Implantation d'un hypercall: ajout d'une nouvelle commande XL

Stella Bitchebe et Alain Tchana (alain.tchana@ense-lyon.fr)

1. Objectif

Une commande xl s'appuyant sur un hypercall, on peut être amené à définir un nouvel hypercall lorsqu'on souhaite implémenter une nouvelle commande xl. Mais heureusement il peut exister des hypercalls qui font déjà ce dont on a besoin, et dans ce cas il ne nous reste plus qu'à nous appuyer sur ceux-ci. L'objectif de ce TP est donc d'utiliser un hypercall existant pour implémenter une nouvelle commande xl.

2. Les hypercalls

a. Définition

Littéralement hypercall signifie *appel à l'hyperviseur*.

Un hypercall est pour un hyperviseur, ce qu'est un syscall (appel système) est pour un système d'exploitation. Un hypercall est une interruption logicielle d'une machine virtuelle (VM) vers un hyperviseur, tout comme un appel système est une interruption logicielle d'une application vers le noyau. Les VMs utiliseront les hypercalls pour demander à l'hyperviseur d'effectuer des opérations privilégiées (pour lesquelles elles n'ont pas les droits suffisants) telles que des opérations sur la mémoire (e.g. mise à jour des tables de pages).

b. Interface pour la définition des hypercalls dans XEN

Pour l'implémentation d'un nouvel hypercall dans XEN il faut définir un certain nombre de :

- Fonctions :

- HYPERVISOR_mmu_update()
- HYPERVISOR_sched_op()
- HYPERVISOR_console_io()
- etc.

- Structures de données :
  - struct xen_domctl
  - struct xen_sysctl
  - struct arch_shared_info
  - etc.
- Types :
  - typedef arch_vcpu_info_t
  - typedef domid_t
  - typedef xen_hvm_get_time_t
  - etc.
- etc.

L'ajout d'un hypercall nécessite sa définition à la fois au niveau de l'hyperviseur et de la VM (noyau de la VM : **Linux** dans notre cas).

Au niveau de l'hyperviseur :

- Déclaration d'une constante identifiant l'hypercall, dans le fichier ***xen/include/public/xen.h*** (***__HYPERVISOR_my_hypercall***)

```
#define __HYPERVISOR_hvm_op             34
#define __HYPERVISOR_sysctl             35
#define __HYPERVISOR_domctl             36
```

- Ajout d'une entrée dans la table d'hypercalls (il s'agit d'enregistrer l'hypercall et son nombre d'arguments), dans le fichier ***xen/arch/x86/hypercall.c***

```
ARGS(hvm_op, 2),
ARGS(sysctl, 1),
ARGS(domctl, 1),
```

- Ajout du prototype de la fonction handler dans le fichier ***xen/include/xen/hypercall.h*** (***do_my_hypercall***)

```
extern long
do_domctl(
    XEN_GUEST_HANDLE_PARAM(xen_domctl_t) u_domctl);

extern long
do_sysctl(
    XEN_GUEST_HANDLE_PARAM(xen_sysctl_t) u_sysctl);
```

- Implémentation de la fonction ***do_my_hypercall*** proprement dite. Elle se fait dans un fichier en fonction de ce dont traite l'hypercall. Par exemple l'implémentation de l'hypercall domctl, est faite dans le fichier ***xen/common/domctl.c***

Au niveau du noyau de la VM :

- Déclaration d'une constante identifiant l'hypercall, dans le fichier ***include/xen/interface/xen.h*** (***__HYPERVISOR_my_hypercall***)

```
#define __HYPERVISOR_update_descriptor    10
#define __HYPERVISOR_memory_op            12
#define __HYPERVISOR_multicall            13
```

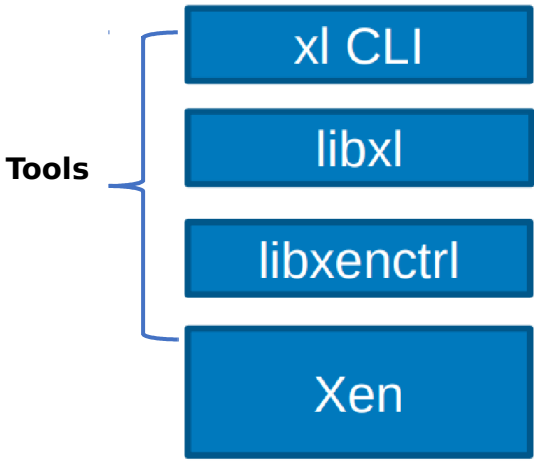
- Ajout du prototype et de la définition de l'hypercall dans le fichier ***arch/x86/include/asm/xen/hypercall.h***

```
static inline long
HYPERVISOR_memory_op(unsigned int cmd, void *arg)
{
    return _hypercall2(long, memory_op, cmd, arg);
}

static inline int
HYPERVISOR_multicall(void *call_list, uint32_t nr_calls)
{
    return _hypercall2(int, multicall, call_list, nr_calls);
}
```

3. Les commandes xl

Une commande xl est le moyen d'encapsuler un hypercall afin de pouvoir l'invoquer depuis la ligne de commande. Tout comme avec un hypercall, pour implémenter une nouvelle commande xl, il y a un certains nombre d'étapes à suivre, et la définition se fait à 2 niveaux: premièrement au niveau des tools et ensuite dans l'hyperviseur (le dossier xen). Voir la figure ci-contre.



Au niveau des tools :

a. Il faut tout d'abord ajouter une entrée à la table/liste des commandes dans le fichier ***tools/xl/xl_cmdtable.c***. il s'agit de déclarer le nom de la commande ***my_xl_command***, le nombre de paramètres attendus ainsi que les différentes options, d'expliquer comment elle doit être appelée et quel est son rôle.

b. Ensuite, il faut ajouter la signature de cette commande dans le fichier ***tools/xl/xl.h***.

c. Dans un des fichiers c du dossier libxc (***librairy xencontrol***), il faut donner la définition de la fonction qui va exécuter l'hypercall pour notre commande. En fonction du traitement effectué on utilisera un fichier ou un autre: ***xc_domain.c*** s'il s'agit des domaines, ***xc_evtchn.c*** s'il s'agit des eventchannel, etc. On aura donc des fonctions avec pour noms ***xc_domain_my_xl_command*** ou ***xc_evtchn_my_xl_command***. La signature de cette fonction sera enregistrée dans le fichier ***tools/libxc/include/xenctrl.h***. C'est dans cette méthode qu'on fera appel à l'hypercall sur lequel s'appuie la commande xl (***do_domctl***, ***do_sysctl***, ***etc.***).

d. Maintenant il faut relier ***my_xl_command*** à ***xc_domain_my_xl_command***. Pour se faire il faut définir une fonction dans un des fichiers c du dossier libxl, ***libxl_domain_my_xl_command***, qui appellera la fonction ***xc_domain_my_xl_command***. Ensuite, dans un des fichiers c du dossier xl, nous allons mettre le corps de la fonction ***my_xl_command*** (dont nous avons défini le prototype au début) qui appellera à son tour ***libxl_domain_my_xl_command***. Le prototype de ***libxl_domain_my_xl_command*** est enregistré dans le fichier ***tools/libxl/libxl.h***.

Au niveau de l'hyperviseur, si on utilise un hypercall existant, il n'y a pas grand chose à définir. Toutefois on peut être amené à modifier le comportement des hypercalls sur lesquels on s'appuie (si on a des comportements particuliers à définir ou rajouter). On peut également être amené à modifier les structures de données impliquées dans ces hypercalls (si on a des données supplémentaires à introduire ou d'autres informations à transmettre) ou à en définir de nouvelles.

4. Exemples de commande xl

Dans le cadre du projet, nous aurons à implémenter de nouvelles commandes xl relatives à l'utilisation des différentes fonctionnalités que nous utilisons (par exemple pour les activer ou les désactiver).

Pour le PML par exemple, qui n'est pas activé par défaut pour une VM, il faut trouver le moyen de l'activer lorsqu'on veut l'utiliser et le désactiver directement après. Nous allons donc implémenter les commandes suivantes entre autres :

- xl enable-logdirty dom_id
- xl disable-logdirty dom_id
- xl vtf options

La troisième commande contrairement aux 2 premières, renverra pour certaines options, un résultat de la part de l'hyperviseur (comme la commande ***xl list*** qui renvoie la liste des VMs en cours d'exécution par l'hyperviseur ainsi que les

informations relatives à celles-ci).

A. **xl enable-logdirty/disable-logdirty dom_id**

Ces commandes permettent d'activer/désactiver le mode ***logdirty*** pour une VM depuis le dom0. Le mode logdirty est celui qui permet la virtualisation de la mémoire de la VM, à travers l'utilisation de l'EPT, sans lequel le PML ne peut être utilisé.

En suivant les étapes d'implémentation d'une commande xl précédemment énumérées, nous avons :

i. Fichiers **tools/xl/xl_cmdtable.c** et **tools/xl/xl.h** resp.

```
{ "enable-logdirty",
  &main_enable_logdirty, 1, 1,
  "Enable log dirty mode for domain <domid>",
  "[options] [Domain]",
  "-h                       Print this help.\n"
},
```

```
int main_clean_dirty_bitmap(int argc, char **argv);
int main_enable_logdirty(int argc, char **argv);
int main_disable_logdirty(int argc, char **argv);
```

ii. Fichiers **tools/libxc/include/xenctrl.h** et **tools/libxc/xc_domain.c** resp.

```
int xc_domain_enable_logdirty(xc_interface *xch, uint32_t dom, int is_vtf, uint32_t pid_list[25]);
int xc_domain_disable_logdirty(xc_interface *xch, uint32_t dom, int is_vtf, uint32_t pid_list[25]);
```

```c
int xc_domain_enable_logdirty(xc_interface *xch, uint32_t dom, int is_vtf, uint32_t pid_list[25])
{
    int rc;
    DECLARE_DOMCTL;
    memset(&domctl, 0, sizeof(domctl));

    if(is_vtf) {
        memcpy(domctl.u.vtf.pids, pid_list, 25*sizeof(int));
        domctl.cmd = XEN_DOMCTL_vtf;
        domctl.u.vtf.hw = XEN_DOMCTL_vtf_hw_PML;
    }
    else
    {
        domctl.cmd = XEN_DOMCTL_shadow_op;
        domctl.domain = (domid_t)dom;
    }

    domctl.u.shadow_op.op     = XEN_DOMCTL_SHADOW_OP_ENABLE_LOGDIRTY;

    rc = do_domctl(xch, &domctl);

    return rc;
}
```

iii. Fichiers **tools/xl/libxl.h** et **tools/xl/libxl_domain.c** resp.

```c
int libxl_domain_disable_logdirty(libxl_ctx *ctx, uint32_t domid);
int libxl_domain_enable_logdirty(libxl_ctx *ctx, uint32_t domid);
int libxl_collect_dirty_logs(libxl_ctx *ctx, uint32_t domid);
int libxl_clean_dirty_bitmap(libxl_ctx *ctx, uint32_t domid);
```

```c
int libxl_domain_enable_logdirty(libxl_ctx *ctx, uint32_t domid)
{
    int ret;
    GC_INIT(ctx);

    ret = xc_domain_enable_logdirty(ctx->xch, domid, 0, NULL);
    GC_FREE;

    return ret;
```

iv. Fichier **tools/xl/xl_vmcontrol.c**

```c
int main_enable_logdirty(int argc, char **argv)
{
    uint32_t domid = atoi(argv[1]);
    return libxl_domain_enable_logdirty(ctx, domid);
}
```

v. Exemple d'appel de la commande pour la VM 1

```
~$ sudo xl enable-logdirty 1
```

B. **xl vtf (VirTualization Features)**

A l'opposé des commandes précédentes, la nouvelle commande ***xl vtf*** permettra aux VMs de demander elles mêmes l'activation/désactivation d'une fonctionnalité à l'hyperviseur. Normalement une VM non privilégiée ou domU (toute VM autre que le dom0) n'a pas les privilèges pour appeler une commande xl, nous allons donc devoir modifier le comportement des hypercalls que nous allons utiliser afin de pouvoir donner, à un domU exécutant la commande ***xl vtf***, les droits nécessaires pour :

- Connaître la liste des fonctionnalités qu'elle peut activer : ***xl vtf --list/-l***
- Activer une de ces fonctionnalités : ***xl vtf --enable/-e PML***
- Désactiver une de ces fonctionnalités : ***xl vtf --disable/-d PML***

a. Au niveau des tools

i. Fichiers **tools/xl/xl_cmdtable.c** et **tools/xl/xl.h** resp.

```
{ "vtf",
  &main_vtf, 0, 0,
  "Informations about VTF (Virtual Technology Extensions)",
  "[options] [VMX] [process_pid]",
  "-l, --list               List VMX supported by the vtf library\n"
  "-e, --enable             Enable a vtf for this domain\n"
  "-d, --disable            Disable a vtf for this domain\n"
  "-h  --help               Print this help."
},
```

```
int main_reboot(int argc, char **argv);
int main_vtf(int argc, char **argv);
int main_collect_dirty_logs(int argc, char **argv);
```

ii. Fichiers **tools/libxc/include/xenctrl.h** et **tools/libxc/xc_domain.c** resp.

```
int xc_vtf(xc_interface *xch, int arg, xc_vtf_cpuinfo_t *xcinfo, char *hw, uint32_t pid_list[25]);
int xc_collect_dirty_logs(xc_interface *xch, uint32_t dom);
int xc_clean_dirty_bitmap(xc_interface *xch, uint32_t dom);
```

```c
int xc_vtf(xc_interface *xch, int arg, xc_vtf_cpuinfo_t *xcinfo, char *hw, uint32_t pid_list[25])
{
    int ret = 0;
    uint32_t dom = -1;
    DECLARE_SYSCTL;
    DECLARE_HYPERCALL_BOUNCE(xcinfo, sizeof(*xcinfo), XC_HYPERCALL_BUFFER_BOUNCE_OUT);

    switch (arg)
    {
    case 1:
        if (xc_hypercall_bounce_pre(xch, xcinfo))
            return -1;

        sysctl.cmd = XEN_SYSCTL_vtf;
        sysctl.u.vtf.check = 1;
        sysctl.u.vtf.cmd.action = XEN_SYSCTL_vtf_action_list;
        set_xen_guest_handle(sysctl.u.vtf.buffer, xcinfo);
        ret = xc_sysctl(xch, &sysctl);
        xc_hypercall_bounce_post(xch, xcinfo);
        break;

    case 2:
        if (!strcmp(hw, "PML"))
            ret = xc_domain_enable_logdirty(xch, dom, 1, pid_list);
        else if (!strcmp(hw, "EPT"))
        {
        }

        break;

    case 3:
        if (!strcmp(hw, "PML"))
            ret = xc_domain_disable_logdirty(xch, dom, 1, pid_list);
        else if (!strcmp(hw, "EPT"))
        {
        }

        break;
    }

    return ret;
}
```

Nous allons à présent donner quelques explications sur le *case 1*, i.e. l'option "**--list**". Cette option a ceci de particulier qu'à la fin de l'hypercall il faudra renvoyer un résultat à la VM, à savoir la liste des fonctionnalités matérielles auxquelles l'hyperviseur lui donne accès.

- ***set_xen_guest_handle(sysctl.u.vtf.buffer, xcinfo)*** : cette ligne relie les 2 variables ***buffer*** et ***xcinfo*** pour l'hypercall qui va être appelé, de sorte que les informations qui seront contenues dans ***buffer*** à la fin puissent être transmises à ***xcinfo*** afin de pouvoir en lire le contenu.
- ***ret = xc_sysctl(xch, &sysctl)*** : cette ligne va lancer l'hypercall ***do_sysctl***

proprement dit.

- ***xc_hypercall_bounce_post(xch, xcinfo)*** : cette dernière place dans ***xcinfo*** le résultat de l'hypercall qui nous intéresse, à savoir la liste de fonctionnalités à afficher à la VM.

iii. Fichiers **tools/xl/libxl.h** et **tools/xl/libxl.c** resp.

```
int libxl_vtf(libxl_ctx *ctx, int arg, char *hw, uint32_t pid_list[25]);
```

```
int libxl_vtf(libxl_ctx *ctx, int arg, char *hw, uint32_t pid_list[25])
{
    int ret = 0;
    xc_vtf_cpuinfo_t xcinfo;
    GC_INIT(ctx);

    switch (arg)
    {
    case 1:
        ret = xc_vtf(ctx->xch, arg, &xcinfo, hw, pid_list);
        printf("VMX: Supported advanced features:\n");
        if(xcinfo.pml)
            printf("    - PML: Page Modification Logging.\n");
        if(xcinfo.ept)
            printf("    - EPT: Extended Page Table.\n");
        break;

    case 2:
    case 3:
        ret = xc_vtf(ctx->xch, arg, &xcinfo, hw, pid_list);
        break;

    default:
        break;
    }

    GC_FREE;

    return ret;
}
```

A la fin de l'exécution de la fonction ***xc_vtf***, on a dans la variable ***xcinfo***, le résultat de l'hypercall (voir l'explication précédente 4.B.a.ii). Si ***xcinfo.pml*** est vrai ceci signifie que l'hyperviseur donne à la VM la possibilité d'utiliser la fonctionnalité du PML.

Le type ***xc_vtf_cpuinfo_t*** est défini dans le fichier tools/libxc/include/xencrtl.h :

```
typedef xen_sysctl_vtf_getcpuinfo_t xc_vtf_cpuinfo_t;
```

Par la suite, nous verrons dans les modifications au niveau de l'hyperviseur, le lien avec xen_sysctl_vtf_getcpuinfo_t.

iv. Fichier **tools/xl/xl_vcpu.c**

```
int main_vtf(int argc, char **argv)
{
    #define LIST 1
    #define ENABLE 2
    #define DISABLE 3

    int opt, arg=0, pid_idx=0;//, *pid_list
    uint32_t pid_list[25];
    char *hw = NULL;
    static struct option opts[] = {
        {"list", 0, 0, 'l'},
        {"enable", 1, 0, 'e'},
        {"disable", 1, 0, 'd'},
        COMMON_LONG_OPTS
    };

    SWITCH_FOREACH_OPT(opt, "led", opts, "vtf", 0) {
        case 'l':
            arg = LIST;
            break;
        case 'e':
            arg = ENABLE;
            hw = optarg;
            break;
        case 'd':
            arg = DISABLE;
            hw = optarg;
            break;
    }

    hw = argv[optind];
    argc--;

    if( argc > 3 && !strcmp(argv[++optind], "-p") )
    {
        while(optind < argc)
        {
            pid_list[pid_idx++] = atoi(argv[++optind]);
        }
        pid_list[pid_idx] = (int)(uintptr_t)NULL;
    }

    return libxl_vtf(ctx, arg, hw, pid_list);
}
```

Cette fonction est appelée lorsqu'on tape la commande ***xl vtf --option***. Mais avant cela, la fonction ***int main(arg)*** du fichier ***tools/xl/xl.c***, qui initialise la contexte pour toute commande ***xl***, est d'abord appelée. Nous aurons donc à intervenir également

au niveau de cette fonction pour contourner les politiques de sécurité de l'hyperseur et permettre à une VM non priviligiée de pouvoir exécuter cette commande xl.

v. Fichier **tools/xl/xl.c**

- **Dans la méthode main(arg)**

  À l'initialisation du contexte xl, on regarde s'il s'agit de ***xl vtf*** et si c'est le cas on met à ***true*** le ***flag*** correspondant :

```
if(!strcmp(cmd, "vtf"))
    is_vtf_cmd = 1;

parse_global_config(XL_GLOBAL_CONFIG, config_data, config_len);
```

- **Dans la méthode parse_global_config(arg)**

```
if (!xlu_cfg_get_long (config, "max_grant_frames", &l, 0))
    max_grant_frames = l;
else {
    libxl_physinfo_init(&physinfo);
    if(is_vtf_cmd){
        physinfo.domain_unpriv = LIBXL_DOMAIN_UNPRIV_DOMU;
    }
    max_grant_frames = (libxl_get_physinfo(ctx, &physinfo) != 0 ||
                        !(physinfo.max_possible_mfn >> 32))
                       ? 32 : 64;
    libxl_physinfo_dispose(&physinfo);
}
```

La méthode ***libxl_physinfo_init(struct libxl_physinfo *physinfo)*** va effectuer l'initialisation proprement dite du contexte ***xl***. Et dans sa définition, il y a une étape où elle effectue également des contrôles de sécurité pour vérifier qu'il ne s'agit pas d'un domU.

Ce contrôle est également fait dans la méthode ***libxl_get_physinfo*** (qui prend en paramètre ***physinfo***), définie dans le fichier ***tools/libxl/libxl.c*** :

```c
int libxl_get_physinfo(libxl_ctx *ctx, libxl_physinfo *physinfo)
{
    xc_physinfo_t xcphysinfo = { 0 };
    int rc;
    long l;
    GC_INIT(ctx);

    if(physinfo->domain_unpriv){
       xcphysinfo.vtf_cmd = 1;
    }
    rc = xc_physinfo(ctx->xch, &xcphysinfo);
    if (rc != 0) {
        LOGE(ERROR, "getting physinfo");
        GC_FREE;
        return ERROR_FAIL;
    }
```

Comme nous pouvons le voir dans cette portion de code, il s'agit de rajouter un flag parmi les champs de la structure ***xc_physinfo_t***, car il est utilisé pour lancer l'hypercall qui va renvoyer les informations physiques (telles que le nombre de cpus, la quantité de mémoire, le nombres de pages libres, etc.) à chaque création de VM. Cette structure est définie dans le fichier ***xen/include/public/sysctl.h*** :

```c
struct xen_sysctl_physinfo {
    boolean_t vtf_cmd;
    uint32_t threads_per_core;
    uint32_t cores_per_socket;
    uint32_t nr_cpus;     /* # CPUs currently online */
```

Intéressons nous maintenant au rajout dans la structure ***libxl_physinfo***, du champ ***domain_unpriv***, qui nous permettra de d'effectuer la commande xl s'il s'agit d'un domU et d'une commande ***vtf***. L'ajout de ce champ se fait dans le fichier suivant.

vi. Fichier **tools/libxl/libxl_types.idl**

```
#
# Constants / Enumerations
#

libxl_domain_unpriv = Enumeration("domain_unpriv", [
    (0, "DOM0"),
    (1, "DOMU"),
])
```

```
libxl_physinfo = Struct("physinfo", [
    ("domain_unpriv", libxl_domain_unpriv),
    ("threads_per_core", uint32),
    ("cores_per_socket", uint32),

    ("max_cpu_id", uint32),
```

Après recompilation, les modifications enregistrées dans ce fichier sont prises en compte et ajoutées dans les différentes structures données.

b. Au niveau de l'hyperviseur

Etant donné que nous voulons accorder des droits particuliers aux VMs pour exécuter les commandes ***xl vtf***, nous devons trouver le moyen de distinguer celles-ci des autres commandes xl (car nous ne voulons pas non plus enfreindre toutes les politiques de sécurité et permettre à une VM d'effectuer des opérations sur la mémoire par exemple). Pour cela, nous devons apporter des modifications au comportement des hypercalls que nous utilisons mais aussi aux structures de données y relatives.

i. Hypercall do_domctl

Nous allons utiliser cet hypercall particulièrement pour l'activation et la désactivation des fonctionnalités (options **--enable** et **--disable**).

- Modification de la structure de données ***xen_domctl*** (dans le fichier ***xen/include/public/domctl.h***) : nous allons rajouter aux champs de la structure ***xen_domctl***, une structure ***xen_domctl_vtf***.

```
struct xen_domctl_vtf {
    uint32_t hw;
    #define XEN_DOMCTL_vtf_hw_PML        1
    #define XEN_DOMCTL_vtf_hw_EPT        2
    uint32_t pids[25];
};
typedef struct xen_domctl_vtf xen_domctl_vtf_t;
DEFINE_XEN_GUEST_HANDLE(xen_domctl_vtf_t);
```

```
#define XEN_DOMCTL_vuart_op                      81
#define XEN_DOMCTL_vtf                           82
#define XEN_DOMCTL_gdbsx_guestmemio            1000
```

```
    struct xen_domctl_vuart_op          vuart_op;
    struct xen_domctl_vtf               vtf;
    uint8_t                             pad[128];
} u;
```

- Modification de la structure de données ***domain*** (dans le fichier ***xen/include/xen/sched.h***) : nous allons y rajouter un flag qui nous dira si la commande xl que la VM veut exécuter est ***xl vtf***, si oui nous lui donnerons les permissions nécessaires pour que l'hypercall aboutisse sinon une erreur sera générée; nous rajouterons également un tableau qui contiendra la liste des pids pour lesquels la VM demande l'activation (ceci nous permettra de voir comment passer des parmètres à l'hyperviseur, même si finalement ce n'est la méthodologie que nous employons).

```
struct domain
{
    domid_t          domain_id;

    unsigned int     max_vcpus;
    unsigned int     vtf_pids[25];
    bool     is_vtf;
```

- Modification du comportement de l'hypercall (dans les fichiers ***xen/common/domctl.c & xen/arch/x86/domctl.c*** resp.) : nous allons rajouter, dans la définition de l'hypercall, un *case*; au cas où il s'agit d'une commande vtf, ***case XEN_DOMCTL_vtf***, nous vérifions bien que la VM en qui exécute la commande n'est pas le dom0 (***domain !=0***) et si tel est le cas nous effectuons le traitement approprié en donnant à la VM les droits nécessaires.

```
long arch_do_domctl(
    struct xen_domctl *domctl, struct domain *d,
    XEN_GUEST_HANDLE_PARAM(xen_domctl_t) u_domctl)
{
    struct vcpu *curr = current;
    struct domain *currd = curr->domain;
    long ret = 0;
    bool copyback = false;
    unsigned long i;

    switch ( domctl->cmd )
    {

    case XEN_DOMCTL_vtf:
    case XEN_DOMCTL_shadow_op:
        ret = paging_domctl(d, &domctl->u.shadow_op, u_domctl, 0);
        if ( ret == -ERESTART ){
            if(!currd->is_vtf)
                return hypercall_create_continuation(__HYPERVISOR_arch_1,
                                                     "h", u_domctl);
        }
        copyback = true;
        break;
```

```
long do_domctl(XEN_GUEST_HANDLE_PARAM(xen_domctl_t) u_domctl)
{
    long ret = 0; int i=0;
    bool_t copyback = 0;
    struct xen_domctl curop, *op = &curop;
    struct domain *d = NULL;

    if ( copy_from_guest(op, u_domctl, 1) )
        return -EFAULT;

    if ( op->interface_version != XEN_DOMCTL_INTERFACE_VERSION )
        return -EACCES;

    switch (op->cmd)
    {
    case XEN_DOMCTL_vtf:
        op->domain = current->domain->domain_id;

        if (op->domain != 0)
        {
            d = rcu_lock_domain_by_id(op->domain);
            if (!d)
                return -ESRCH;

            d->is_vtf = 1;
            memcpy(d->vtf_pids, op->u.vtf.pids, 25*sizeof(op->u.vtf.pids[0]));

            ret = arch_do_domctl(op, d, u_domctl);
        }

        goto domctl_out_unlock_domonly;
        break;
    }
```

ii. Hypercall do_sysctl

Nous utiliserons cet hypercall principalement pour l'option --list (qui doit retourner à la VM la liste des fonctionnalités auxquelles l'hyperviseur lui donne accès).

- Modification de la structure de données ***xen_sysctl*** (dans le fichier ***xen/include/public/sysctl.h***) : comme pour la structure xen_domctl, nous allons rajouter des champs relatifs aux commandes xl.

```
#define XEN_SYSCTL_livepatch_op                  27
#define XEN_SYSCTL_set_parameter                 28
#define XEN_SYSCTL_vtf                           29
```

```
    struct xen_sysctl_set_parameter     set_parameter;
    struct xen_sysctl_vtf               vtf;
    uint8_t                             pad[128];
} u;
```

```
struct xen_sysctl_vtf_getcpuinfo {
    uint32_t ept:1;
    uint32_t pml:1;
};
typedef struct xen_sysctl_vtf_getcpuinfo xen_sysctl_vtf_getcpuinfo_t;
DEFINE_XEN_GUEST_HANDLE(xen_sysctl_vtf_getcpuinfo_t);

struct xen_sysctl_vtf_action {
    uint32_t action;
    #define XEN_SYSCTL_vtf_action_list       1
    #define XEN_SYSCTL_vtf_action_help       2
    uint32_t pid;
};
typedef struct xen_sysctl_vtf_action xen_sysctl_vtf_action_t;
DEFINE_XEN_GUEST_HANDLE(xen_sysctl_vtf_action_t);

struct xen_sysctl_vtf {
    boolean_t check;
    xen_sysctl_vtf_action_t cmd;
    XEN_GUEST_HANDLE_64(xen_sysctl_vtf_getcpuinfo_t) buffer;
};
```

- Modification du comportement de l'hypercall (dans les fichiers ***xen/common/domctl.c & xen/arch/x86/hvm/vmx/vmcs.c*** resp.): nous allons également ajouter dans la définition de l'hypercall ***do_sysctl***, un cas particulier pour une opération de type ***vtf***. Nous devons faire ceci avant l'opération de contrôle qu'effectue l'hyperviseur sur la nature du domaine :

```
ret = xsm_sysctl(XSM_PRIV, op->cmd);
if ( ret )
    return ret;
```

xsm signifie ***xen security modules***. Dans cette portion de code, l'hyperviseur vérifie bien que la VM à l'origine de la commande a les privilèges nécessaires pour

l’exécuter sinon une erreur est générée. C’est la raison pour laquelle nous devons intervenir (rajouter notre portion de code) avant l’hyperviseur! (voir les figures suivantes respectivement dans ***xen/common/domctl.c*** (les 2 premières) et ***xen/arch/x86/hvm/vmx/vmcs.c*** (la dernière).

```
long do_sysctl(XEN_GUEST_HANDLE_PARAM(xen_sysctl_t) u_sysctl)
{
    long ret = 0;
    int copyback = -1, rc=0;
    struct xen_sysctl curop, *op = &curop;
    static DEFINE_SPINLOCK(sysctl_lock);

    if ( copy_from_guest(op, u_sysctl, 1) )
        return -EFAULT;

    if ( op->interface_version != XEN_SYSCTL_INTERFACE_VERSION )
        return -EACCES;

    /*vtf vtf*/
    switch ( op->cmd )
    {
        case XEN_SYSCTL_vtf:
            rc=1;
            ret = do_vtfctl(op);
            break;

        case XEN_SYSCTL_physinfo:
            if(op->u.vtf.check) {
                ret = get_physinfo(u_sysctl, op);
                rc = 1;
            }
            break;

        default:
            break;
    }

    if( rc )
        return ret;
    /***/
```

```
long do_vtfctl(struct xen_sysctl *op)
{
    long ret = 0;
    struct xen_sysctl_vtf_getcpuinfo info;
    static DEFINE_SPINLOCK(sysctl_lock);

    switch (op->u.vtf.cmd.action)
    {
    case XEN_SYSCTL_vtf_action_list:
        get_vtf_cpuinfo(&info);
        if ( copy_to_guest_offset(op->u.vtf.buffer,
                                  0, &info, 1) )
            {
                ret = -EFAULT;
                break;
            }
        break;

    default:
        break;
    }

    spin_unlock(&sysctl_lock);
    return ret;
}
```

```
void get_vtf_cpuinfo(struct xen_sysctl_vtf_getcpuinfo *info){
    if(cpu_has_vmx_pml)
        info->pml = 1;
    if(cpu_has_vmx_ept)
        info->ept = 1;
}
```

c. Exemples d’appels

Il faut tout d’abord installer les tools que l’on vient de modifier dans la VM. Pour cela suivre les étapes suivantes à l’intérieur de la VM :

- Installer les dépendances : **sudo apt install libc6-dev libglib2.0-dev libyajl-dev yajl-tools libbz2-dev bison flex zlib1g-dev git-core texinfo debhelper debconf-utils debootstrap fakeroot gcc make binutils liblz-dev python-dev libncurses-dev libcurl4-openssl-dev libx11-dev uuid-dev libyajl-dev libaio-dev libglib2.0-dev pkg-config bridge-utils iproute udev bison flex gettext bin86 bcc iasl markdown git gcc-multilib texinfo libperl-dev libgtk2.0-dev liblzma-dev**

- Copier dans la VM le dossier xen/ avec lequel vous travaillez et que vous avez modifé. Vous pourrez trouver un code fonctionnel sur le dépôt git suivant : **git clone https://github.com/bstellaceleste/PML-Xen10**
- Se placer dans le dossier tools et taper les commandes suivantes :
  - **cd tools**
  - **sudo rm -r qemu-xen-dir/ qemu-xen-traditional-dir/ qemu-xen-build/**
  - **sudo ./configure**
  - **sudo make clean**
  - **sudo make**
  - **sudo make install**
  - **sudo ldconfig**
  - **sudo /etc/init.d/xencommons start**

Après l'on peut exécuter les différentes commandes xl vtf qu'on vient d'implémenter :

```
bitchebe@ubuntu:~/PML-Xen10/tools$ sudo xl vtf --help
Usage: xl [-v] vtf [options] [VMX] [process_pid]

Informations about VTF (Virtual Technology Extensions).

Options:

-l, --list              List VMX supported by the vtf library
-e, --enable            Enable a vtf for this domain
-d, --disable           Disable a vtf for this domain
-h  --help              Print this help.
```

```
bitchebe@ubuntu:~/PML-Xen10/tools$ sudo xl vtf --list
VMX: Supported advanced features:
   - PML: Page Modification Logging.
   - EPT: Extended Page Table.
```

On peut aussi vérifier que pour toute autre commande xl le domU n'a pas les droits :

```
bitchebe@ubuntu:~/PML-Xen10/tools$ sudo xl list
libxl: error: libxl.c:367:libxl_get_physinfo: getting physinfo: Operation not permitted
libxl: error: libxl_domain.c:315:libxl_list_domain: getting domain info list: Operation not permitted
libxl_list_domain failed.
```