Demonstração de Técnicas de Manipulação de Memória de Aplicações em Execução

Lucas Baltazar Cayres[1], Luiz Carlos Querino Filho[2]

[1]Fatec – Faculdade de Tecnologia de Garça
Garça – SP

{lucas.cayres,luiz.querino}@fatec.sp.gov.br

Abstract. *This paper presents an application for handling memory of a running process. Currently, a large number of people have Windows operating system that provides several resources for its developers, some of these low-level resources can be used to intercept and/or modify information of running processes.*

Resumo. *Este artigo apresenta uma aplicação para manipulação da memória de um processo em execução. Atualmente, um grande número de pessoas possui o sistema operacional Windows, que por sua vez disponibiliza diversos recursos para seus desenvolvedores, alguns desses recursos de baixo nível podem ser utilizados para interceptar e/ou modificar informações de processos em execução.*

1. Introdução, Justificativa e Objetivo

Nos dias contemporâneos é imprescindível o total cuidado com o sigilo das informações. Com a disseminação da internet nos mais variados locais do mundo é possível ter contato com informações praticamente em tempo real. Tal fato traz grande poder de comunicação, materiais para estudo, tutoriais diversos, contato com vídeos, imagens entre outros. Estes itens são acessíveis por qualquer pessoa, ou seja, simplesmente estão disponíveis na internet.

Com um vasto conhecimento espalhado pela rede mundial de computadores é possível aprender de maneira autodidata inúmeras técnicas e métodos avançados, incluindo conceitos de tecnologia da informação específicos para desenvolvedores de *software*. Utilizar esses conhecimentos para obter vantagens ilícitas é uma realidade. Quebra de segurança, domínio sobre uma aplicação ou até mesmo modificação da informação sem possuir acesso ao código fonte de um projeto são exemplos de possíveis manipulações tendo um processo em execução como alvo.

O objetivo do aplicativo apresentado neste trabalho é utilizar alguns recursos fornecidos pelo sistema operacional Windows para alterar dados de uma aplicação carregada na memória principal do computador, ou seja, alterar os dados de uma aplicação em execução sem possuir os privilégios necessários.

As técnicas aplicadas sobre o programa alvo demonstrarão potenciais falhas e brechas de segurança que fornecem ao praticante privilégios necessários para alterar ou reescrever rotinas e funções da aplicação alvo. Gerenciar integralmente o espaço virtual de memória alocado para a aplicação é a falha de segurança principal e comprometedora

que fornece ao praticante um leque de possibilidades para executar ataques ou interceptar informações do alvo.

2. Metodologia Utilizada

A ferramenta apresentada neste artigo é completamente desenvolvida na linguagem C através do ambiente de desenvolvimento Dev-C++. A linguagem C fornece ao desenvolvedor a possibilidade de acessar não somente as funções nativas da linguagem, mas também funções do próprio sistema operacional.

Será utilizado o jogo *Secret Maryo Chronicles* para fazer o papel de aplicação alvo, distribuído através do site http://www.*secretmaryo*.org/. Trata-se de um jogo baseado nas quatro liberdades de software livre[1]:

1. A liberdade de executar o programa como desejar e para qualquer propósito.
2. A liberdade de estudar como o programa funciona, e adaptá-lo às próprias necessidades.
3. A liberdade de redistribuir cópias de modo que ajude o próximo.
4. A liberdade de distribuir cópias modificadas ao próximo.

Em conjunto é utilizada a ferramenta Cheat Engine[2], um *scanner* de memória[3]. Essa ferramenta faz uma leitura em toda região de memória da aplicação alvo procurando por valores especificados pelo usuário. Além de trazer os resultados esperados também apresenta uma estrutura do processo carregado na memória bem como o endereçamento de valores de tipo primitivo até estruturas complexas de objetos envolvendo diversos ponteiros.

---

1 As quatro liberdades do software livre: Disponível em: <http://www.gnu.org/*philosophy*/*free*-sw.pt-br.html>. Acesso em: 29 maio 2016.

2 *Cheat Engine*: Disponível em: <http://*cheatengine*.org/>. Acesso em: 15 maio 2016.

3 "*It comes with a memory scanner to quickly scan for variables used within a game and allow you to change them, but it also comes with a debugger, disassembler, assembler, speedhack, trainer maker, direct 3D manipulation tools, system inspection tools and more.*" (*CHEAT ENGINE*, [201-?]).

3. Memória

“Memória é um termo genérico usado para designar as partes do computador ou dos dispositivos periféricos onde os dados e programas são armazenados. Sem uma memória de onde os processadores podem ler e escrever informações, não haveria nenhum computador digital de programa armazenado” (LIMA, 2001). Neste artigo será explanado especificamente sobre a memória principal de um computador, conhecida também como memória RAM ou memória primária.

Em um computador a memória RAM têm a função de armazenar dados que estão sendo processados (aplicações, arquivos, imagens etc.). Ela atua como uma unidade de rápido acesso se comparada ao disco rígido; perde no quesito velocidade somente para as memórias cachê e registradores do processador (PACIEVITCH, [201-?]). A principal característica da memória RAM é a volatilidade, ou seja, todo conteúdo armazenado é perdido assim que o computador desliga. Cenários como esse tornam impossível a recuperação dos dados alterados sem prévio salvamento após a queda de energia.

Sabe-se que toda aplicação ao ser executada deve antes ter seus dados coletados no disco rígido e carregados na memória principal do computador, ou seja, passará a existir na memória RAM uma cópia do conteúdo original contido no disco rígido.

Regras como essa citada anteriormente explicam acontecimentos como abrir e alterar arquivos de texto, mas antes de salvar qualquer alteração o usuário é surpreendido por uma queda de energia e toda modificação é perdida. Isso se deve ao fato de que quando o arquivo estava aberto no editor o conteúdo apresentado na tela era o mesmo carregado na memória principal do computador e não houve tempo de salvar as modificações consequentemente o conteúdo do arquivo na memória principal estava alterado mas o conteúdo do disco rígido manteve-se original por este motivo ao abrir novamente o arquivo de texto suas informações serão coletadas no disco rígido e carregadas novamente na memória principal do computador e nenhuma alteração estará salva.

Quando um programa está em execução, tem um espaço armazenado na memória RAM para guardar informações durante seu ciclo de vida. O sistema operacional gerencia isso, ou seja, ele define qual o espaço, tamanho e região de memória que será separada para um determinado processo, porém essas regiões não são apontadas fisicamente, mas são regiões de memória virtuais que podem ser traduzidas em endereços físicos no *hardware*.

O sistema operacional possui uma tabela de tradução de endereços virtuais para endereços físicos, essa técnica permite que todos os programas em execução não necessitem estar carregados no mesmo instante na memória principal, assim o sistema operacional faz a gerência de quais processos estarão carregados na memória principal quando estiverem em execução, mas não estão sendo utilizados pelo usuário (TANENBAUM, 2010).

4. Aplicação Alvo

Existem aplicações de todos os tipos espalhadas pela internet. Geralmente os usuários procuram por aplicações do próprio interesse, e estas normalmente não trazem consigo o código fonte, mas trazem arquivos binários compilados para uma plataforma específica e outros arquivos como bibliotecas para garantir o correto funcionamento do *software*.

Caso o usuário queira expandir ou alterar as funcionalidades originais de um *software* sem possuir o código fonte do projeto, com certeza terá uma tarefa difícil e árdua, mas não impossível. Há muitas maneiras de se fazer isso, o ideal é que o usuário já esteja bastante adaptado ao software e com profundo conhecimento na maioria das funcionalidades, também é indispensável profundo conhecimento da plataforma para qual o *software* foi compilado.

A aplicação alvo apresentada é um jogo basicamente constituído por um personagem denominado *Maryo.* Além de possuir vários cenários que são explorados conforme o jogador avança na partida, esses cenários estão repletos de armadilhas e passagens secretas incluindo adversários que tentam bloquear a passagem do jogador. Ao tocar os adversários em uma região que não seja a parte superior (exceto adversários que não foram programados para morrer) o jogador perde uma vida e retorna ao início do cenário, se o total de vidas for igual a zero então o jogo recomeça. De início, o personagem possui 3 vidas disponíveis que são decrescidas de uma em uma toda vez que o jogador é derrotado ou acrescidas de uma em uma a cada cem moedas coletadas pelo jogador. O objetivo do jogo é coletar o número máximo de moedas e eliminar os adversários que se coloquem no caminho do personagem afim de obter a maior pontuação possível.

5. Manipulação dos Dados

Como explicado em tópicos anteriores deste artigo, todo processo em execução é carregado na memória principal do computador. Ao entrar em uma partida os dados vitais do personagem estarão carregados na memória primária uma vez que esses dados fazem parte da aplicação alvo em execução.

No canto superior à direita da tela está visível para o jogador o contador de vidas que marca um total de três vidas para o personagem logo de início.

Figura 1. Contador de vidas.

O contador de vidas é um valor do tipo inteiro, ou seja, não é um valor decimal pois logicamente não é possível possuir a metade de uma vida, mas somente vidas inteiras. Operações aritméticas com a quantidade de vidas são realizadas sempre que o jogador obtém uma vida ou a perde; assim, as operações de soma e subtração são

respectivamente aplicadas. Para saber a quantidade de vidas do personagem em determinados momentos do jogo é necessário consultar esse valor, que está armazenado em uma variável alocada em algum endereço aleatório da memória principal mas dentro dos limites do endereço virtual definido para o processo.

Para localizar o endereço de memória que armazena a quantidade de vidas do personagem será utilizado os recursos da ferramenta *Cheat Engine*. Ao abrir a ferramenta, o primeiro passo é anexar a aplicação alvo a ela para que seja possível escanear sua região de memória. Através do ícone de um computador localizado no canto superior esquerdo é feita a anexação da aplicação alvo à ferramenta.

Sabe-se que a quantidade de vidas é igual a três devido a notificação do contador de vidas. Na ferramenta *Cheat Engine*, no campo de texto *Value*, o valor três deve ser inserido com as propriedades *Scan Type* igual a *Exact Value* pois o valor a ser procurado na memória é exatamente igual a três (quantidade de vidas) e o *Value Type* igual a quatro *bytes* pois o valor três é um inteiro, um tipo inteiro possui quatro *bytes.* Em seguida, o botão *First Scan* deve ser pressionado e então todo endereço de memória da aplicação alvo que estiver armazenando o valor três será mostrado.

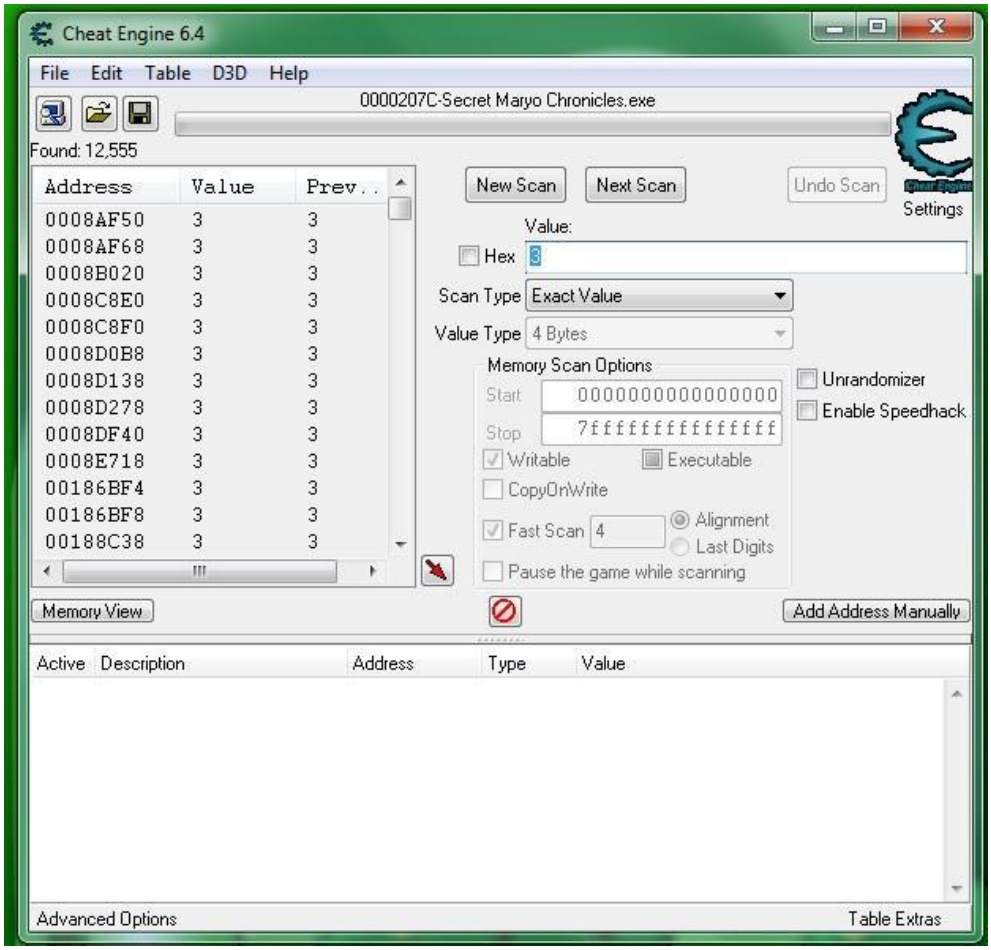

Figura 2. Ferramenta Cheat Engine.

Muitos endereços são encontrados ao ativar esse recurso, pois outras variáveis

(endereços de memória) estão armazenando o valor “3” ao mesmo tempo. Para saber exatamente o endereço que armazena a quantidade de vidas, o jogador deve ir ao jogo e perder uma vida propositalmente. Será decrescido o total de vidas em uma unidade; logo em seguida, na ferramenta *Cheat Engine*, o campo *Value* deve ser alterado de três para o número dois e o botão *Next Scan* pressionado, assim a ferramenta irá vasculhar todos os endereços que estavam armazenando o valor três e que agora estão armazenando o valor dois. Essa sequência de passos deve ser repetida até que sobre somente um endereço na lista de endereços. O endereço final é o endereço que armazena a quantidade de vidas.

O endereço encontrado faz referência à um endereço virtual de memória e não um endereço físico. Além disso é possível existir dois endereços virtuais iguais desde que eles estejam em processos diferentes. Com o endereço de memória que armazena a quantidade de vidas em mãos é possível escrever uma ferramenta que altere o valor do endereço de memória desejado. Com a ferramenta Dev-C++ o seguinte código deve ser escrito como ilustra a figura 3:

```
1  #include <windows.h>
2
3  int main(void) {
4
5      DWORD dwPid;
6      LPVOID enderecoVidas = (LPVOID) 0x05426F0;
7      int novaQuantidadeDeVidas = 10;
8
9      HWND hWindow = FindWindow(0, "Secret Maryo Chronicles");
10
11     GetWindowThreadProcessId(hWindow, &dwPid);
12
13     HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, false, dwPid);
14
15     WriteProcessMemory(hProcess, (LPVOID)enderecoVidas, &novaQuantidadeDeVidas, 4, NULL);
16
17     CloseHandle(hProcess);
18
19     return 0;
20 }
```

Figura 3. Código para alterar a memória.

Alguns endereços virtuais não são fixos, como o caso do endereço virtual da quantidade de vidas; a cada vez que o processo é executado ele muda. Existe, porém, uma regra para definir sua posição na memória e é possível capturar o endereço onde a variável estará alocada a cada execução do programa (neste caso será utilizado o endereço randômico).

A seguir, são explicados os principais procedimentos realizados no código listado na figura 3:

- Linha 1: é possível ver a inclusão da biblioteca “windows.h” que contém as funções utilizadas no restante do código, logo após a função *main* é declarada para que o programa comece a executar a partir dela.

- Linha 5: uma variável do tipo DWORD é declarada; ela armazenará um número inteiro positivo identificador do processo alvo (PID). Esse identificador é necessário para obter um manipulador do processo para posteriormente modificar a região virtual de memória alocada para ele.

- Linha 6: uma variável do tipo LPVOID é declarada; essa tipagem significa um ponteiro genérico longo (*Long Pointer Void*). Essa variável apontará para o endereço de memória onde está localizada a quantidade de vidas o valor atribuído a ela é um valor hexadecimal, pois é assim que os endereços de memória estão organizados.

- Linha 7: uma variável do tipo inteiro é declarada, e conterá o novo valor desejado para a quantidade de vidas.

- Linha 9: a função *FindWindow* é utilizada com dois parâmetros. O primeiro é o valor zero e o segundo é o título da janela do processo alvo a ser procurada. O retorno da função é um manipulador da janela que possua o título igual ao segundo parâmetro informado na função.

- Linha 11: a função *GetWindowThreadProcessID* é chamada com dois parâmetros o primeiro sendo o manipulador da janela (já obtido com a função *FindWindow*), e o segundo parâmetro é um ponteiro para uma variável do tipo DWORD, ou seja, o endereço da variável para que a função chamada vá até o endereço do segundo parâmetro e escreva nele o PID do processo solicitado, então ao final da função chamada o valor do PID estará armazenado na variável dwPid. Para fornecer o endereço de uma variável como parâmetro basta acrescentar o caractere '&' antes do nome da variável.

- Linha 13: uma variável do tipo *HANDLE* é declarada e logo em seguida é atribuído à ela o *HANDLE* do processo alvo através da função *OpenProcess*. Todo processo em execução possui um *HANDLE*; ele é armazenado em uma tabela do sistema operacional. Com o manipulador e os privilégios necessários é possível gerenciar a memória virtual do processo alvo. O primeiro parâmetro da função chamada diz respeito ao nível de privilégio que será atrelado ao manipulador a ser retornado, os possíveis valores são constantes definidas pela documentação da Microsoft. O segundo parâmetro é um *boolean* que informa se os processos criados por esse processo herdaram o mesmo manipulador. O terceiro parâmetro é o PID do processo alvo, um manipulador será retornado especificamente para operações com esse processo.

- Linha 15: a função *WriteProcessMemory* é invocada. Esta função é a principal, pois ela é quem vai escrever no endereço virtual de memória que foi especificado. O primeiro parâmetro é um *HANDLE* já obtido com a função *OpenProcess*, que definirá qual processo terá a região virtual de memória alterada (no caso, a aplicação alvo). Para utilizar essa função, o manipulador deve obrigatoriamente possuir privilégios de escrita (neste artigo foi requisitado privilégios máximos com a constante *PROCES_ALL_ACCESS*, mas é possível obter outros tipos de privilégios como somente escrita, leitura, escrita e leitura etc.). O segundo parâmetro é um LPVOID, um ponteiro que aponta para o

endereço de memória a ser alterado, no caso a quantidade de vidas do personagem. O terceiro parâmetro é um ponteiro para a variável que armazena o novo valor a ser colocado no endereço que contém o total de vidas do personagem (no caso, sabe-se que as vidas é um valor do tipo inteiro consequentemente a variável que contém o novo valor de vidas deve ser do tipo inteiro também). O quinto parâmetro é a quantidade de *bytes* a ser escrita no endereço alvo, ou seja, a partir do endereço especificado no segundo parâmetro os primeiros quatro *bytes* serão alterados. O sexto parâmetro não é obrigatório; é um ponteiro que a função utiliza como valor de *callback*, caso não seja *NULL* será armazenado nele a quantidade de *bytes* que a função teve êxito em sobrescrever (no caso o programador especifica quatro *bytes* para a função, mas devido as permissões da página de memória que a variável está contida talvez não seja possível sobrescrever integralmente os *bytes* e somente parte deles).

- Linha 17: a função *CloseHandle* é chamada para fechar o manipulador do processo alvo que foi aberto anteriormente com a função *OpenProcess*. Sempre que um manipulador é aberto após realizada as operações desejadas ele deve ser fechado, isso é uma boa prática de desenvolvimento.

- Linha 19: a linha *return* 0; indica o fim da execução do programa retornando zero erros, ou seja, tudo ocorreu como esperado.

Após compilar e executar essa ferramenta, a quantidade de vidas será alterada na memória, mas o contador de vidas não terá seu valor alterado pois o valor visual manteve sua integridade. O endereço de memória que contém a quantidade de vidas é que foi modificado - na próxima morte do personagem a vida será decrescida em uma e a função que atualiza o contador será chamada. O valor inserido no endereço de memória pela ferramenta manipuladora foi o “10”, decrescido de 1 será igual a 9, consequentemente após a primeira morte do personagem depois da modificação de memória o valor do contador e a quantidade de vidas do personagem serão iguais a nove, por fim a memória foi modificada.

6. Considerações Finais

Toda aplicação em execução necessita de recursos fornecidos pelo sistema operacional (funções, interrupções, execuções, entre outros.). Alguns desses recursos permitem ações que podem ser utilizadas contra a própria aplicação. A falha apresentada obtém vantagens e privilégios ilícitos com base na negligência do desenvolvedor que não tratou os cenários permissores de potenciais brechas de segurança no software. Não há limites para a execução desse tipo de prática, é possível também manipular não somente endereços únicos do software tal como variáveis alocadas em determinados endereços na memória RAM, mas alocar recursos computacionais para um determinado processo e implementar procedimentos e funções não existentes no projeto original é algo real, bem como alterar as funcionalidades originais para que apontem outros endereços de memória desejados executando trechos de códigos modificados ou criados pelo atacante.

Referências

CHEAT ENGINE. Disponível em: <http://www.*cheatengine*.org/*about*ce.php>. Acesso em: 15 maio 2016.

PACIEVITCH, YURI. **Memória Cachê.** Disponível em: <http://www.infoescola.com/informatica/memoria-cache/>. Acesso em: 15 maio 2016.

LIMA, E. M. Wesley. **A Memória do Computador**. Abr 2001. Disponível em: <http://www.ime.usp.br/~weslley/memoria.htm>. Acesso em: 15 maio 2016.

TANENBAUM, S. Andrew. **Sistemas operacionais modernos**. 3.ed. São Paulo: Prentice-Hall, 2010.

GNU. Disponível em: <http://www.gnu.org/*philosophy*/*free*-sw.pt-br.html>. Acesso em: 15 maio 2016.

Microsoft API *and Reference Catalog*. Disponível em: <https://msdn.microsoft.com/en-us/*library*/ms123401.aspx>. Acesso em: 15 maio 2016.

Anexo A

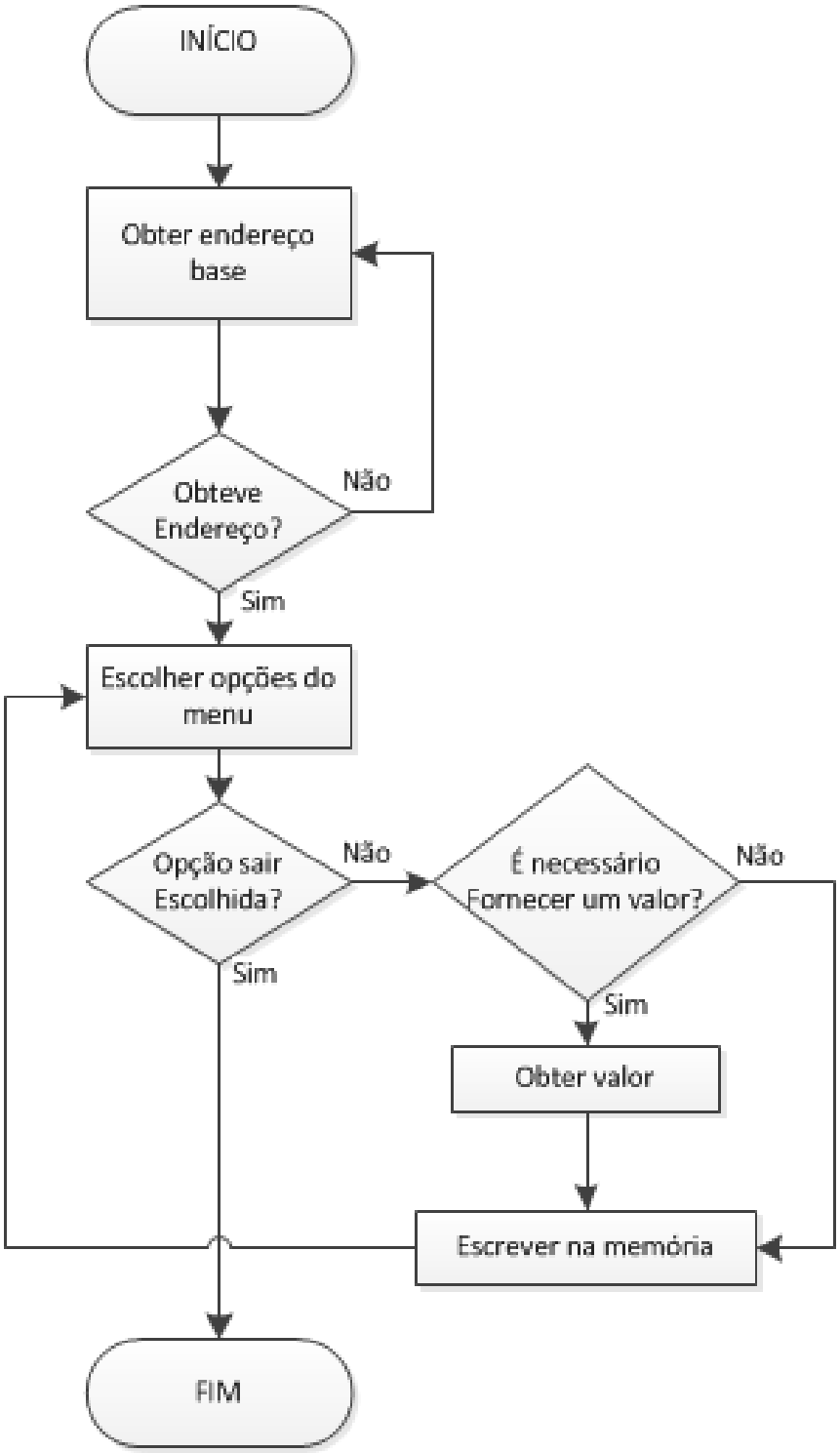

Figura 4.1 Fluxograma da aplicação.

Anexo B

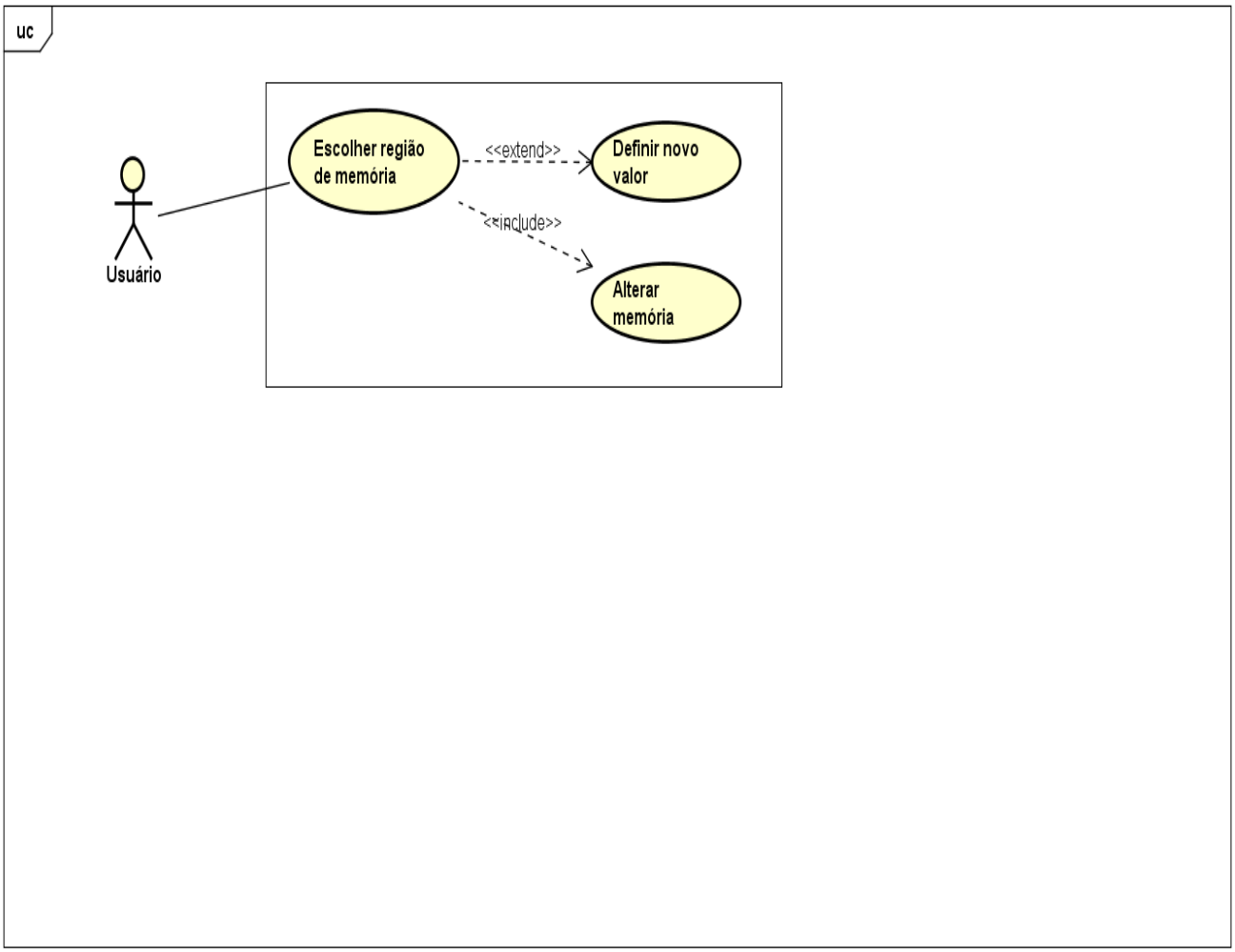

Figura 4.2 Diagrama de caso de uso.