Mise en Cohérence des Objectifs de TIPE (MCOT)
Attaques par dépassement de tampon

Tristan GIERCZAK–GALLE, MPI*, 2025 – 2026

Positionnement thématique

Informatique, Architecture

Mots-clefs

Mots-clefs – Architecture Hardware/Software – Virtualisation Mémoire – Tampon – Allocation – Exploit

Keywords – Hardware/Software Architecture – Memory Virtualisation – Buffer – Allocation – Exploit

Bibliographie commentée

Dès l'émergence de langages impératifs (C, C++), ou de l'utilisation humaine d'Assembleur (pour ceux qui s'y sont vaillament attelés), offrant à l'utilisateur un contrôle total, parfois mal utilisé, de la mémoire, apparaît une vulnérabilité à pléthore d'exploits informatiques : le dépassement de tampon.

Grossièrement, un dépassement de tampon désigne un bug par lequel un processus, lors de l'écriture dans un tampon (i.e. un aggrégat de mémoire, précédemment alloué par l'utilisateur et **supposé** comme valide à l'instant d'utilisation), écrit à l'extérieur de l'espace alloué au tampon, écrasant ainsi des informations nécessaires au processus.

En réalité, cette attaque ne peut exister qu'en supposant une gestion maladroite et incomplète des ressources au sein d'un processus, qui inclut des manipulations manuelles de la mémoire. Autrement dit, la source se situe tant dans une compréhension faussée ou incomplète des opérations de la mémoire faites par l'utilisateur,

que dans des fonctionnalités dangereuses et normalement inutilisables, mais longtemps soutenues par les standards POSIX / ISO C (*strcpy* est l'exemple bien connu des vulnérabilités que renferme la librairie standard C (stdlibC)). Malgré tout, il est très difficile de garantir l'absence de fuites de mémoire et de maladresses lors de l'interaction avec les ressources, dans des langages autant déphasés des moyens d'attaque actuels, qui se basent notamment sur une compréhension très fine du problème et de son potentiel.

Depuis le début des années 2000, ces failles de sécurité ont été fréquemment exploitées, notamment dans OpenSSH.

Que cela concerne des allocations ou libérations, ou bien des écritures ou lectures, un agrégat non libéré ou l'utilisation de fonctionnalités anciennes potentiellement dangereuses – du langage C pour le meilleur exemple – peuvent conduire à toutes sortes d'abus de mémoire. En effet, cet exploit se décline en fait en de multiples catégories, qui traitent entre autres des différentes structures de la mémoire (vive, ici), touchées par l'attaque (par exemple, il est commun de traiter de *buffer overflow*, classe elle-même parent de *stack buffer overflow*, qui spécialise le problème à la pile, et de manière analogue, *heap buffer overflow*, affectant le tas). Toutes ces classes restent en réalité peu fines, car chaque cas se distingue des autres dans le but-même de l'attaque, et de la spécificité de l'architecture sur laquelle elle opère.

La découverte de ce potentiel lors de l'apparition de tels langages a très vite motivé des pratiques plus strictes concernant la gestion de la mémoire dans les langages où cela est problématique (C), mais également dans l'élaboration de nouveaux langages, se distinguant notamment par un nouveau paradigme de gestion de la mémoire. Lors de la popularisation du concept de Garbage Collector, de nombreux langages comme *Java, C#, D*, ont motivé le recyclage automatique de la mémoire, ce qui constitue un réel pas en avant, loin du fonctionnement de C.

Malgré les techniques mises en oeuvre pour prévenir ces attaques (dont le TIPE ne fait pas l'objet), la virtualisation de la mémoire, aujourd'hui présente dans *quasi* toutes les architectures mémoire du marché des ordinateurs, n'est pas très efficace. Malgré tout, conformément aux travaux de David A. Patterson [1] et Remzi Arpaci-Dusseau, le TIPE se base en partie sur le principe de pagination, qui se rapproche du cas réel.

Problématique retenue

Dans la plupart des attaques de grande envergure, généralement dirigées contre des organisations vitales (compagnies de transports, hôpitaux, distribution d'énergie), le dépassement de tampon est un élément source. En un sens, la fréquence de problèmes liés à la gestion de la mémoire, même au sein de codebases populaires,

montre bien à quel point cette attaque est généralement favorisée. C'est pourquoi il me semble intéressant d'explorer les possibilités de ce modèle d'attaque dans une approche défensive, et de montrer le danger qu'il implique d'un point de vue théorique.

Objectifs du TIPE

1. Modélisation : après quelques rappels théoriques sur la pagination et les recoins de C propices à cette attaque, je simulerai visuellement, par l'intermédiaire d'un simulateur d'allocation RAM construit par mes soins, plusieurs attaques par dépassement de tampon.
2. Apports conceptuels : L'observation de telles attaques implique (et nécessite) une compréhension minimale de la virtualisation mémoire et une mise en contexte obligatoire du modèle que j'utilise, qui mêle pagination et segmentation.

Références

[1] David A. Patterson : *Computer Architecture : A Quantitative Approach*. Lol, 2018.