Laboratoire - ransTroj

Sécurité logicielle bas-niveau

Annen Rayane, Ducommun Hugo, Martins Alexis

HEIG VD

1er novembre 2023

Table des matières

Réponses aux questions

> Q4.1 : Que fait le code au début de main avant l'invocation de la fonction `bruteforce` ? Quel est le but du malware, d'après vous ?

C'est un trojan, il lance dans un autre thread l'exécutable `calc` mais dans le thread principal il effectue le chiffrement des données.

> Q4.2 : Que fait la fonction bruteforce invoquée au début du main ? Quel est le but du malware, d'après vous ?

Elle semble faire une attaque par force brute sur un tableau donné en paramètre jusqu'à trouver `slbrans` qui va retourner une valeur différente de 0 et qui va déclencher le malware.

> Q4.7 : Quelles exceptions prévoit le code de `encrypt_dir` ? Pourquoi ?

- On ne peut pas chiffrer le binaire lui-même : car le programme reste un trojan, il ne faudrait pas le détruire durant le processus.
- On ne peut pas chiffrer les répertoires au dessus : peut-être pour une raison de privilèges (pas le droit de supprimer un dossier en dehors du /home par exemple), le programme étant un ransomware, il faut que les données soient possiblement récupérables afin de donner du sens à une rançon, autrement le programme détruirait le système et finalement pour des raisons de praticité dans le cadre du laboratoire.

> Q7.2 : Commentez le code C dans ghidra, puis renommer et/ou « retyper » les variables locales en conséquence de la réponse à la question 7.1.

```
void encrypt2(FILE *file_ptr)
{
  char *key;
  int const_three;
  byte intermediate_calc;
  int curr_char;
  uint intermediate_calc2;
  byte xor_curr_char;

  generateKey(&key,0x10);
  xor_curr_char = 0;
  intermediate_calc2 = 0;
  while( true ) {
    curr_char = fgetc(file_ptr);
    if (curr_char == -1) break;
    // parcours du fichier à l'endroit
    fseek(file_ptr,-1,1);
    xor_curr_char = xor_curr_char ^ (byte)curr_char;
```

```
19      const_three = 3;
20      // rotation cyclique gauche de 3
21      intermediate_calc = xor_curr_char << 3 | xor_curr_char >> 5;
22      intermediate_calc2 = intermediate_calc2 + 3 & 0xe;
23      curr_char = (int)(char)((byte)curr_char ^ *(byte *)((int)&key + intermediate_calc2 +
            1));
24      xor_curr_char = intermediate_calc;
25      fputc(curr_char,file_ptr);
26    }
27    fseek(file_ptr,0,0);
28    while( true ) {
29      curr_char = fgetc(file_ptr);
30      if (curr_char == -1) break;
31      // parcours du fichier à l'endroit
32      fseek(file_ptr,-1,1);
33      curr_char = (int)(char)((byte)curr_char ^ xor_curr_char);
34      xor_curr_char = xor_curr_char - 2;
35      fputc(curr_char,file_ptr);
36    }
37    fputc((int)(char)xor_curr_char,file_ptr);
38    return;
39  }
```

Q8.1 : Expliquez l'algorithme de déchiffrement que vous avez implémenté. Indiquez les formules mathématiques appliquées par chaque méthode de déchiffrement ainsi que la logique du flux d'instruction qui permet d'appliquer l'une ou l'autre.

Nous utilisons un algorithme de parcours du répertoire récursif récupéré du programme de base.

- Pour chaque fichier : nous vérifions que ce n'est pas le programme lui-même ou bien le répertoire du dessus. Les exceptions sont précisées dans la question 4.7.
- Si le fichier est valide : on récupère la longueur l du nom du fichier puis nous appliquons la formule : $\ell \,\&\, 3$, si nous obtenons 0, la fonction `decrypt0` est appelée, si nous obtenons 1 `decrypt1` et `decrypt2` sinon. Pour chaque fichier, nous récupèrons dans buffer alloué dynamiquement le contenu du fichier puis nous travaillons exclusivement dessus.
- Une fois le buffer déchiffré on créé un nouveau fichier ayant pour nom l'ancien nom du fichier puis `_decrypted`.

Algorithme de déchiffrement de `decrypt0`:

- pour chaque caractère c du buffer
  - si $c \,\&\, 1 = 0$
    - $c := c - 0xaa$
  - sinon
    - $c := c + 0xba$

Algorithme de déchiffrement de `decrypt1`:

- On initialise $j = 0$
- pour chaque caractère c du buffer
  - $c := c \oplus j$
  - $c := c >> 2 \,|\, c << 6$, rotation cyclique droite de 2
  - $j := c << 4 \,|\, c >> 4$, rotation cyclique gauche de 4

Algorithme de déchiffrement de `decrypt2`

- On initialise un tableau clef nommé k :

```
{ 0xee, 0x9a, 0x60, 0x1c, 0xd4, 0xfc, 0x04, 0x6a,
    0x05, 0xfe, 0xc4, 0x33, 0x5d, 0xa0, 0xc2, 0x8b };
```

- On note l le dernier caractère du buffer.
- Pour chaque caractère c du buffer lu dans le sens inverse et en commençant à partir de l'avant dernier caractère :
  - $l := l + 2$
  - $c := c \oplus l$
- On initialise $j = 0$.
- Pour chaque caractère c du buffer :
  - $j := j + 3 \,\&\, 15$
  - $j \oplus k[j + 1]$