

Rapport Malware

1) Fonctionnement global du programme

Notre code lit l'entrée via argv. Si elle ne vérifie pas le format exigée, c'est à dire si l'entrée n'est pas sous la forme de chaînes de caractères dans [a-f0-9]* avec une longueur plus petite que 64, le code renvoie un message disant que la chaîne de caractère n'est pas valide:

```
C:\Documents and Settings\Administrateur\Mes documents\Visual Studio 2010\Projec
ts\test\Release>test.exe aaaaaaaaaRRRR
IlSembleraitQueLentreeNeSoitPasValide
```

Ensuite, si l'entrée vérifie le format mais ne correspond pas à la clé, l'entrée est affichée sur la console. Si l'entrée correspond à la clé, un message explicite est affiché sur la console:

```
C:\Documents and Settings\Administrateur\Mes documents\Visual Studio 2010\Projec
ts\test\Release>test.exe 7c3a1f2e0d9b5a8c7f6e9b0c2d4a3b1e8d6f0a2
IlSembleraitQueVousAyezTrouveLaCleBRAVO
```

De plus, si un débbugger est détecté, le programme s'arrête.

2) Techniques utilisées

Dans cette partie, nous allons détailler les différentes méthodes utilisées dans le code.

a) Chiffrement des chaînes de caractères présentes dans le code

Tout d'abord, toutes les chaînes de caractères ont été chiffrées. En effet, les messages indiquant que l'entrée n'est pas valide ou que l'entrée correspond à la clé secrète ont été chiffrée à l'aide d'un chiffrement de vigenère avec les clés suivantes: utlxzqfdgoyrvawjcmhk et fbsjbfqjkbqllfzaf.

IlSembleraitQueVousAyezTrouveLaCleBRAVO -> NmKnngbnbbnjBzjVnzxBqnaYhxewjBIHqeAWFWG
IlSembleraitQueLentreeNeSoitPasValide -> CeDblrqhxogkLuaUgzabyxYbRenwVoqMvlemg

De plus, toutes les chaînes de caractère comprenant les messages chiffrées et les clés utilisées pour les chiffrements, hormis un leurre qui a été écrit normalement, ont été écrites dans le code ont été écrites sous forme hexadécimale :

```
char z[64]:
char j[64] = {'\x6e', '\x68', '\x75', '\x67', '\x77', '\x6c', '\x79', '\x72', '\x73', '\x61', '\x71', '\x78', '\x6a', '\x76', '\x62', '\x6d', '\x66', '\x63', '\x6f', '\x70', '\x7a', '\x74', '\x69', '\x65', '\x64', '\x6b', '\x61', '\x60', '\x7e', '\x7f', '\x5c', '\x5d', '\x5e', '\x5f', '\x40', '\x41', '\x42', '\x43', '\x44', '\x45', '\x46', '\x47', '\x48', '\x49', '\x4a', '\x4b', '\x4c', '\x4d', '\x4e', '\x4f', '\x50', '\x51', '\x52', '\x53', '\x54', '\x55', '\x56', '\x57', '\x58', '\x59', '\x5a', '\x5b', '\x5c', '\x5d', '\x5e', '\x5f', '\x60', '\x61', '\x62', '\x63', '\x64', '\x65', '\x66', '\x67', '\x68', '\x69', '\x6a', '\x6b', '\x6c', '\x6d', '\x6e', '\x6f', '\x70', '\x71', '\x72', '\x73', '\x74', '\x75', '\x76', '\x77', '\x78', '\x79', '\x7a', '\x7b', '\x7c', '\x7d', '\x7e', '\x7f', '\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89', '\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93', '\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d', '\x9e', '\x9f', '\xa0', '\xa1', '\xa2', '\xa3', '\xa4', '\xa5', '\xa6', '\xa7', '\xa8', '\xa9', '\xaa', '\xab', '\xac', '\xad', '\xae', '\xaf', '\xb0', '\xb1', '\xb2', '\xb3', '\xb4', '\xb5', '\xb6', '\xb7', '\xb8', '\xb9', '\xba', '\xbb', '\xbc', '\xbd', '\xbe', '\xbf', '\xc0', '\xc1', '\xc2', '\xc3', '\xc4', '\xc5', '\xc6', '\xc7', '\xc8', '\xc9', '\xca', '\xcb', '\xcc', '\xcd', '\xce', '\xcf', '\xd0', '\xd1', '\xd2', '\xd3', '\xd4', '\xd5', '\xd6', '\xd7', '\xd8', '\xd9', '\xda', '\xdb', '\xdc', '\xdd', '\xde', '\xdf', '\xe0', '\xe1', '\xe2', '\xe3', '\xe4', '\xe5', '\xe6', '\xe7', '\xe8', '\xe9', '\xea', '\xeb', '\xec', '\xed', '\xee', '\xef', '\xf0', '\xf1', '\xf2', '\xf3', '\xf4', '\xf5', '\xf6', '\xf7', '\xf8', '\xf9', '\xfa', '\xfb', '\xfc', '\xfd', '\xfe', '\xff'};
char *cipher;

cipher = encrypt(input, SHIFT);
vigenereEnc(cipher, j, z);

char y[64]:
char u[64] = {'\x6e', '\x68', '\x75', '\x67', '\x77', '\x6c', '\x79', '\x72', '\x73', '\x61', '\x71', '\x78', '\x6a', '\x76', '\x62', '\x6d', '\x66', '\x63', '\x6f', '\x70', '\x7a', '\x74', '\x69', '\x65', '\x64', '\x6b', '\x61', '\x60', '\x7e', '\x7f', '\x5c', '\x5d', '\x5e', '\x5f', '\x40', '\x41', '\x42', '\x43', '\x44', '\x45', '\x46', '\x47', '\x48', '\x49', '\x4a', '\x4b', '\x4c', '\x4d', '\x4e', '\x4f', '\x50', '\x51', '\x52', '\x53', '\x54', '\x55', '\x56', '\x57', '\x58', '\x59', '\x5a', '\x5b', '\x5c', '\x5d', '\x5e', '\x5f', '\x60', '\x61', '\x62', '\x63', '\x64', '\x65', '\x66', '\x67', '\x68', '\x69', '\x6a', '\x6b', '\x6c', '\x6d', '\x6e', '\x6f', '\x70', '\x71', '\x72', '\x73', '\x74', '\x75', '\x76', '\x77', '\x78', '\x79', '\x7a', '\x7b', '\x7c', '\x7d', '\x7e', '\x7f', '\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89', '\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93', '\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d', '\x9e', '\x9f', '\xa0', '\xa1', '\xa2', '\xa3', '\xa4', '\xa5', '\xa6', '\xa7', '\xa8', '\xa9', '\xaa', '\xab', '\xac', '\xad', '\xae', '\xaf', '\xb0', '\xb1', '\xb2', '\xb3', '\xb4', '\xb5', '\xb6', '\xb7', '\xb8', '\xb9', '\xba', '\xbb', '\xbc', '\xbd', '\xbe', '\xbf', '\xc0', '\xc1', '\xc2', '\xc3', '\xc4', '\xc5', '\xc6', '\xc7', '\xc8', '\xc9', '\xca', '\xcb', '\xcc', '\xcd', '\xce', '\xcf', '\xd0', '\xd1', '\xd2', '\xd3', '\xd4', '\xd5', '\xd6', '\xd7', '\xd8', '\xd9', '\xda', '\xdb', '\xdc', '\xdd', '\xde', '\xdf', '\xe0', '\xe1', '\xe2', '\xe3', '\xe4', '\xe5', '\xe6', '\xe7', '\xe8', '\xe9', '\xea', '\xeb', '\xec', '\xed', '\xee', '\xef', '\xf0', '\xf1', '\xf2', '\xf3', '\xf4', '\xf5', '\xf6', '\xf7', '\xf8', '\xf9', '\xfa', '\xfb', '\xfc', '\xfd', '\xfe', '\xff'};
char m[] = {'\x4e', '\x6d', '\x4b', '\x6e', '\x6e', '\x67', '\x62', '\x6e', '\x62', '\x62', '\x6e', '\x6a', '\x42', '\x7a', '\x6a', '\x56', '\x6e', '\x7a', '\x7b', '\x42', '\x71', '\x6e', '\x61', '\x59', '\x68', '\x7b', '\x65', '\x60', '\x7e', '\x7f', '\x5c', '\x5d', '\x5e', '\x5f', '\x40', '\x41', '\x42', '\x43', '\x44', '\x45', '\x46', '\x47', '\x48', '\x49', '\x4a', '\x4b', '\x4c', '\x4d', '\x4e', '\x4f', '\x50', '\x51', '\x52', '\x53', '\x54', '\x55', '\x56', '\x57', '\x58', '\x59', '\x5a', '\x5b', '\x5c', '\x5d', '\x5e', '\x5f', '\x60', '\x61', '\x62', '\x63', '\x64', '\x65', '\x66', '\x67', '\x68', '\x69', '\x6a', '\x6b', '\x6c', '\x6d', '\x6e', '\x6f', '\x70', '\x71', '\x72', '\x73', '\x74', '\x75', '\x76', '\x77', '\x78', '\x79', '\x7a', '\x7b', '\x7c', '\x7d', '\x7e', '\x7f', '\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89', '\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93', '\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d', '\x9e', '\x9f', '\xa0', '\xa1', '\xa2', '\xa3', '\xa4', '\xa5', '\xa6', '\xa7', '\xa8', '\xa9', '\xaa', '\xab', '\xac', '\xad', '\xae', '\xaf', '\xb0', '\xb1', '\xb2', '\xb3', '\xb4', '\xb5', '\xb6', '\xb7', '\xb8', '\xb9', '\xba', '\xbb', '\xbc', '\xbd', '\xbe', '\xbf', '\xc0', '\xc1', '\xc2', '\xc3', '\xc4', '\xc5', '\xc6', '\xc7', '\xc8', '\xc9', '\xca', '\xcb', '\xcc', '\xcd', '\xce', '\xcf', '\xd0', '\xd1', '\xd2', '\xd3', '\xd4', '\xd5', '\xd6', '\xd7', '\xd8', '\xd9', '\xda', '\xdb', '\xdc', '\xdd', '\xde', '\xdf', '\xe0', '\xe1', '\xe2', '\xe3', '\xe4', '\xe5', '\xe6', '\xe7', '\xe8', '\xe9', '\xea', '\xeb', '\xec', '\xed', '\xee', '\xef', '\xf0', '\xf1', '\xf2', '\xf3', '\xf4', '\xf5', '\xf6', '\xf7', '\xf8', '\xf9', '\xfa', '\xfb', '\xfc', '\xfd', '\xfe', '\xff'};
char x[] = {'\x32', '\x70', '\x30', '\x6f', '\x30', '\x61', '\x33', '\x68', '\x33', '\x72', '\x34', '\x6f', '\x32', '\x6e', '\x37', '\x69', '\x38', '\x64', '\x39', '\x72', '\x34', '\x75', '\x37', '\x69', '\x31', '\x72', '\x35', '\x37', '\x65', '\x30', '\x62', '\x38', '\x63', '\x33', '\x61', '\x66', '\x32', '\x34', '\x64', '\x32', '\x61', '\x31', '\x65', '\x39', '\x64', '\x36', '\x62', '\x35', '\x33', '\x61', '\x39', '\x66', '\x30', '\x61', '\x32', '\x67', '\x69', '\x6d'};
char ar[] = {'\x37', '\x65', '\x30', '\x62', '\x38', '\x63', '\x33', '\x61', '\x66', '\x32', '\x34', '\x64', '\x32', '\x61', '\x31', '\x65', '\x39', '\x64', '\x36', '\x62', '\x35', '\x33', '\x61', '\x39', '\x66', '\x30', '\x61', '\x32', '\x67', '\x69', '\x6d'};
char ar2[] = "c6b731e8f92d5a74eab9c9e1d0d26673f69d4";
```

b) Chiffrement de l'entrée

L'entrée qui est lue via argv est lue en deux étapes. Tout d'abord, l'entrée est chiffrée un chiffrement de César, qui consiste à décaler chaque lettre d'un certain nombre de positions dans l'alphabet. La valeur de décalage est déterminée par un entier initialement fixé à 3 et qui est incrémenté selon la position du caractère.

Ensuite, l'entrée est chiffrée aussi à l'aide d'un chiffrement de vigenère en utilisant la clé suivante : nhugwlyrsaqxjvbmfcopztiedkabifroscdzjxqel.

La comparaison est faite avec la clé chiffrée en utilisant les mêmes méthodes qui est : 2p0o0a3h3r4o2n7i8d9r4u7i1r5h6o6c5r5s1n5.

c) Anti-debugger et obfuscation de fonction

Nous avons utilisé trois anti-debugger en vérifiant la présence d'un debugger à l'aide de : IsDebuggerPresent(), CheckRemoteDebuggerPresent(GetCurrentProcess(),&res) et de PEB.

Les fonctions IsDebuggerPresent, CheckRemoteDebuggerPresent et GetCurrentProcess ont été cachées en utilisant leur différence de position avec memcmp pour les deux premières et en utilisant la différence de position avec scanf pour GetCurrentProcess. Pour ceci, nous avons créé trois types de fonctions :

```
typedef int (*type_debug) ();  
typedef BOOL (*type_check) (HANDLE ,PBOOL,...);  
typedef HANDLE (*type_process) ();
```

Dans un autre programme, nous avons relevé la valeur de ces différences et nous les initialisons en utilisant des fonctions qui réalisent les instructions dans un tableau. On définit la aussi un nouveau type de fonction et nos fonctions cachées sont obtenu comme ceci :

```
char nb_debug[] = {'\xb8','\x5c','\x86','\xd7','\x03','\xc3'};  
type_int o =(type_int) &nb_debug;  
int decal_debug = o();  
  
char nb_check[] = {'\xb8','\xfd','\x45','\xda','\x03','\xc3'};  
type_int o1 =(type_int) &nb_check;  
int decal_check = o1();  
  
char nb_process[] = {'\xb8','\x31','\x83','\xd0','\x03','\xc3'};  
type_int o2 =(type_int) &nb_process;  
int decal_process = o2();  
  
unsigned int pos_memcmp=(unsigned int) memcmp;  
unsigned int pos_scanf=(unsigned int) scanf;  
  
type_debug h;  
h=(type_debug) (pos_memcmp + decal_debug);  
if (h()){return 0;}  
  
type_check e;  
e=(type_check) (pos_memcmp + decal_check);
```

d) Autre obfuscation de fonctions

Nous avons également caché deux autres fonctions qui sont strcmp et printf en utilisant également la position de memcmp. Nous créons deux autres types de fonctions :

```
typedef int (*type_strcmp) (const char*,const char*,...);  
typedef int (*type_printf) (const char*,...);
```

```
char nb_cmp[] = {'\xb8', '\x93', '\x48', '\x01', '\x00', '\xc3'};
type_int o3 =(type_int) &nb_cmp;
int decal_cmp = o3();

char nb_pr[] = {'\xb8', '\x21', '\xe6', '\x04', '\x00', '\xc3'};
type_int o4 =(type_int) &nb_pr;
int decal_pr = o4();


type_strcmp f;
f=(type_strcmp) (pos_memcmp - decal_cmp);

type_printf g;
g=(type_printf) (pos_memcmp + decal_pr);
```