Projet Nixwen : scanner mémoire
Cahier des charges, planning, et répartition des tâches

AMOSSÉ Adrien LA SELVE Patrick MALLARD Enzo MONNET Alexis
TAUNAY Rémi

Projet S3/S4, année 2016–2017
Encadré par : M. Faucou

1 Cahier des charges

1.1 Présentation générale du problème

1.1.1 Projet

Nous devons créer un programme permettant d'explorer la mémoire et de la modifier (injection de code et modification de variables). Ce logiciel devra fonctionner sous système GNU/Linux.

1.1.2 Contexte

Lorsque l'on vient à parler de scanner mémoire, une application viens rapidement à l'esprit de certain. En effet, Cheat Engine est un outil permettant d'effectuer l'ensemble des opérations que l'on peut attendre de ce type d'outil : observation de la mémoire, modifications, recherche de structures... Cet outil open source qui fonctionne sous Windows n'est ni plus ni moins qu'un ensemble d'outils visant à modifier les comportements d'une application ou de la ou déboguer. Ainsi, lors du développement de notre propre scanner mémoire, il sera pertinent de garder à l'esprit les différentes fonctionnalités que propose Cheat Engine.

Dans le cadre des enseignements nous ayant été dispensés, un module a été consacré à l'apprentissage du fonctionnement et à l'inspection mémoire. Cela inclut notamment l'usage d'un débogueur tel que *gdb*. De ce fait ce projet peut être vu comme une continuité de ce module, où il va venir renforcer et développer davantage nos compétences sur les aspects bas-niveaux de linux en programmation (processus, fonctionnement de la mémoire).

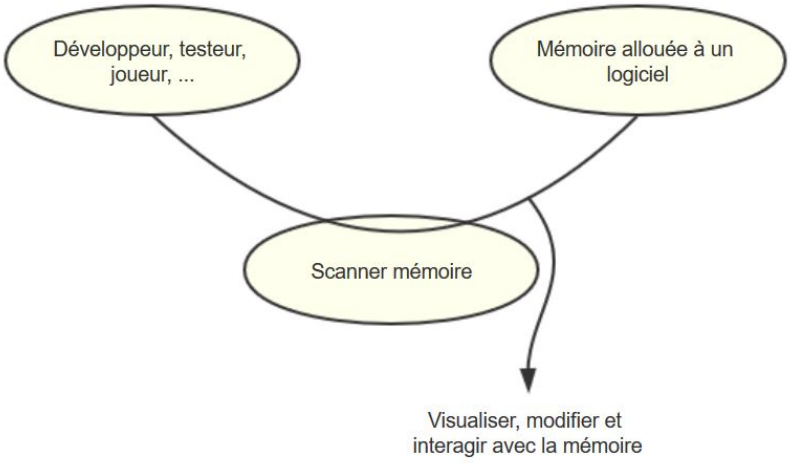
Postérieurement, ce projet pourra être poursuivi en proposant des portages sur d'autres plateformes que GNU/Linux telles que Windows ou Android.

Il nous est demandé de réaliser d'une part le scanner mémoire (le programme en lui-même) ; et d'autre part un wiki (ensemble de documents détaillant nos recherches sur le sujet et sa périphérie).

Ce projet est à réaliser dans le cadre des projets tutorés inhérents au DUT Informatique pour les semestres 3 et 4.

1.1.3 Énoncé du besoin

FIGURE 1 – Bète à corne

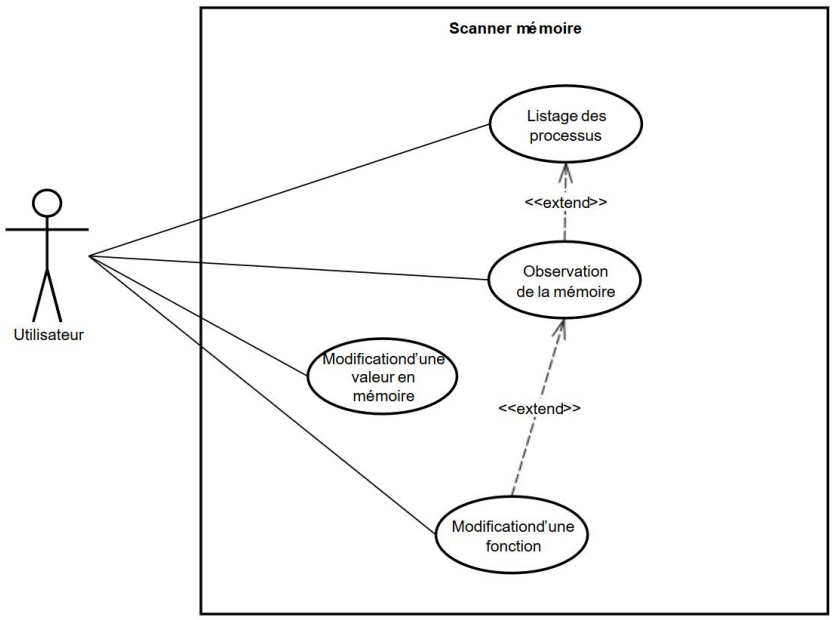


1.1.4 Environnement du produit recherché

Ce projet va être réalisé par les étudiants listés ci-après : LA SELVE Patrick MALLARD Enzo, MONNET Alexis, AMOSSÉ Adrien et TAUNAY Rémi. Le développement utilisera un environnement constitué d'outils collaboratifs tels que git (utilisé conjointement avec github), de CLion (IDE de JetBrains).

1.2 Expression fonctionnelle du besoin

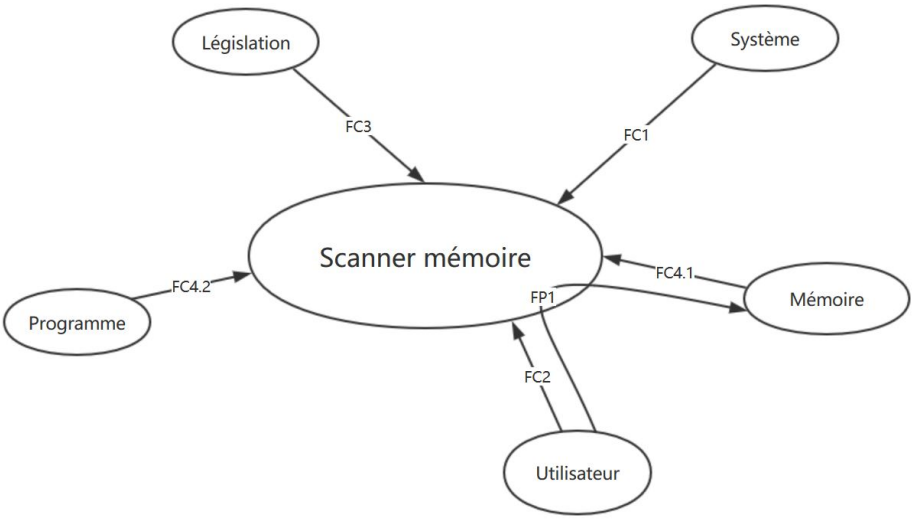
FIGURE 2 – Cas d'utilisation



Fonction	Importance	Critère d'appréciation	Flexibilité
Listage des processus	4	durée d'exécution : 0.5s	0.25s
Observation de la mémoire	5	durée d'exécution : 0.5s	0.25s
Modification d'une valeur en mémoire	5	durée d'exécution : 0.75s	0.15s
Modification d'une fonction	3	durée d'exécution : 1s	0.5s

1.2.1 Fonctions de service et de contrainte

FIGURE 3 – Diagramme pieuvre – Fonctions de services



	Fonctions	Critère	Niveau
FP1	Accéder et modifier la mémoire		
FC1	Fonctionner sous GNU/Linux	Version noyau	>2.4.6
FC2	Avoir une interface utilisateur	Type	TUI au minimum
FC3	Respecter la législation en vigueur	CGU des logiciels, lois, ...	
FC4.1	Pouvoir accéder à la partie mémoire concerné par la programme		
FC4.2	Avoir les droit nécessaires au déboguage du programme	UID	identique au lanceur du programme

Au premier abord, les deux fonctions strictement nécessaires à l'utilisation du scanner mémoire sont :
— Observation de la mémoire
— Modification d'une valeur en mémoire
En effet, le besoin premier se décompose en deux services principaux que sont l'observation (une vue agrémentée d'informations telle que la correspondance adresse / valeur) ; et la modification (l'application ne doit pas être uniquement contemplative).

Deux autres fonctions viennent compléter et améliorer les services principaux, à savoir :
— Listage des processus
— Modification d'une fonction
Lister les processus peut permettre une observation plus poussée telle qu'une vue sélective vis-à-vis d'un processus en particulier. De ce fait, le listage des processus vient faciliter l'observation de la mémoire (comme par exemple permettre la recherche d'une variable au sein de la mémoire d'un processus choisi).

Permettre la modification d'une fonction peut soulever des difficultés techniques (exemple : le code de remplacement doit venir remplacer exactement l'emplacement mémoire du code d'origine). Sa réalisation n'est de ce fait pas prioritaire, et s'effectuera donc probablement en dernier lieu.

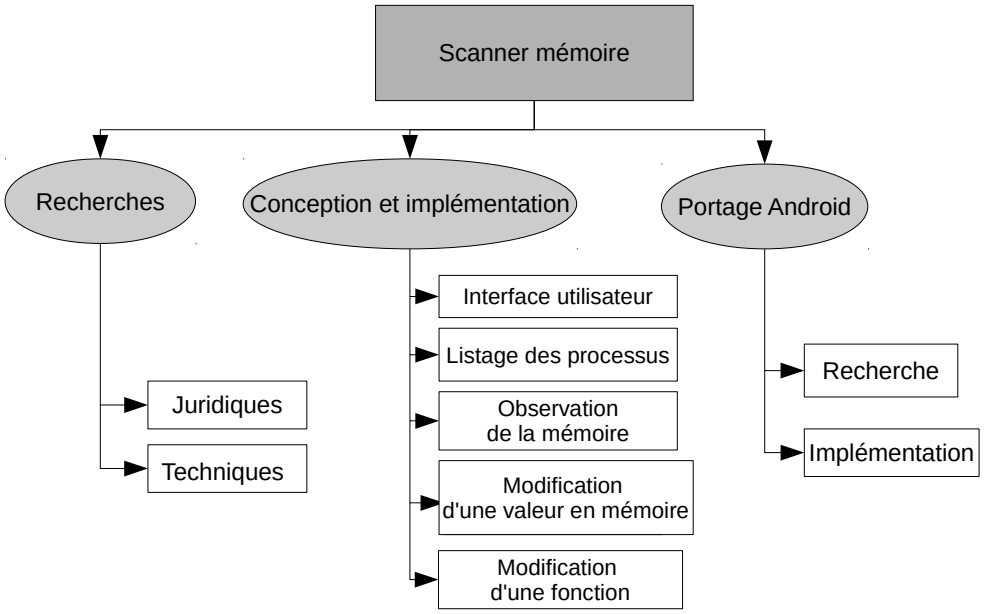
1.2.2 Critères d'appréciation

Les critères d'appréciation ne dépendent ici que d'un paramètre : le temps. On attend une certaine réactivité de la part de l'application, les traitements demandés ne demandant que peu de ressources système. C'est pourquoi on se concentrera sur les durées d'exécution pour chacune des opérations demandées. En effet la latence lors de l'utilisation de l'application doit être telle que l'on ne ressente pas une quelconque lenteur injustifiée lors des différents traitements.

1.2.3 Niveaux des critères d'appréciation et ce qui les caractérise

Le temps d'exécution est ici le seul critère d'appréciation. En effet, un temps d'exécution trop important entrave l'utilisation d'un logiciel, et ce quelque soit sa finalité. Il est donc nécessaire que chaque fonction s'opère en un minimum de temps, de manière à conserver une utilisation fluide et commode du scanner mémoire. Les temps indiqués pour chaque fonction semblent dérisoires, cependant il faut garder à l'esprit les différents appels qui pourront avoir lieu entre les différentes fonctions c'est pourquoi le temps nécessaire de traitement aux yeux de l'utilisateur sera plus important. Ainsi on pourra tenter d'optimiser au mieux les différentes fonctions afin d'obtenir un temps d'exécution qui reste convenable au vu de l'action demandée.

1.3 Cadre de réponse



1.3.1 Descriptif approfondi de chaque fonction

Observation de la mémoire Cette fonction se présentera en interface dite textuelle, appelée aussi TUI pour Text User Interface, afin de permettre une lecture claire de la mémoire. Son rôle est de donner une correspondance entre chaque adresse mémoire et la valeur qui lui est associée. Elle devra permettre une navigation aisée dans la vue (défilement, saut à une adresse donnée).

Modification d'une valeur en mémoire Cette fonction a pour but d'aller injecter la valeur de son choix à un emplacement mémoire donné. Le choix de l'adresse incombe donc à l'utilisateur. La modification est généralement voulue suite à une observation de la mémoire, lors de laquelle l'utilisateur a été repérer l'adresse de la variable à même d'être modifié par ses soins.

Listage des processus Comme son nom l'indique, la fonction doit permettre de connaître les différents processus en cours d'exécution sur la machine. Il peut-être ajouté à cette fonction la possibilité de choisir un processus en particulier afin d'obtenir de plus amples d'informations à son égard : PID, éventuels threads, etc...

Modification d'une fonction Cette fonctionnalité doit permettre, dans sa finalité, de modifier le comportement même du programme, sans avoir uniquement recours à des changements de valeurs, mais bel et bien à l'altération du processus de traitement de ces dernières. Ainsi, cette fonctionnalité doit permettre de localiser une fonction au sein du tas afin de la modifier en injectant son propre code, tout en respectant un certain nombre de vérifications préalables. En effet, sa taille doit rester inchangée par rapport à sa taille nominale. De plus, d'autres facteurs seront à prendre ne considération tels que le respect du type de retour. La plage de ces vérifications peut par la suite être étendue à souhait (validité du code en s'assurant que les parenthèses sont correctement refermées, par exemple).

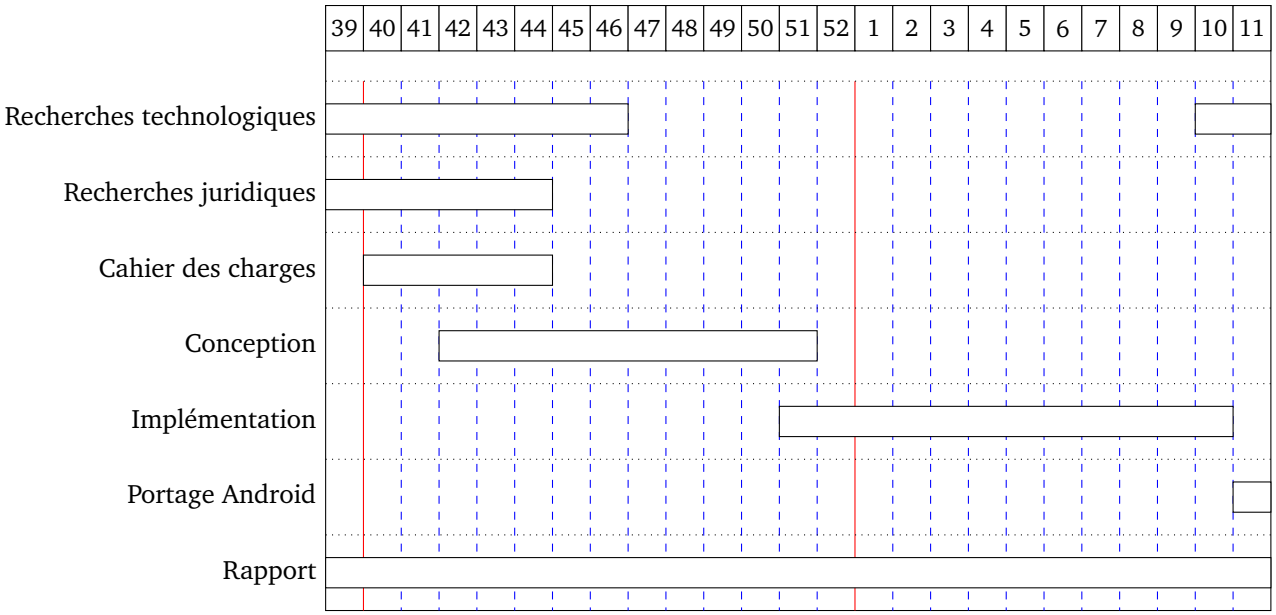
1.3.2 Ensemble du produit

La réalisation d'une interface graphique est consommatrice de temps. De plus, l'utilisation basique d'un scanner mémoire n'impose pas sa réalisation. Enfin, ce n'est pas le cœur du sujet. L'amoncellement de ces arguments font que cette option n'est pas retenue au cahier des charges.

Il peut s'avérer judicieux d'effectuer une première décomposition qui permette de séparer la gestion des interactions avec l'utilisateur (traitement des commandes) via un ensemble de fonctions dédiées, lesquelles viendront appeler lorsque demandé les fonctions correspondantes, avant d'en reformater correctement la sortie au moment de l'affichage.

Une nouvelle fois, en dehors de la probabilité sur d'autres systèmes d'exploitation, l'application pourra subir des évolutions qui s'inspireront de Cheat Engine, ou d'un logiciel semblable. En effet, les fonctionnalités proposées par Cheat Engine sont pertinentes, il est de ce fait possible de se baser dessus afin de faire évoluer notre propre application. Bien entendu, si des idées extérieures permettraient de venir enrichir l'éventail de possibilité il faudrait également les considérer, en les jugeant selon leur utilité vis-à-vis de l'utilisateur final, ainsi que leur faisabilité technique.

2 Planning et répartition des tâches



2.0.1 Tâches

Tâche 1 : Recherches techniques (sécurité, droits d'accès)
Durée estimée : 9 semaines — Date de début : début S39
Responsable : Rémi — Autres participants : Enzo
Résumé : Répondre à des questions de l'ordre : Comment fonctionne la sécurité de manière générale sous GNU/Linux ? Comment obtenir les droits de débogage sur un logiciel donné ?
Comment */proc* est-il organisé ?
Eventuellement en fin de semestre 3, recherches complémentaires sur les autres OS (Android, Windows)

Tâche 2 : Wiki complet au sujet des aspects juridiques
Durée estimée : 6 semaines — Date de début : début S39
Responsable : Alexis — Autres participants : Adrien
Résumé : Condensé d'informations clair sur l'ensemble des aspects juridiques périphériques aux scanners mémoire.

Tâche 3 : Rédaction du rapport
Durée estimée : totalité du projet — Date de début : début S39
Responsable : Patrick — Autres participants : Adrien, Alexis, Enzo, Rémi
Résumé : Compte-rendu du déroulement du projet. Regroupe l'ensemble des informations autres que celles d'ores-et-déjà contenues au sein du cahier des charges.

Tâche 4 : Conception
Durée estimée : 10 semaines Date de début : début S41
Responsable : Enzo :3 Autres participants :
Résumé : Réalisation d'un ensemble de schémas et d'explications attenantes permettant de structurer l'application. Mise en place du pseudo code (avec appels systèmes) et choix du langage d'implémentation

Tâche 5 : Implémentation
Durée estimée : 12 semaines Date de début : début S51
Responsable : Adrien Autres participants : Reste de l'équipe
Résumé : Implémentation du pseudocode en C ou C++ et mise en place de l'interface H/M.