



# Malware (logiciel côté client)

## La structure de la solution de développement

La solution de développement est découpée en projets, permettant ainsi une séparation des responsabilités.

- **Injector**: bibliothèque chargée dynamiquement<sup>1</sup> (création d'une DLL) qui est le programme injectant les dépendances du malware ainsi que le malware lui-même.
- **MalwareEntryPoint**: bibliothèque chargée dynamiquement (création d'une DLL) qui correspond au point d'entrée du malware, utilisant le framework Qt. Ce dernier fait appelle au projet *SharedKernel*.
- **SharedKernel**: bibliothèque chargée statiquement<sup>2</sup> (sans DLL) partagée entre plusieurs autres projets de la solution qui contient le code du malware, utilisant le framework Qt. Le but de cette séparation est que cette bibliothèque n'est pas couplé à un système d'exploitation, contrairement aux projets créant des DLL.
- **DebugUi**: application graphique permettant d'exécuter le malware en faisant appelle au projet *SharedKernel*. Il permet de contrôler son activité et de voir les messages de debug de ce dernier.

## Les bibliothèques de liens dynamiques (DLL)

Une bibliothèque de liens dynamiques est un fichier contenant un ensemble de fonctions logicielles. Dans Windows, ces fichiers possèdent l'extension "DLL" et possèdent un point d'entrée, tout comme les programmes traditionnels<sup>3</sup>. Les fichiers DLL peuvent être attachés à un processus au démarrage ou dès que le processus en a besoin. Il est aussi possible d'injecter une bibliothèque dans un processus manuellement. Ainsi, le code contenu dans une DLL peut être exécuté dans l'espace mémoire d'un processus donné. Il s'agit de la façon dont le malware développé se propage dans les processus du système d'exploitation.

## Le mécanisme d'injection

Lorsque la solution de développement est compilée, deux bibliothèques (DLL) sont créées:

- Injector.dll
- MalwareEntryPoint.dll

---

<sup>1</sup> Dynamic library: <https://msdn.microsoft.com/en-us/library/ms235636.aspx>

<sup>2</sup> Static library: <https://msdn.microsoft.com/en-us/library/ms235627.aspx>

<sup>3</sup> DLL Entry-Point: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms682596\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682596(v=vs.85).aspx)

Le fichier *injector.dll* est injecté dans un processus. Ce dernier charge les dépendances nécessaire au framework Qt (une suite de fichiers bibliothèques) puis injecte le fichier *MalwareEntryPoint.dll*. Il est nécessaire d'utiliser deux fichiers car *MalwareEntryPoint.dll* possède des dépendances qui doivent déjà se trouver chargées dans le processus. Ces dernières sont alors chargées par *injector.dll*. Le projet *SharedKernel* qui est statiquement compilé avec le fichier *MalwareEntryPoint.dll* contient le code du malware.

## Le déploiement du malware

Le malware possède plusieurs dépendances qu'il est nécessaire d'installer et de configurer.

### Microsoft Visual C++ 2015 Redistributable

Le compilateur C/C++ choisi pour ce projet est celui de Visual Studio, car il est facile de le mettre en place et seul Windows est actuellement ciblé par le malware. Il serait tout à fait possible d'utiliser un autre compilateur si nécessaire, comme Qt est multi plateformes. Le logiciel a donc besoin de dépendances fournies par Microsoft, nommées [Microsoft Visual C++ 2015 Redistributable](#). Il serait possible de fournir ces dépendances directement avec le malware (il s'agirait d'une collection de DLLs), mais pour ce cas d'étude, il est plus simple d'effectuer une installation standard.

### Installation du malware

Le malware ainsi que ces DLLs doivent être placé dans un dossier préalablement choisi avant sa compilation, car il est nécessaire de renseigner le chemin de certaines dépendances lors de sa compilation. Ce dossier doit aussi être renseigné dans la variable d'environnement PATH afin que l'injection des DLLs puissent se réaliser sans spécifier leur chemin d'accès. Dans ce cas d'étude, le malware est installé dans le dossier C:\Malware et les dépendances sont situées dans C:\Qt. En pratique, les deux entités seraient localisées dans C:\Windows\System32 afin d'être plus discret. Ainsi la configuration suivante a été appliquée:

- Création du dossier C:\Qt avec les dépendances Qt puis enregistrer ce dossier dans la variable d'environnement PATH.
- Création du dossier C:\Malware avec les binaires du malware puis enregistrer ce dossier dans la variable d'environnement PATH.

Bot

Creating and Using a Dynamic Link Library (C++):

<https://msdn.microsoft.com/en-us/library/ms235636.aspx>

# Communication entre le serveur et le malware

## Les messages

Le malware communique avec le serveur à l'aide d'une API REST. Ce dernier envoie des messages à interval régulier afin de maintenir le serveur à jour et permet de récupérer des actions à effectuer. Voici deux tableaux résumant les messages que ces deux entités échangent.

Opcodes envoyés par le serveur		
Opcode	Définition	Payload
0x100	AskInformation Demande d'information au malware afin de récupérer des informations sur l'ordinateur infecté.	-
0x110	StartAttack Informe le malware de démarrer une attaque DOS.	Id (attack), IPv4, port, méthode d'attaque.
0x120	StopAttack Informe le malware d'arrêter une attaque DOS.	Id (attack).
0x666	Wipe Désinstalle le malware de l'ordinateur infecté.	-

Opcodes envoyés par le malware		
Opcode	Définition	Payload
0x10	Hello Message régulièrement envoyé au serveur afin que le serveur sache que le malware est toujours connecté.	Id (adresse MAC).
0x20	GiveInformation Informe le serveur des caractéristiques de l'ordinateur infecté..	Id (adresse MAC), nom de l'ordinateur, CPU, version de l'OS.



Voici un exemple d'une réponse du serveur pour démarrer une attaque en JSON:

```
{
  "opcode": 100,
  "payload":
  {
    "id": 1,
    "ipv4": "127.0.0.1",
    "port": 80,
    "methodOfAttack": "SYN"
  }
}
```