Reversing - Les fondamentaux

Galathil

Important : Ce tutoriel a été rédigé par Nobodie pour le site wow-emu.fr. Je le recopie ici à l'identique, Il se peut que certaines informations soient obsolètes à ce jour. Il manque les images, leurs présences seront cependant signalées dans le texte. Bonne lecture !

Reversing - Les fondamentaux

Bonsoir à tous,

aujourd'hui, nous allons voir les bases du Reversing appliqué à Wow. Nous verrons d'abord ce qu'est le reversing dans ses grandes lignes, puis nous identifierons les cas dans lesquels nous avons besoin du reversing durant le développement d'un serveur WoW. Après une partie pratique vous permettant d'entrer en contact avec le monde du reversing, nous indiquerons une liste des bases à avoir pour commençer dans le reversing.

1) Qu'est-ce que le reversing ?

Nous commençons par un peu de théorie. Si vous avez lu quelques uns de mes tutoriels, vous avez vu que je parlais de reversing, mais de quoi s'agit-il exactement ?

Pour rester très large, le reversing aussi appelé la rétro-ingénierie, est l'activité qui consiste à étudier un objet pour en déterminer le fonctionnement interne ou la méthode de fabrication. Alors oui, cette définition est un peu pompeuse et ne veut pas dire grand chose. Basiquement, cela consiste à trouver une information à partir d'une base limitée de connaissances. Par exemple, les développeurs du Jailbreak iPhone font de la rétro-ingénierie sur le téléphone pour comprendre comment il fonctionne et trouver des failles potentiellement exploitables. De même, les copies de matériel informatique (les contrefaçons) sont souvent faites à partir de rétro-ingénierie sur leurs modèles. On utilise également la rétro-ingénierie pour étudier les virus informatiques afin de comprendre leur fonctionnement et déjouer leurs protections. Enfin, les plus grosses entreprises ont un pôle de recherche qui tente de faire du reversing sur leurs propres applications afin de les rendres moins faciles à détourner par les crackers. Evidemment, cela ne constitue pas une liste exhaustive □

Ce qu'il faut bien comprendre, c'est qu'**aucun système n'est infaillible**. N'importe quelle application, n'importe quel matériel peut être reversé à partir du moment ou quelqu'un possède cet objet dans ses mains. Ce qui fait la différence, c'est le rapport entre les connaissances nécessaires au reversing de l'objet, et les protections qui ont été mises en place.

On pense souvent, à tord, que le reversing ne consiste qu'en l'étude des applications binaires (les exécutables). Pourtant, il faut bien comprendre que **toute information est potentiellement importante**. Des logs d'une application permettent d'en connaître davantage. Des sniffs réseau dans le cas d'une application en ligne ou d'un virus sont également très utiles. Dans le cas d'une entreprise par exemple, le fait de savoir qu'une personne a été employée pour renforcer la sécurité d'une application est également importante pour ses recherches. **N'importe quelle information** touchant de près ou de loin à votre objet d'étude **peut faire gagner du temps**. Il est donc tout à fait indispensable de regrouper le maximum d'informations pour maximiser ses chances et trouver au plus vite l'information recherchée.

Ce qu'il faut comprendre également, c'est qu'il est rarement possible, dans le cas d'une application compilée, de retomber sur le code source originel une fois l'application compilée. Cela veut dire qu'une perte d'informations a lieu entre l'étape ou vous codez une application et l'étape ou l'application est compilée et transformée en exécutable. Par exemple, tous les commentaires, noms de fonctions ou de variables disparaissent pour être remplacés par leur adresse en mémoire (je simplifie). Ainsi, durant le processus de décompilation qui permet de passer d'un exécutable binaire à un code assembleur, puis un pseudocode C par exemple, il devient très difficile de lire le code et l'identifier. Une des seules aides disponibles sont les chaines de caractères utilisées dans l'application, qui sont heureusement très utiles pour comprendre certains bouts de code. Toute la démarche consiste donc à comprendre ce pseudocode et en identifier le fonctionnement.

2) Que pouvons-nous reverser dans le client Wow ?

Il y a plusieurs choses à reverser dans un client Wow. Voici une liste non exhaustive de choses qu'on peut trouver/faire à partir d'une rétro-ingénierie :

- Identifier l'ensemble du protocole de communication. Cela est indispensable pour développer un serveur Wow dans une version précise du jeu. Avant qu'un serveur Wotlk, Cataclysm ou MoP n'ouvre ses portes, il a fallu reverser ce protocole. Jusqu'à Cataclysm, des projets publics proposaient des core contenant une bonne partie de ce travail, permettant à de nombreuses personnes de ne pas avoir à faire ce travail. Depuis MoP, avec l'accroissement de la complexité du reversing, presque aucune source publique n'est désormais disponible. Cela ne devrait pas s'arranger avec la prochaine extension Warlords of Draenor. Si une équipe souhaite donc se lancer dans ce projet, il y a fort à parier qu'elle devra tout faire d'elle même.
- Ajouter des fonctionnalités dans l'exécutable client. Blocage du realmlist, download automatique de mises à jour, vidange du cache, News feed sur la page de login du jeu. Autant de petites fonctionnalités qui nécessitent souvent plusieurs modifications afin de mettre en place ces fonctions.
- Développer des Bots/Hacks pour le jeu. Petits malins C'est pas bien
- Integrer des virus ... No comment

Pour notre part, nous allons nous intéresser plus particulièrement au premier point qui consiste à identifier le protocole de communication. Si vous avez lu un de mes précédents tutoriels, vous avez compris que pour fonctionner, le client a besoin d'être constamment en communication avec le serveur. Ceux-ci échangent de nombreuses informations (déplacements, sorts lancés, items portés, attaques, ...) liées au jeu à l'aide de paquets de données qui sont forgés de façon bien précise et ne supportent aucune erreur. Depuis Cataclysm, chacun de ces paquets (et il y en a plusieurs milliers) changent d'une sous version à une autre (par exemple, il y a eu 6 changements de protocole entre la 5.1.0 et la 5.4.2). Si l'on veut donc travailler sur un serveur MoP, il faut donc reverser le protocole d'une version en particulière. Il n'est pas humainement possible de suivre Blizzard et de refaire ce travail à chaque changement. Il daut donc accepter de s'arrêter à une version (dans notre exemple de notre serveur Mist-Eria, nous avons choisi la 5.1.0) et de travailler sur celle-ci pour une durée plus ou moins longue, puis de passer à une version ultérieure par la suite.

Cependant, ce choix n'est pas du tout anodin. Contrairement à ce que les joueurs pensent en général, ce n'est plus comme sous BC ou WOTLK ou nous choisissions simplement la dernière version (2.4.3 ou 3.3.5) car elle contient le plus de fonctionnalités. Ce n'est plus au bon vouloir de l'équipe de développement, mais cela dépend désormais de la capacité de l'équipe à avancer sur l'une ou l'autre version. Ainsi, dans notre cas, nous avions travaillé en parallèle sur la 5.1, la 5.2 et la 5.3 durant plusieurs semaines, avant d'abandonner les deux plus jeunes pour nous concentrer sur la 5.1 car elle nous donnait les meilleures chances de réussite. Maintenant, il faut tout de même relativiser. La différence entre deux version n'est que pur marketing, les choix de développement étant toujours très limités et ne privilégiant que les dernières instances/raids/bg ouverts.

Toujours est-il que reversing ce protocole n'est plus mince affaire. Heureusement, il est possible d'utiliser d'autres informations que le binaire client afin de comprendre le fonctionnement de l'application. On peut utiliser :

- Les rapports de crash du client (situés dans le dossier Errors) qui permettent de retracer les derniers appels de fonctions ayant causé le crash.
- Les sniffs effectués sur le réseau afin d'identifier une partie du protocole de communication et notamment des opcodes transmis (on y reviendra dans un autre tutoriel).
- Les données du client telles que les DBCs qui permettent d'identifier un certain nombre de variables dans le client et d'en déduire les variables à envoyer durant les transmissions.
- Les ressources fournies par différentes équipes internationales qui partagent certaines informations (on remercie au passage TOM_RUS qui fournit une base de données faisant la relation entre les adresses et les noms de fonction à chaque mise à jour et qui permet d'y voir vraiment plus clair)
- N'importe quelle autre information dont je n'aurais pas encore moi même connaissance et sur lesquelles j'espère tomber un beau jour

Voici un aperçu des choses qui sont faisables grace à la rétro-ingénierie. Nous reviendrons d'ailleurs plus en détail sur les protocoles de communication dans un futur tutoriel expliquant étape par étape les objectifs à accomplir pour espérer ouvrir un serveur sur une nouvelle extension.

3) Se lancer dans le reversing Wow, que faut-il savoir/faire

Si les deux précédentes partie ne vous ont pas découragé et que vous avez compris, dans l'ensemble, de quoi on parlait, vous êtes prêts à vous frotter à votre désormais némésis, j'ai nommé, l'exécutable Wow.exe!!! Du sang va couler, des larmes vont être versées, et la cervelle va tâcher le sol, mais vous en voulez ! alors c'est parti

Premièrement, on rappelle les grandes étapes de la compilation. On prend évidemment pour exemple le client Wow.

1. D'abord, Blizzard développe son client en CPP (On le sait parce que le code compilé ressemble à du code CPP compilé. Et entre nous, ça ne fait pas l'ombre d'un doute). Le code est découpé en des centaines (milliers ?) de classes contenant des fonctions, dispersées

dans des centaines de fichiers.
2. Ensuite, la première étape de la compilation va compiler l'ensemble de ces fichiers dans un langage intermédiaire. A ce moment, le code n'est déjà plus lisible par un humain.
3. Troisième étape, le compilateur va transformer tous les appels de fonctions par leurs adresses. Il va également optimiser le code pour la machine. A ce moment, le langage utilisé est de l'assembleur.
4. Quatrième étape, il regroupe l'ensemble des fichiers compilés à l'intérieur d'un seul et même fichier qui deviendra l'exécutable final de l'application. On appelle cet exécutable un binaire.

Cela donne donc, au final, un exécutable binaire qui ressemble à ça, ouvert dans un éditeur héxadécimal (HxD pour ceux qui veulent se lancer) :

[Image: 141223063148590091.jpg]

Vous vous doutez bien, c'est imbouffable. Et pourtant, ceci marque déjà la première étape de la démarche de reversing.

Maintenant, l'objectif final, est de remonter le plus possible dans la compilation afin de retomber sur un code source le plus proche possible de la source de base, celle que les développeurs de Blizzard avaient devant les yeux quand ils ont développé le jeu.

Pour cela, on va commencer par passer du code binaire vers le code assembleur. Cela se fait facilement avec n'importe quel désassembleur (ben oui, lors de la compilation, on a un code assembleur qui sert à **assembler** l'application, donc pour retrouver l'assembleur, on doit **désassembler** l'application). Il en existe toute une floppée (OllyDBC, W32DASM, IDA, Hopper Disassembler ...), avec chacun leurs avantages et inconvénients. Pour ma part, j'utilise IDA Pro 6.1. Malheureusement, je ne pourrais pas vous fournir de lien pour récupérer cette version, à vous donc de la trouver sur le Net (c'est vraiment simple à trouver).

Avec votre désassembleur, vous ouvrez donc votre exécutable. Le logiciel va alors effectuer ce qu'on appelle une **analyse statique** de l'application (au contraire d'une analyse dynamique qui elle se fait lorsque vous lancez l'application et y attachez votre désassembleur). Sans rentrer dans les détails, l'analyse statique va parcourir l'ensemble de l'application afin de retrouver le code assembleur. L'opération peut prendre plusieurs minutes selon la machine que vous possédez, n'ayez donc pas peur si celle-ci souffre un peu

Je vous ai dit que c'était facile tout à l'heure, et ça l'est. Mais pourquoi alors cela prend-il un temps monstre, même pour votre bête de course capable de faire tourner n'importe quel jeu ? Tout simplement parce que l'assembleur est un langage très capricieux. Pour retrouver le code assembleur, votre désassembleur doit parcourir **l'ensemble du code** plusieurs dizaines de fois et surtout ne rien louper. En assembleur, chaque "ligne de code" prend une taille différente dans l'exécutable. Il faut donc parcourir le fichier du début et identifier à chaque fois la fin de la ligne pour ne pas avoir de décalage par la suite (sinon, le programme ne veut strictement plus rien dire). Tout cela, l'ordinateur le fait très bien, mais il lui faut du temps. Alors laissons le faire sagement et patientons

[Image: 141223063149605953.jpg]

Après une bonne dizaine de minutes pour moi, l'analyse statique est enfin terminée et nous obtenons alors un magnifique code assembleur en mode Graph View (que l'on désactive directement en faisant un **clic droit -> Text View** dans la fenêtre principale). Ahhh, ce beau code assembleur, on en aurait presque des larmes. Bon, c'est pas byzance hein, mais on commence à voir des lettres :o

L'étape suivante, c'est de retrouver un semblant de code lisible. Alors biensûr, on l'a dit avant, il ne faut pas espérer retomber sur un code CPP avec les commentaires, les noms de fonction etc, c'est impossible. En fait, même obtenir directement du CPP est très difficile. En réalité, nous allons réussir à obtenir ce qu'on appelle un **Pseudocode C**. Vous êtes déçus ? Bah oui mais vous en demandez beaucoup hein Wink Vous allez voir qu'un pseudocode C permet déjà de comprendre un bon nombre de choses. Pour faire ça, j'utilise le plugin HexRays d'IDA. On se place dans une fonction dont vous voulez lire le code (pour les besoins du test, vous allez vous placer dans une fonction quelconque, posez simplement votre curseur quelque part dans le code, et appuyez sur **Tab**. Après quelques secondes de calcul, vous aurez devant vos yeux ébahis un Pseudocode C presque compréhensible

[Image: 14122306314930091.jpg]

Ah, oui, pour info, vous risquez d'obtenir une erreur "Decompilation failure, FFFFFFFFFF .....". Si ça vous arrive, ne paniquez pas. Allez dans Options->Compiler->sous sizeof(bool), mettez 4, ça devrait régler le souci

Pour la suite, et bien ... il n'y a pas de tuto :x Plus vraissemblablement, j'établis plus loin une FAI (Foire aux infos) des principales choses à savoir pour se lancer, mais il faut bien comprendre qu'il n'y a pas de RoadMap pour ce travail. Tout cela se fait à grands coups de logique informatique, de recherche, de tests, d'erreurs, et surtout d'incompréhensions. Au début, tout vous paraitra trop compliqué, trop difficile, trop barbare, trop |insérer un mot fort ici|, et c'est le cas ! Mais ne désespérez pas Commencez d'abord par vous faire la main en appuyant plusieurs fois sur Tab afin de passer du code assembleur au pseudocode C généré. Si vous tombez sur une fonction vraiment longue, commencez par une plus courte (positionnez le curseur un peu plus loin dans l'assembleur et réappuyez sur **Tab**.

En plaçant votre curseur, vous voyez exactement la correspondance entre le code C et le code assembleur correspondant. N'hésitez pas à aller voir du côté de la doc sur l'assembleur sur Internet, vous aurez tout un tas d'informations pour comprendre ce qu'il s'y passe. Et n'oubliez pas que c'est en s'acharnant qu'on devient forgeron. Ou quelque chose comme ça :x

4) Foire Aux Infos

Première liste, tous les liens Wikipédia qui vous permettront de mieux comprendre les différentes étapes et notions abordées.

- **http://fr.wikipedia.org/wiki/Compilation...matique%29**
- **http://fr.wikipedia.org/wiki/Assembleur**
- **http://fr.wikipedia.org/wiki/R%C3%A9troing%C3%A9nierie**
- **http://en.wikibooks.org/wiki/X86_Disasse...ecompilers**

Cette partie sera remplie au fur et à mesure des travaux que nous menons au sein de notre équipe.

Voilà qui achève ce long tutorial permettant d'avoir une première approche du Reversing et de ce qu'il comporte comme difficultés. Nombreux seront les lecteurs à abandonner durant la lecture ou durant les premières minutes de tests avec votre premier désassembleur. Et personne ne saura vous le reprocher ! N'oubliez pas que personne n'a jamais appris en deux jours ce domaine. Moi-même, c'est le fruit de longues années d'expérimentations et d'incompréhensions. Il y a quelques années, je vénérais ces personnes qui savaient faire ça, les considérant littéralement comme des Dieux de l'informatique. Pourtant, aujourd'hui, je connais **un peu** ce domaine et je me rend compte que tout est accessible a qui a la volonté d'apprendre et de **comprendre**. Alors ne vous découragez pas, si vous ne comprenez pas de suite, repasser dans quelques jours, relisez le tutorial 2 fois, 10 fois, 100 fois s'il le faut. Mais à force de persévérance, tout le monde arrive à ses fins

~~N'hésitez pas à poster vos remarques si un paragraphe vous parait trop difficile ou si vous souhaitez en apprendre davantage.~~