Examen de virtualisation – 2/2/2021

NOTE : correction étudiante, à prendre avec des pincettes !

Question de cours

Q1 – L'intérêt majeur de SMEP est d'augmenter la difficulté ("raise the bar") nécessaire pour effectuer une exécution de code utilisateur dans le noyau. Un buffer overflow ne permettra pas d'exécuter directement du code noyau, il faudra ROP-er pour désactiver SMEP et sauter à l'adresse souhaitée.

Q2 – La Nested Virtualization permet d'exécuter un hôte invité accéléré matériellement dans un environnement qui est lui même invité. Dis plus clairement, les VMs peuvent exécuter des Vms avec un overhead faible.

Q3 – Les conteneurs Linux (docker, liblxc, etc) partagent le noyau de l'hôte, et un mauvais paramètre de sécurité ou un défaut dans le noyau impactera tous les conteneurs. Plus grave, un conteneur peut réciproquement impacter l'hôte de la même manière. L'isolation n'est pas garantie par le matériel, et repose uniquement sur l'isolation logicielle du noyau.

Q4 – Xen et KVM utilisent QEMU, notamment pour émuler des pilotes de périphériques : e1000, QXL, contrôleur USB virtuel… et également dans certains pour fournir une interface de para-virtualisation: virtioblk, virtioscsi, virtio-balloon, etc.

Q5 – Un TPM (Trusted Platform Module) peut aider à la mise en place d'un secure boot, en contenant les clés permettant de valider de l'authenticité du module suivant dans la chaîne de boot : bootloader et/ou OS suivant les cas, voire modules UEFI suivant l'implémentation.

Q6 – Les Nested Page Tables (Intel EPT/AMD NPT) permettent de gérer finement l'emplacement physique des pages d'une machine virtuelle, en donnant l'illusion au système d'exploitation invité qu'il a accès à tout l'espace d'adressage physique de la machine, et cela sans devoir payer le prix des Shadow Pages, puisque la traduction est effectuée matériellement et ne nécessite pas un travail aussi important de l'hyperviseur pour intercepter les éditions des tables de page et copier/mettre à jour les tables réelles.

Q7 – L'IOMMU (I/O Memory Management Unit) permet de se protéger des attaques DMA en restreignant les plages d'adresses accessibles à un périphérique. Pour cela, il « donne » au noyau le contrôle sur ce qu'il choisit d'exposer aux périphériques.

Q8 – Un malware peut tenter de détecter qu'il s'exécute dans une VM de nombreuses façons, mais le plus simple est sans doute d'énumérer les périphériques USB et PCI(e) afin détecter la présence de composants émulés/para-virtualisés.

Q9 – Un noyau d'OS para-virtualisé est un OS qui a connaissance du fait qu'il s'exécute dans un hyperviseur, et qui collabore activement avec celui-ci, en utilisant des interfaces exposées par l'hyperviseur et qu'il peut appeler par le biais d'hypercalls. Cela permet bien souvent de payer un coup bien moindre par rapport à l'émulation d'un périphérique (pour la gestion des interruptions par exemple) , puisque ces appels permettent d'avoir à intercepter moins d'instructions, faisant donc moins de vmexit/vmenter (appels plutôt couteux en temps et en énergie), et calquent souvent plus aux opérations nécessaires du côté de l'hyperviseur, évitant ainsi d'émuler une interface bien souvent obsolète.

QA – La virtualisation matérielle est généralement plus performante puisqu'on bénéficie de tous avantages (et aussi des inconvénients, cf. Meltdown/Spectre:/) de l'exécution native d'un programme : exécution spéculative, out-of-order, décodage matériel des instructions, « sécurité » assurée par le matériel (pas besoin de patcher le code utilisateur à la volée pour intercepter les instructions problématiques), pas de transcriptions d'assembleur à effectuer (ce qui est principalement une perte de cycles CPU sur la même architecture)…

QB – Un systemcall consiste à déclencher une interruption (ou à appeler l'instruction syscall si les MSRs dédiés sont correctement configurés par le système d'exploitation) pour demander au noyau d'effectuer une tâche (ouvrir un fichier par exemple). Un hypercall est le même mécanisme, sauf que le client est ici le noyau invité, et que le serveur est l'hyperviseur. Pour cela, l'hypercall va provoquer un vmexit, l'hyperviseur va traiter la demande en fonction du contexte (en lisant la mémoire du processus, ses registres depuis le contexte et en désassemblant si besoin le code de l'invité), préparer le contexte et rendre la main à l'invité avec un vmenter.
QC – Les composants les plus vulnérables d'un hyperviseur sont probablement
1) le code en charge de traiter les hypercalls/exceptions (à l'instar du code en charge du traitement des appels système dans un noyau) parce qu'il s'agit de l'interface exposée aux invités.
2) les périphériques virtualisés parce qu'il s'agit de beaucoup de code qui s'exécute avec des privilèges élevés sur l'hôte (ou l'équivalent du dom0 Xen).

Problème A
PAQ1 – L'émulation logicielle a lieu soit dans les services noyaux (étape 4), qui s'exécute dans le ring0 du système d'exploitation principal (le « dom0 »), soit dans l'espace utilisateur de ce même système (on peut présumer qu'il s'agit du ring3), en fonction du « propriétaire » du fait qu'un périphérique physique soit associé au périphérique virtuel demandé par le système d'exploitation invité (auquel cas l'exécution se fait dans le noyau) ou que le périphérique est émulé (l'exécution se fait dans en espace utilisateur.
PAQ2 – Les composants critiques de cette architecture sont l'hyperviseur (évidément), le dispatcher IO (un échec de ce composant DOS le système) et les modules noyaux (une faille à cet endroit peut amener à une compromission complète du système, puisqu'il manipule des périphériques physiques. En cas d'échec dans un composant en espace utilisateur, on peut imaginer redémarrer ce composant ou tuer l'invité correspondant, mais sans compromettre tout le système (en supposant que les périphériques émulés ne sont pas partagés entre les différentes machines virtuelles).
PAQ3 – Un attaquant qui contrôle l'OS invité peut tenter de fuzzer l'hyperviseur pour trouver une faille potentielle (crash de l'hyperviseur, fuite d'information).
PAQ4 – Un crash du « kernel BE Service 1 » peut éventuellement aboutir à l'arrêt de toute la machine, suivant le service en question, et également la résistance du noyau au crash d'un de ses modules (Linux supporte bien mieux un crash dans un driver FAT32 que dans le scheduler par exemple, parce qu'il peut quasiment « ignorer » le premier en tuant la tâche associée).

Problème B
PBQ1 – La vulnérabilité se situe dans le gestionnaire de mémoire virtuelle de la VM paravirtualisée (ici, l'attaquant), en particulier la gestion des tables de pages de dernier niveau. On suppose donc que Alice peut écrire des données de son choix dans les tables de pages de son choix (worst-case scenario). On part du principe que EPT n'est pas utilisé (il n'y a ici apparemment aucune raison de l'utiliser, puisque cela risquerait d'augmenter le nombre de pages faults et réduirait automatiquement l'espace libre dans le TLB, puisque cela augmente le nombre d'entrées de TLB requises pour faire la traduction des adresses de l'invité : ~15 au lieu de ~4 sur x86_64). Si tel est le cas, l'attaquant a donc un accès en écriture arbitraire sur les tables de pages de l'hôte. C'est bien une faille très critique.
PBQ2 – Alice peut lancer une attaque pour contrôler les adresses de l'hôte, mais il lui faut encore obtenir la valeur du CR3 de la VM de Bob pour accéder plus aisément à la mémoire virtuelle de son navigateur et lire les données du processus associé à Chrome.
PBQ3 - Alice pourrait mapper les pages de l'hôte sur son propre espace, et essayer de localiser le noyau. Puis trouver le module qui gère l'hyperviseur, et obtenir la position du context de la VM de Bob. Elle peut ensuite récupérer la liste des processus en parsant la mémoire du noyau, déterminer le PID associé, attendre que ce processus passe en « Running », récupérer la valeur de CR3 depuis le

contexte de Bob. En bricolant un peu, elle peut aussi obtenir le mapping [VM de Bob]→[mémoire physique]. Elle peut alors faire mapper la mémoire de chrome sur elle-même, lire cette mémoire et Jackpot ! (Bon, je n'aimerais pas être cet attaquant, ça fait quand même énormément de boulot pour des coordonnées bancaires :p)
PBQ4 – Tout dépend de la motivation d'Alice. Dans l'absolu, le risque n'est pas énorme, mais si Alice a déjà identifié pareille faille, on peut supposer qu'elle sera motivée à l'exploiter. Je pense aussi qu'il vaut mieux surestimer les risques que les sous-estimer. De plus, un professeur m'a dit une fois "il faut un peu roter du sang pour apprécier l'hydromel", et Alice partage peut-être cet avis !