Les protections de Blizzard

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. Bonne lecture !

Les protections de Blizzard

Prélude : Ce sujet n'est pas original, il a été transféré d'un précédent forum d'émulation en date d'Octobre 2013. J'en suis l'auteur, je le transfère donc pour les besoins de ce forum. Bien qu'il soit en partie ré-écrit, il se peut que certains termes soient inadaptés ou ne soient plus d'actualité.

Bonjour à tous,

aujourd'hui nous allons parler des protections mises en place par nos chers amis de Blizzard à l'encontre des serveurs privés. Il est aujourd'hui très difficile de développer un serveur sous MoP, mais ça n'a pas toujours été le cas. Petit flashback dans le lointain passé des versions magiques 1.X, 2.X etc. Mais d'abord, un petit point sur les éléments qui ont changé.

Avant de faire le point sur les différentes évolutions et complexifications du système, nous allons rapidement réfléchir sur les "leviers" sur lesquels Blizzard peut jouer pour mettre des bâtons dans les roues des serveurs de jeu. Si vous avez lu mon précédent guide sur le fonctionnement général d'un serveur de jeu, vous avez vu qu'il existe plusieurs points critiques. Blizzard peut donc jouer par exemple sur le contenu, en créant du contenu de plus en plus difficile à implémenter. C'est tout à fait faisable, mais techniquement, le but du contenu est de faire plaisir au joueur. Complexifier le contenu au détriment du plaisir du joueur, c'est un peu se tirer une balle dans le pied. Il pourrait également changer les systèmes de jeu (par exemple, changer en profondeur le système de dégats pour faire des dégâts de zone à la manière de Rift et non plus le système de ciblage comme on le connait. Malheureusement, ça aussi, ça touche à la politique même du jeu. Non, pour enquiquiner le plus possible les développeurs de serveurs wow, il faut trouver quelque chose de bien plus machiavélique Heureusement, il existe une solution qui regroupe tous ces points :

- Indépendant du gameplay du jeu
- Impossible à contourner (pour que les serveurs privés soient obligés de s'y conformer)
- Peu coûteux
- Invisible pour les joueurs (comme ça en plus les joueurs du privé ne voient pas que les développeurs font un gros travail ! Quand je vous disais que c'était machiavélique)

Il existe un seul système qui regroupe tous ces points sur lequel Blizzard peut tout gérer : il s'agit de la couche Réseau ! Tout le protocole de communication Client<->Serveur. Il est :

- **Indépendant du gameplay du jeu :** C'est la couche la plus basse du jeu. Peu importe le système qui se trouve au dessus, les données sont envoyées.
- **Impossible à contourner :** C'est un système très sensible. Si le client attend de recevoir un message contenant l'id du joueur, son level, ses spells et un pain au chocolat, alors le client doit recevoir l'id du joueur, son level, ses spells et un pain au chocolat. Si il y a une erreur, vous obtenez a coup sûr soit un crash client, soit un comportement ingame tout à fait mortel. Et impossible de contourner ce système pour le serveur, il est tellement imbriqué dans le client que tenter de cracker le système revient à recoder tout cela .... en assembleur bonne chance à vous
- **Peu coûteux :** Techniquement, ça ne coûte rien d'inverser des valeurs à chaque mise à jour. Il suffit que le serveur officiel fasse cette inversion également. Facile lorsque Blizzard possède les sources de son propre jeu. Sans avoir les sources, cela devient beaaaaaucoup plus complexe.
- **Invisible pour les joueurs :** Qui s'est déjà rendu compte que derrière un jeu tel que Wow, vous envoyiez l'équivalent de 100 gros SMS chaque seconde ? Pas moi, avant de me confronter à cette horreur.

La couche Réseau est donc le fier gagnant, et depuis les débuts de Wow, Blizzard "s'amuse" à notre grand regret, à obfusquer un peu plus chaque version. Les paquets envoyés se sont donc petit à petit allongés et complexifiés, comme nous allons le voir sous peu.

Dernier petit rappel préliminaire : Le contenu d'un paquet "type"

Dernier petit rappel préliminaire : Le contenu d'un paquet "type" :

Imaginons un paquet type simplifié. Ce paquet est un paquet utilisé à la fois du côté client, comme du côté serveur (sa structure est la même des deux côtés). Il contient les informations suivantes :

- Un Opcode sur 2 octets : C'est l'identifiant pur du paquet. Il sert à identifier le paquet pour savoir quoi en faire (par exemple, imaginons l'opcode 100 qui est chargé d'envoyer des infos sur un joueur)
- Un Guid sur 8 octets : C'est l'identifiant du joueur pour lequel on envoie ces informations.
- Un float (un chiffre à virgule) sur 4 octets : la hauteur du joueur par rapport au sol. Cette information permet de savoir si le joueur est en train de voler.
- Une chaîne de caractères sur X octets : C'est le nom du joueur. Comme on ne sait pas d'avance quelle taille fait cette chaîne, on utilise un caractère spécial en fin de chaîne pour signifier qu'on vient de terminer la lecture (si on veut écrire NoboDie, on écrit le pseudo suivi d'un caractère 0x00, un caractère NULL délimitant la fin de la chaîne)
- Un compteur sur 1 octet : ce compteur va nous indiquer le nombre d'éléments qui vont être envoyés dans la boucle suivante.
- Une boucle qui envoie une variable sur 4 octets : Un Id de spell. Imaginons que le joueur ai appris 4 sorts. Le compteur précédent vaudra 4, et la boucle remplira le paquet avec 4 IDs de spells les uns après les autres.

Ceci est un paquet très simple qui n'est là que pour l'exemple. Evidemment, le système envoie une multitude de paquets différents et atrement plus complexes que celui-ci, mais il faut bien un début à tout et ce paquet "type" regroupe différents types de variables que l'ont est amenés à rencontrer sur wow.

Dernier point sur la façon la plus simple dont un paquet peut être traité (en fait, comme ça a été fait au début). Le système le plus simple est d'avoir un tableau du type tableau[opcode] = fonctionquiremplitlepaquet();

C'est assez intuitif : on associe à chaque opcode (qui, je le rappelle, est l'identifiant du paquet et sert à savoir de quoi il va parler), la fonction qui le traite (dans le cas ou le message arrive vers le client) ou la fonction qui le remplit (dans le cas ou le message part du client). Nous verrons que cela s'est évidemment complexifié au fil des années

Bien, maintenant que nous avons défini un paquet (fictif, je le rappelle) et la façon la plus simple de le traiter, voyons comment celui-ci a pu être géré au fil des versions (je ne complexifie pas les choses, c'est réellement le fonctionnement du système)

1. Le bonheur de Wow Vanilla

Ok, on commence doucement, Blizzard vient de sortir son premier MMOrpg, il faudrait pas qu'ils ne se plantent et que leur jeu ait des soucis. Afin de simplifier l'explication, je ne traiterais ici que l'envoi et la réception de variables de type UINT64, cad en général un Guid (de joueur, de créature, d'item, ...) :

```
int64 guid;
packet << guid;
```

Comme vous pouvez vous rendre compte, le système est très simple. Aujourd'hui, 100% des paquets de la 1.X sont connus. Encore heureux me direz-vous, ça fait 10 ans qu'elle existe ^^. Aucune modification sur le protocole de communication n'a été faite sur l'ensemble de la période 1.X (Wow Vanilla). Les serveurs Vanilla pouvaient donc vivre tranquilles. Il y avait seulement une centaine de paquets différents.

2. Wow Burning Crusade, ça commence ..

Arrivée de la nouvelle extension. Surprise, avec les nouveautés du gameplay, près de 200 nouveaux opcodes (et donc paquets) apparaissent, ce qui multiplie par 3 le travail. Heureusement, les 100 premiers paquets ne changent pas d'identifiant, il est donc facile d'avoir une version à peu près aussi fonctionnelle que la version 1.X Vanilla. Autre nouveauté, certains paquets ne sont utilisés que dans un sens (du serveur vers le client par exemple). Ca ne pose pas gros problème, les opcodes sont toujours enregistrés dans un tableau avec leur handler associé. Presque du bonheur, j'vous dit .Toujours aucun changement d'une sous version à une autre, en dehors de nouveaux opcodes pour les nouvelles fonctionnalités. On passe à la suite.

3. Wow Wrath Of The Lich King, Aïe ...

Et voilà, ça devait arriver. Evidemment, ça devenait trop simple de trouver les opcodes directement dans le tableau de handlers. Les opcodes sont maintenant de trois types : les opcodes d'authentification (les authOpcodes), les opcodes de type JAM (les jamOpcodes) et les opcodes de type table (les tableOpcodes). Ces derniers sont toujours enregistrés via un tableau, mais il faut maintenant faire un calcul sur l'opcode. L'opcode n'est plus enregistré en dur, il faut donc jouer et sniffer les paquets pour trouver les correspondances. Ca devient long en temps de calcul. Par contre, les opcodes sont toujours les mêmes depuis la première extension, on peut encore retrouver des choses. Ouf !

Oh et puis tiens, on va plus simplement envoyer les GUIDs tels qu'ils sont, c'est à dire un uint64 (8 octets). On va maintenant faire un packedGUID. C'est la même chose, mais on va mettre en place un algorithme pour rapetisser le guid comme ça non seulement il ne prendra plus 8 octets en taille, mais en plus il sera plus compliqué à déchiffrer pour les développeurs du privé. Niark, Evil inside x)

En revanche, il "suffit" de comprendre le système une seule fois, puisqu'ensuite c'est la même fonction qui est appelée à chaque fois.

```
int64 appendPackedGuid(int64 guid) {
uint8 packGUID[8+1];
packGUID[0] = 0;
size_t size = 1;
for (uint8 i = 0;guid != 0;++i) {
if (guid & 0xFF) {
packGUID[0] |= uint8(1 << i);
packGUID[size] = uint8(guid & 0xFF);
++size;
}
guid >>= 8;
}
append(packGUID, size);
}

int64 guid;
packet.appendPackedGuid(guid);
```

Vous avez l'impression que ça se complique ? c'est bien, vous n'avez encore rien vu ...

4. Wow Cataclysm, La revanche du Retour du Jedi ...

Bon aller maintenant ça suffit, on en a marre des serveurs privés ! Oh punaise ils vont en baver là ! On va leur implanter ... des Opcodes aléatoires ! Waouw grande idée ! Ca revient à quoi ? et bien, à chaque sous version (4.0.6, 4.1.0, 4.1.0a, 4.1.3, ...) on va changer aléatoirement tous les Opcodes ! Hop hop hop ! Oh et puis tant qu'à faire, on va inverser les structures de chaque paquet ! Ca se traduit par plusieures choses pour les serveurs privés :

- Plus possible de se baser sur les versions précédentes pour retrouver les handlers, il faut tout refaire de zéro !
- Le serveur officiel change d'opcodes chaque version. Si on veut passer à la version suivante, il faut à nouveau tout refaire.
- Pour trouver les informations sur Wotlk et identifier les paquets/structures, on utilisait des sniffers. C'est cool les sniffers, sauf que vu que l'officiel change se paquets à chaque version, ça veut dire que les sniffs ne sont valables que le temps d'une version. S'il nous manque les sniffs pour un système tel que celui des guildes, on perd tout espoir d'ouvrir le système de guilde dans des temps raisonnables. Biensûr, les sniffs ne sont pas indispensables, mais ils simplifient légèrement le travail. Une fois le serveur officiel passé à une version ultérieure, impossible de refaire des sniffs pour la version travaillée actuellement. Difficile, n'est-ce pas ? □ Et ce n'est pas tout pour cette version ...

Arrivée de la version 4.3.4. Les serveurs privés en 4.0.6 commencent à pulluler, et ça, c'est pas bon pour Blizzard. Alors pour éviter que des serveurs 4.3.4 n'arrivent trop vite, on va inventer .... Les GUIDs XORés ! MOUAHAHAHAHAHAHAH

Comment ça marche ? et bien c'est un nouvel algorithme qui complexifie l'envoi des GUIDs. Basiquement, on envoie un premier octet qui sert de "mode d'emploi" à la lecture du guid. Chaque bit de l'octet correspond à un octet du guid (vous me suivez là ? x)). Si le bit est à 1, alors il faudra lire l'octet correspondant. Et puis comme c'st trop simple, on va le XORer avec cette valeur justement comme ça on est sûrs que c'est bien compliqué. Oh et puis comme maintenant on a divisé le guid en deux (octet "mode d'emploi" et guid à part), et bien on va envoyer l'octet au début, et puis le guid plus loin. Oh et puis, puisqu'on peut lire chaque octet du guid séparément selon le mode d'emploi, on va envoyer d'abord les 2 premiers octets du guid, puis on va envoyer quelques autres variables inutiles, puis les 4 octets suivants du guid. Puis quelques autres variables, puis enfin les deux dernier octets ! ouais, ça ça va être bien !

Ah au passage, on va aussi créer une nouvelle façon d'envoyer une chaine de caractères. On va envoyer la taille à part, puis la chaîne sans caractère de fin (le NULL dont on a parlé plus haut). Aller avec ça, je crois que c'est bon on leur en a fait assez baver ...

5. Wow Mists Of Pandaria, Pour moi ce sera un bon Random avec une rondelle de citron et des glaçons. Oh et apportez de la crème pour la

rondelle de citron et des glaçons. Oh et apportez de la crème pour la rondelle ... beaucoup de crème ....

QUoi ?! ils ont sorti un TrinityCore 4.3.4 ? Ok ! Maintenant ça va CH#&^{ER} !!!

Okkkk donc on prend tous les ajouts de la 4.3.4 (donc les Random Opcodes à chaque mise à jour intermédiaire, ainsi que la nouvelle façon de XORer un GUID). Et comme c'est devenu trop facile pour eux, on va faire encore pire. Avant, on envoyait le mask dans l'ordre, et le guid dans l'ordre, il était juste éparpillé dans tout le code. Et bien maintenant, on va envoyer tout ça dans le désordre AHAH ! Et en plus on va faire ça à chaque mise à jour intermédiaire ! MUAHAHAHAHAHAHA (vous voyez pourquoi on a besoin de la crème maintenant ?)

Donc à présent, et bien on va envoyer le bit 4 du mask, puis le 7, le 3, le 2, le 0, puis on va envoyer 20 autres bits qui servent à autre chose qui va bien foutre le bordel, puis on va envoyer le bit 1, le bit 6, et le numéro complémentaire, le 5 ! Par ici les gagnants du Loto. Oh pardon, vous avez loupé votre chance ? vous en avez une deuxième avec le GUID : envoyez d'abord l'octet 3 du guid, puis le 2, puis le 7, puis le 1, puis le 0, puis le 6, puis le 5. On non attendez, foutez quelques variables moisies entre tout ça, et finissez par l'octet 4 ! Bravo au gagnant de l'île-et-vilaine, il vient de remporter ... ah bah non rien, car il doit faire ça pour encore 1500 autres paquets ...

```
packet.WriteBit(guid[4]);
packet.WriteBit(guid[3]);
packet.WriteBit(guid[6]);
... 10 autres données qui n'ont rien à voir
packet.WriteBit(guid[2]);
packet.WriteBit(guid[5]);
packet.WriteBit(guid[0]);
... puis 3 autres
packet.WriteBit(guid[7]);
... puis une
packet.WriteBit(guid[1]);
... encore deux
packet.WriteByteSeq(guid[2]);
... Un pikachu sauvage apparaît ici
packet.WriteByteSeq(guid[0]);
packet.WriteByteSeq(guid[6]);
... J'aime les Kiwis
packet.WriteByteSeq(guid[3]);
packet.WriteByteSeq(guid[4]);
packet.WriteByteSeq(guid[1]);
... Ca sent le brûlé :/
packet.WriteByteSeq(guid[5]);
... Serait-ce la fin des haricots ?
packet.WriteByteSeq(guid[7]);
... OH YEAH ENCORE DES CHOSES MAIS ON A ENFIN FINI D'ENVOYER LE GUID !
```

Oh et puis on revient au sujet des Opcodes qui étaient divisés en 3 catégories. Bon c'est trop simple, on va les diviser en 7 et puis on va mettre une formule mathématique bien pourrave pour bien les emm..... ces développeurs de serveurs privés !

Et bien vous savez quoi ? ce n'est pas suffisant ! On l'a fait les amis ! On travaille bel et bien sur cette version et on va en sortir une très prochainement ! (un peu de pub personnelle n'a jamais fait de mal :3)

6. (Annexe) Prochaine extension, quelques idées pour nous faire suer un peu plus

Et oui on est comme ça nous ! On donne qelques idées au cas ou Blizzard serait en manque d'inspiration ! Top 5 des meilleures idées :

1. On randomize les variables elles mêmes. Rien de mieux que d'envoyer une variable sur 12 bits mélangés !
2. On prévoit plusieurs opcodes pour le même paquet ! Ben oui, si on utilise plusieurs opcodes, ça devient galère d'automatiser l'identification des opcodes dans les sniffs !
3. On créée un paquet qui définit comment on définit le paquet qui va être réellement envoyé ! Yo dawg, j'ai entendu dire que tu aimais les paquets de paquets de paquets de paquets ...
4. On réinvente tout un nouveau système de paquet 100% différent de la version précédente, comme ça on est sûrs de bien foutre tout

le monde dans la merde !
5. On sort le code source en version publique et on arrête d'embêter les serveurs privés qui ammènent du monde sur officiel (ben quoi, j'ai le droit de rêver non ?)

7. Le point sur Warlords of Draenor

Cette partie est rajoutée près d'un an après l'écriture des premières parties du tutoriel. Celle-ci fait le point sur ce qui a été mis en place par Blizzard entre temps pour la sortie de leur prochaine extension

Alors, tout d'abord, il faut savoir qu'au passage de la 5.3, il y a eu un changement très utile pour les serveurs privés : l'arrivée d'un nouveau compilateur chez Blizzard. Et ce compilateur apporte une chose très intéressante : un mauvais support des fonctions inline (ou, du moins, cela n'a pas été activé correctement du côté des développeurs de Blizzard). Cela a pour effet de rendre le code source client bien plus lisible. Voilà un exemple :

Ce code côté client en 5.1 :

```
if ( v11 == 8 )
{
LOBYTE(a2) = 0;
CDataStore__GetInt8(this, (int)&a2);
v10 = a2;
v11 = 0;
}
v17 = (unsigned int)v10 >> 7;
v16 = v11 + 1;
v15 = 2 * v10;
*(_BYTE *)(v12 + *((_DWORD *)v6 + 7) + 7) = v17;
if ( v16 == 8 )
{
LOBYTE(a2) = 0;
CDataStore__GetInt8(this, (int)&a2);
v15 = a2;
v16 = 0;
}
v20 = (unsigned int)v15 >> 7;
v19 = v16 + 1;
v18 = 2 * v15;
*(_BYTE *)(v12 + *((_DWORD *)v6 + 7)) = v20;
if ( v19 == 8 )
{
LOBYTE(a2) = 0;
CDataStore__GetInt8(this, (int)&a2);
v18 = a2;
v19 = 0;
}
v23 = (unsigned int)v18 >> 7;
v22 = v19 + 1;
v21 = 2 * v18;
*(_BYTE *)(v12 + *((_DWORD *)v6 + 7) + 4) = v23;
if ( v22 == 8 )
{
LOBYTE(a2) = 0;
CDataStore__GetInt8(this, (int)&a2);
v21 = a2;
v22 = 0;
}
v26 = (unsigned int)v21 >> 7;
v25 = v22 + 1;
v24 = 2 * v21;
```

```
*(_BYTE *)(v12 + *((_DWORD *)v6 + 7) + 90) = v26;
if ( v25 == 8 )
{
LOBYTE(a2) = 0;
CDataStore__GetInt8(this, (int)&a2);
v24 = a2;
v25 = 0;
}
```

Devient celui-ci côté client en 5.4 :

```
*(_BYTE *)(*((_DWORD *)v3 + 5) + v5 + 91) = CDataStore__ReadBit(&v36);
*(_BYTE *)(*((_DWORD *)v3 + 5) + v5 + 104) = CDataStore__ReadBit(&v36) != 0;
*(_BYTE *)(*((_DWORD *)v3 + 5) + v5 + 6) = CDataStore__ReadBit(&v36);
*(_BYTE *)(*((_DWORD *)v3 + 5) + v5 + 89) = CDataStore__ReadBit(&v36);
*(_BYTE *)(*((_DWORD *)v3 + 5) + v5 + 1) = CDataStore__ReadBit(&v36);
*(_BYTE *)(*((_DWORD *)v3 + 5) + v5 + 5) = CDataStore__ReadBit(&v36);
*(_BYTE *)(*((_DWORD *)v3 + 5) + v5 + 94) = CDataStore__ReadBit(&v36);
```

Comme vous le constatez, le changement est appréciable et permet d'aller un peu plus vite sur le reversing de paquets contenant des GUIDs de joueur. Il permet également de lire le code de façon itérative plutôt que de vérifier que certaines optimisations du compilateur n'aient eu lieu (par exemple, lire plusieurs données du paquet puis les écrire dans un ordre différent nous a posé un certain nombre de problèmes sur la 5.1 à l'époque).

Un autre petit changement que nous avions prévu mais qui n'est pas si problématique au final, est l'utilisation du même opcode pour plusieurs systèmes différents. Sans rentrer dans les détails, l'opcode est tout de même facilement identifiable selon le contexte d'exécution et il n'est finalement pas si problématique.

Un des changements majeurs imprévus a été la modification du système de fichier gérant les ressources du jeu (modèles 3D, DBCs, etc). Le système est passé de MoPaQ (les fameux MPQs) à CASC (voir ICI) ce qui nécessite de recoder l'ensemble des logiciels d'extraction (dbc, maps, vmaps et mmaps).

Mais, en dehors de ça, le plus gros challenge reste la mise à jour du protocole d'authentification. Pour faire simple, avant Wotlk il n'y avait qu'un seul protocole appelé le GruntLogin. Puis, de Wotlk à MoP, Blizzard a intégré le protocole BattleNet tout en maintenant le protocole GruntLogin. Jusqu'à ce jour, tous les serveurs privés tournent donc sur ce protocole et n'utilisent aucunement le protocole BattleNet. A présent, avec WoD, le GruntLogin n'existe plus et seul le protocole BattleNet est utilisable. Il faut donc reverse l'ensemble du protocole BattleNet bien plus complexe (la DLL de BattleNet est connue pour être une des plus obfusquées)

Quelques sandbox WoD existent déjà sur Internet, mais elles passent habilement l'écran de login. En réalité, ils considèrent toute personne qui tente de se connecter comme authentifiée à un seul et unique compte. Cela a l'avantage de pouvoir rapidement annoncer qu'une sandbox est disponible, mais seule 1 personne peut se connecter en même temps et la protection des comptes joueurs est impossible (puisque chacun peut se faire passer pour un autre)

Au final, le protocole BattleNet sera certainement le plus grand défi proposé à l'émulation mondiale et nuls doutes que cette épine saura causer de nombreux problèmes à qui souhaite ouvrir un serveur Warlords of Draenor.

Voilà qui clôture ce guide des protections de Blizzard à l'encontre des serveurs privés. Vous remarquerez dans mes écrits une once d'énervement (oh, si peu :x) mais avec ces explications, j'espère que vous serez un peu plus attentifs à la complexité du système et aux raisons pour lesquelles très peu de serveurs ouvrent leurs portes avec la dernière version en date. J'espère également que vous comprendrez pourquoi il n'est plus possible de suivre aussi rapidement l'officiel par rapport à l'époque Burning Crusade/Wotlk.

J'espère que ce guide vous aura plu et vous permettra de prendre conscience de la tâche colossale qui se cache réellement derrière un serveur privé, et que ce n'est pas juste une histoire de contenu pur (qui s'avère finalement être assez simple compte tenu de la difficulté principale qu'est le protocole de communication). ~~Comme d'habitude, je reste ouvert à toutes les remarques et appréciations constructives~~
