 private $object path host

 # retourne la liste des opérations possibles pour les noeuds de type liste
destinataires depeche
 return [$object AddButtonBar $cont [list Migrer "$object Migration"]]
}
```

```
BalsDep method IsTerminal {} {
 return 0
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
BalsDep method Stat {} {
 return [super Stat]
}
File subclass Depeches

Depeches method NewChild {name path} {
 private $object host

 set stget [File Stat $host $path]
 array set child_st $stget
 if {$child_st(type) == "file"} {
  return [Depeche new $name [file tail $path] $path $host $stget]
 }
}

Depeches method IsTerminal {} {
 return 0
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Depeches method Stat {} {
 return [super Stat]
}
Depeches subclass BalRec

BalRec method Inspector {cont} {
 # retourne la liste des opérations possibles pour les noeuds de type boite
de reception
 private $object path host

 return [$object AddButtonBar $cont [list intégrer "$object Integration"]]
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
BalRec method Stat {} {
 return [super Stat]
}
Depeches subclass BalDep

BalDep method Inspector {cont} {
 # retourne la liste des opérations possibles pour les noeuds de type boite
d'emission
 private $object path host
```

```
 return [$object AddButtonBar $cont [list constituer "$object Constitution"
Migrer "$object Migration"]]
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
BalDep method Stat {} {
 return [super Stat]
}
File subclass Depeche

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Depeche method Stat {} {
 return [super Stat]
}
File subclass Sauvegardes

Sauvegardes method NewChild {name path} {
 private $object host

 set stget [File Stat $host $path]
 array set child_st $stget
 if {$child_st(type) == "file" && [string match "*.tar.gz" $path]} {
  return [Sauvegarde new $name [file tail $path] $path $host $stget]
 }
}

Sauvegardes method Inspector {cont} {
 private $object path host

 # retourne la liste des opérations possibles pour les noeuds de type
regroupement de sauvegardes
 return [$object AddButtonBar $cont [list sauvegarder "$object RunSauv"]]
}

Sauvegardes method IsTerminal {} {
 return 0
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Sauvegardes method Stat {} {
 return [super Stat]
}
File subclass Sauvegarde

Sauvegarde method init {label path host args} {
 return [super init $label "|tar tzvf $path" $host $args]
}
```

```
# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Sauvegarde method Stat {} {
 return [super Stat]
}
File subclass Traces

Traces method NewChild {name path} {
 private $object host

 set stget [File Stat $host $path]
 array set child_st $stget
 if {$child_st(type) == "file"} {
  return [Trace new $name [file tail $path] $path $host $stget]
 }
}

Traces method IsTerminal {} {
 return 0
}

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Traces method Stat {} {
 return [super Stat]
}
File subclass Trace

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Trace method Stat {} {
 return [super Stat]
}
File subclass Prints

Prints method NewChild {name path} {
 private $object host

 set stget [File Stat $host $path]
 array set child_st $stget
 if {$child_st(type) == "file"} {
  return [Print new $name [file tail $path] $path $host $stget]
 }
}

Prints method IsTerminal {} {
 return 0
}

# ! bug du package Class. super représente $objet du type de sa superclasse
```

```
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Prints method Stat {} {
 return [super Stat]
}
File subclass Print

# ! bug du package Class. super représente $objet du type de sa superclasse
au lieu de repésenter
# $objet du type la classe de la méthode dans laquelle super apparait
Print method Stat {} {
 return [super Stat]
}
# caractérise une connection RPC à un serveur le port $port
TClass subclass Host
MClassTemplate Host

Host method init {host port} {
 $object private host $host
 $object private port $port
 $object private rpc ""
 $object private socks ""

 $object MClassInit
 return [super init]
}

Host method destroy {} {
 private $object socks

 $object MClassDestroy
 $object Disconnect
 foreach sock_i $socks {
  $object UserDisconnect $sock_i
 }
 return [super destroy]
}

# méthode pour établir une connection RPC au serveur
# -> type: quiet (connection sans message d'erreur)
#
Host method Connect {args} {
 private $object host port rpc

 # ouvre la connexion rpc
 set res [catch {set rpc [dp_MakeRPCClient $host $port]} msg]
 if {$args != "quiet"} {
  error $msg
 }
 return [expr {! $res}]
}

Host method Disconnect {} {
 if {[$object IsConnected]} {
```

```
  close [$object private rpc]
 }
}

# méthode pour établir une connexion Utilisateur au serveur
Host method UserConnect {cmd obj handle} {
 private $object socks host port

 # la connexion s'effectue sur le port rpc + 1
 set sock [socket $host [expr $port + 1]]
 fconfigure $sock -blocking 0 -buffering line
 fileevent $sock readable [list $object HandleUserData $sock $obj $handle]
# fileevent $sock writable [list puts $sock $cmd ; fileevent $sock writable
{}]
 puts $sock $cmd
 lappend socks $sock
 return $sock
}

Host method HandleUserData {sock obj handle} {
 if {[eof $sock]} {
  $object UserDisconnect $sock
  $obj Disconnected $sock
 } else {
  $obj $handle [read $sock]
 }
}

Host method UserDisconnect {sock} {
 private $object socks

 puts "UserDisconnect $sock \[$socks\]"
 set idx [lsearch -exact $socks $sock]
 if {$idx != -1} {
  fileevent $sock readable {}
  close $sock
  set socks [lreplace $socks $idx $idx]
 }
}

Host method RPC {args} {
 private $object rpc

 if {[$object IsConnected]} {
  return [eval dp_RPC $rpc $args]
 }
}

Host method RDO {args} {
 private $object rpc

 if {[$object IsConnected]} {
  return [eval dp_RDO $rpc $args]
 }
```

```
}

Host method IsConnected {} {
 private $object rpc

 return [expr {$rpc != ""}]
}
wm title . "Aster - Exploit"
wm geometry . 500x500+[expr int(([winfo screenheight .] - [winfo reqheight
.])/ 2)]+[expr int(([winfo screenwidth .] - [winfo reqwidth .])/ 2)]
wm protocol . WM_DELETE_WINDOW {
 # ferme toutes les connections RPC
 foreach host_i [Host info children] {
  $host_i Disconnect
 }
 # desenregister les services
 foreach child_i [ServiceLog allchildren] {
  $child_i destroy
 }
 # ferme la fenetre
 destroy .
}

Service Flash
set descmenu {
    "&File" {} {} 0 {
        {command "&New"     {} "Create a new document"     {Ctrl n}
-command Menu::new}
        {command "&Open..." {} "Open an existing document" {Ctrl o}
-command Menu::open}
        {command "&Save"    open "Save the document" {Ctrl s} -command
Menu::save}
        {cascad  "&Export"  {} export 0 {
            {command "Format &1" open "Export document to format 1" {}
-command {Menu::export 1}}
            {command "Format &2" open "Export document to format 2" {}
-command {Menu::export 2}}
        }}
        {separator}
        {cascad "&Recent files" {} recent 0 {}}
        {separator}
        {command "E&xit" {} "Exit the application" {} -command Menu::exit}
    }
    "&Options" {} {} 0 {
        {checkbutton "Toolbar" {} "Show/hide toolbar" {}
            -variable Menu::_drawtoolbar
            -command  {$Menu::_mainframe showtoolbar toolbar
$Menu::_drawtoolbar}
        }
    }
}

set mf1 [MainFrame .mf1 -menu $descmenu]
pack .mf1 -expand true -fill both
```

```
set pw2 [PanedWindow [$mf1 getframe].pw2 -side left]
pack [$mf1 getframe].pw2 -expand true -fill both
$pw2 add -minsize 100
$pw2 add -minsize 100 -weight 0
set pw1 [PanedWindow [$pw2 getframe 0].pw1 -side top]
pack [$pw2 getframe 0].pw1 -expand true -fill both
$pw1 add -minsize 100
$pw1 add -minsize 100
set sc1 [ScrolledWindow [$pw1 getframe 0].sc1 -auto both -relief ridge
-scrollbar both]
pack [$pw1 getframe 0].sc1 -expand true -fill both -side left
set t1 [Tree [$sc1 getframe].t1 -closecmd NodeClosed -opencmd NodeOpened
-selectcommand NodeSelected]
$sc1 setwidget $t1
set of1 [TitleFrame [$pw1 getframe 1].of1 -text Opérations]
pack [$pw1 getframe 1].of1 -expand true -fill both
set sc2 [ScrolledWindow [$pw2 getframe 1].sc2 -auto vertical -scrollbar
both]
pack [$pw2 getframe 1].sc2 -expand true -fill both
# constuire la cellule texte pour afficher l'activité des scrutateurs

set txt2 [text [$sc2 getframe].txt2 -height 5 -state disabled -wrap none]
$sc2 setwidget $txt2
set currentlog ""
set i 0
foreach {lib_i host_i port_i} $libhostport {
 set rpchost [Host new $host_i $host_i $port_i]
 $rpchost Connect quiet
 # instanciation des racine représentants un ensemble de sites
 set site_i [Site new s$i $lib_i $rpchost]
 $site_i InitTree $t1 root static
 incr i
}
```