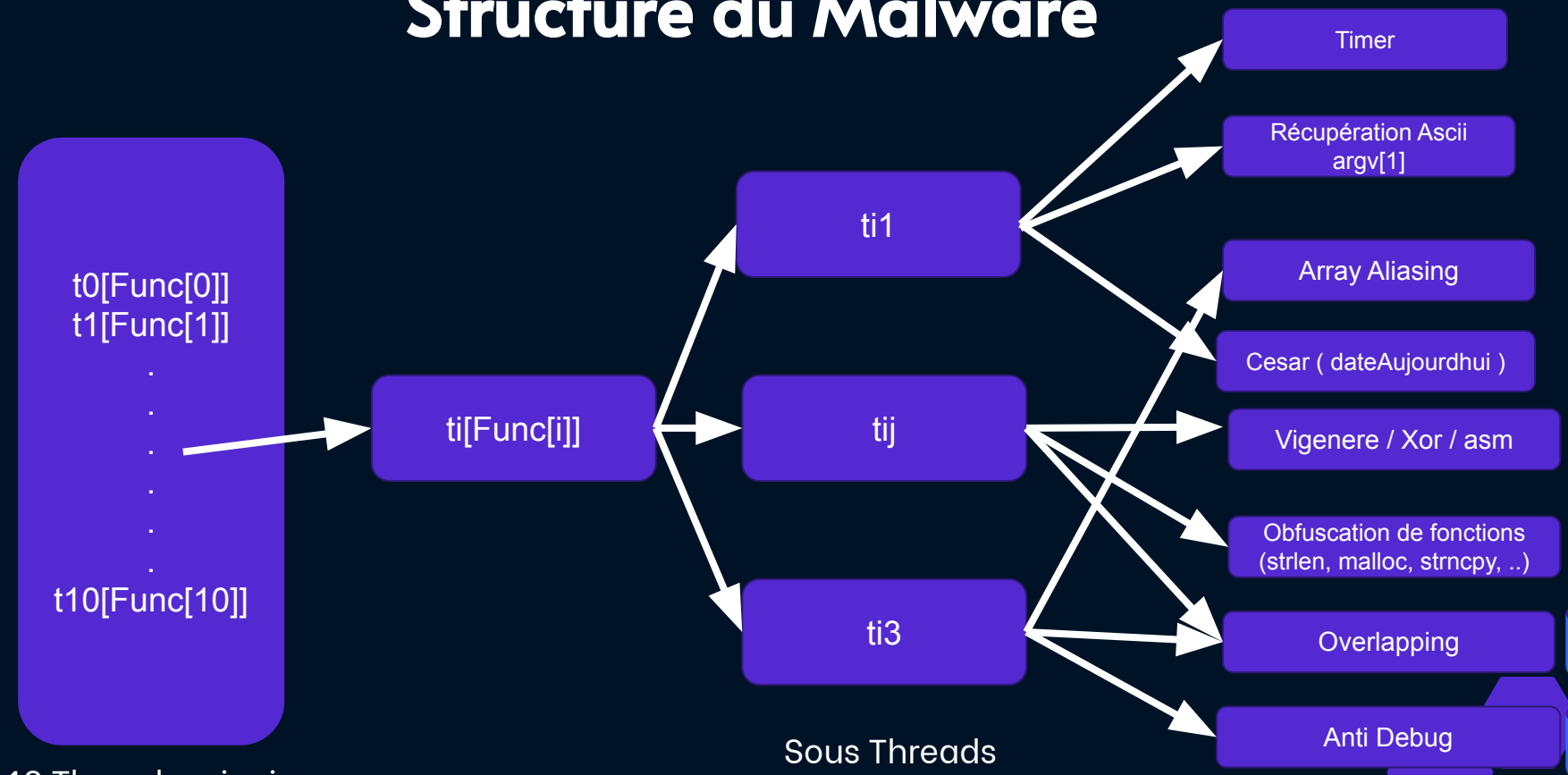
Elaboration Malware

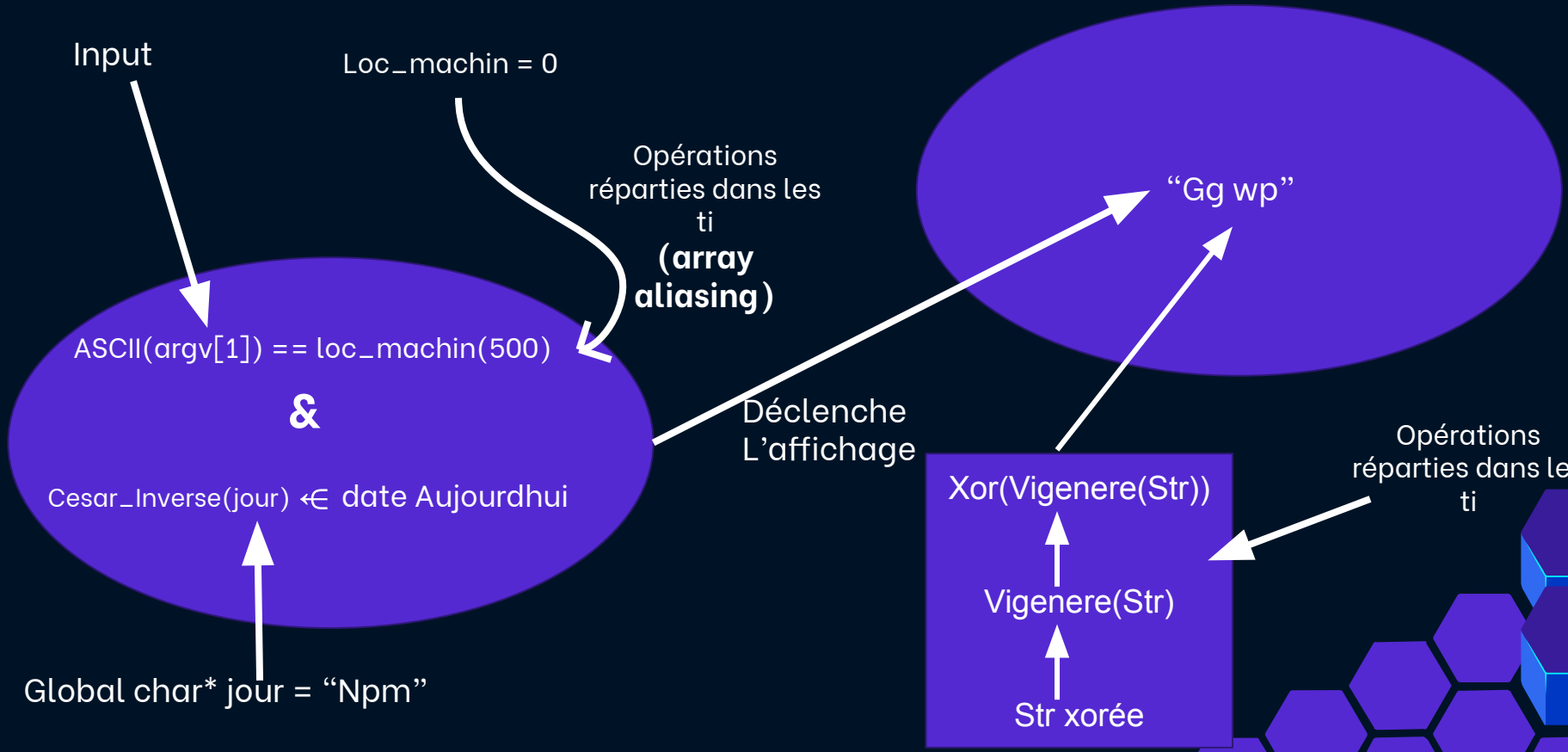
Structure du Malware



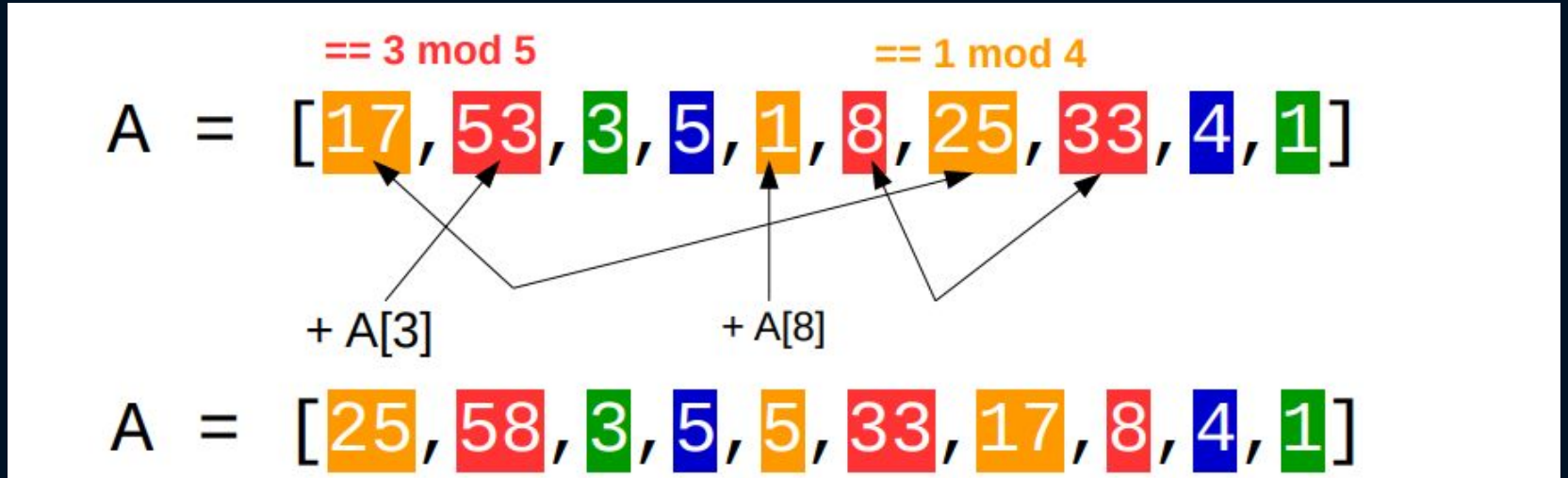
10 Threads principaux
(wait entre chacun)

Sous Threads

Vue d’ensemble du Malware



Loc_Machin(500) : ARRAY ALIASING



- 3 == A[1]%A[4] A[2] == A[5]%A[4]
- 1 == A[5]%A[8] A[9] == A[0]%A[8]

OK

Loc_Machin(500) : ARRAY ALIASING

Source : www.synacktiv.com

```
def add(a, b){
    count = a
    while (b > 0){
        count += 1
        b -= 1
    }
    return count
}
add(2,3)
```

```
0 == A[5]%A[3] - A[1]%A[3]
1 == A[4]%A[8]
1 == A[0]%A[8]
```

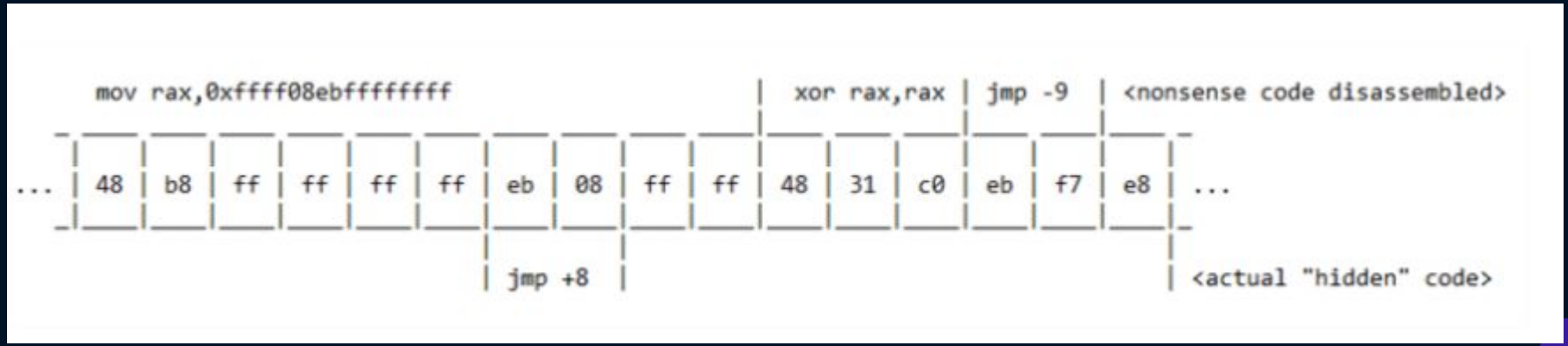
```
A = [17,53,3,5,1,8,25,33,4,1]
def add(a, b){
    count = a
    while (b > A[5]%A[3]-A[1]%A[3]){
        count += A[4]%A[8]
        b -= A[0]%A[8] }
    return count
}
add(2,3)
```

```
A = [17,53,3,5,1,8,25,33,4,1]
def add(a, b){
    A[0]=A[4]
    count = a
    while (b > A[5]%A[3]-A[1]%A[3]){
        A[4] += A[0]%A[8]
        count += A[4]%A[8]
        b -= A[0]%A[8] }
    return count
}
A[5]=(A[1]+A[7])%A[4]+A[7]
add(2,3)
```

XOR à Partir d'un Tableau

```
/*** Global variable ***/
int loc_721D581[] = {51,2,3,4,5};
char loc_72DD581[] = {'n'^51,'k'^52,' '^53,'h'^54,'d'^55, '\x00'^56}; //pass
loc_72000081();
__asm{
        lea eax, loc_72DD581
        mov ecx, 6
        lea esi, loc_721D581
        mov dl, byte ptr [esi]
debut_boucle:
        sub ecx, 1    ; dec ecx
        mov bl, byte ptr [eax]
        xor bl, dl
        mov byte ptr [eax], bl
        add eax,1    ; inc eax
        add dl, 1
        cmp ecx, 0
        jne debut_boucle
    }
```

Overlapping Instruction



Overlapping Instruction

```
1420        BOOL loc_4017896 = IsDebuggerPresent();
1421
1422
1423        __asm {
1424                _emit 0x48
1425                _emit 0xb8
1426                _emit 0xff
1427                _emit 0xeb
1428                _emit 0x07
1429                _emit 0xff
1430                _emit 0x48
1431                _emit 0x31
1432                _emit 0xc0
1433                _emit 0xeb
1434                _emit 0xf8
1435                _emit 0xe8
1436            }
1437        CheckRemoteDebuggerPresent(loc_401780351, &loc_401560283);
1438
1439        PEB *loc_401778245;
1440
1441        __asm{
1442            mov eax, fs:[0x30]
1443            mov loc_401778245, eax
1444        }
1445
```

```
1446        if (loc_4017896 || loc_401560283 || loc_401778245->BeingDebugged) {
1447
1448            exit(-1);
1449        }
1450        loc_401B581 = strstr (loc_786D8ABC, loc_401BA11.c_str());
1451        if (loc_401B581 != NULL){
1452            __asm {
1453                _emit 0x48
1454                _emit 0xb8
1455                _emit 0xff
1456                _emit 0xeb
1457                _emit 0x07
1458                _emit 0xff
1459                _emit 0x48
1460                _emit 0x31
1461                _emit 0xc0
1462                _emit 0xeb
1463                _emit 0xf8
1464                _emit 0xe8
1465            }
1466            cout << loc_720DA81 ;
1467            cout <<endl;
1468        }
```

Renommage des variables

```
string dateAujourdhui = "Std";

//variable to get access to the date
time_t now;
char * findDay;

//global value pour le thread leetcode
const int mod = 1000000007;


// variable pour avoir la somme des valeurs ascii de l'entrée
int count = 0;

/*
Type to obfuscate the 500 condition
*/
int lavaleur = 66;
int kk=0;

//0 == 8%5 - 53%5
// 0 == A[5]%A[11] - A[10]%A[11]
int A[] = {17,4,3,0,1,8,25,120,4,100, 53, 5};
int D[] = {51,2,3,4,5};
//string pass = {'g'^51,'g'^52,' '^53,'w'^54,'p'^55, '\x00'^56};
char pass[] = {'n'^51,'k'^52,' '^53,'h'^54,'d'^55, '\x00'^56};
//pour vigenere
string s1 = "hello";
string passString = "";
//calcul la somme des 4 premiers elements de A[]
int loc_401AA7(){
        int glob_402B8=0;
        for(;kk<4; kk++){
                glob_402B8 += A[kk];
        }
        return glob_402B8;
}
```

Utilisation de la convention de Ida du renommage de variables (loc_bidule)

```
string loc_401BA11 = "Vdc"; //dateAujourdhui

time_t loc_401BA81; //now
char * loc_401B581; //findDay

const int loc_701B581 = 1000000007; //mod
int loc_781B581 = 0; //count

int loc_721B581 = 66; //lavaleur
int kk=0;
int loc_7216581[] = {17,4,3,0,1,8,25,120,4,100, 53, 5}; //A
int loc_721D581[] = {51,2,3,4,5}; //D
char loc_72DD581[] = {'n'^51,'k'^52,' '^53,'h'^54,'d'^55, '\x00'^56}; //pass
string loc_72DDA81 = "hello"; //s1
string loc_720DA81 = ""; //passString
int loc_401AA7(){
        int glob_402B8=0;
        for(;kk<4; kk++){
                glob_402B8 += loc_7216581[kk];
        }
        return glob_402B8;
}
```

“Timer”

```
DWORD WINAPI loc_786D8ACA(LPVOID param){ //timer
	SYSTEMTIME loc_786D8AC8; //time
	GetSystemTime(&loc_786D8AC8);
    loc_72000081();

	int loc_786D8ACC = 5; //nb
	int loc_786D8ACD = 0; //termine
    char loc_776A8BA9[] = {'\x65','\x66','\x75','\x45','\x54','\x67'};
    for(int loc_776A8AA9=0; loc_776A8AA9<strlen(loc_776A8BA9); loc_776A8AA9++) {

	if(loc_776A8BA9[loc_776A8AA9] == '\x45') {
		loc_776A8BA9[loc_776A8AA9] = '\x32';
	}
	else if(loc_776A8BA9[loc_776A8AA9] == '\x89'){
		loc_776A8BA9[loc_776A8AA9] = '\x56';
	}
	else if(loc_776A8BA9[loc_776A8AA9] == '\x56') {
		loc_776A8BA9[loc_776A8AA9] = '\x88';
	}
	else if(loc_776A8BA9[loc_776A8AA9] == '\x65'){
		loc_776A8BA9[loc_776A8AA9] = '\x76';
	}
	else if(loc_776A8BA9[loc_776A8AA9] == '\x96') {
		loc_776A8BA9[loc_776A8AA9] = '\x55';
	}
    }
	while(loc_786D8ACD==0) {
		SYSTEMTIME loc_786D8AD0; //newtime
		GetSystemTime(&loc_786D8AD0);
		if((loc_786D8AD0.wHour != loc_786D8ACB.wHour) && (loc_786D8ACB.wMinute != 59) && (loc_786D8ACB.wSecond < 58)){
			loc_786D8ACD = 1;
		}
		else if((loc_786D8ACB.wDay != loc_786D8AD0.wDay) && (loc_786D8ACB.wHour != 23) && (loc_786D8ACB.wMinute != 59) && (loc_786D8ACB.wSecond < 58)){
			loc_786D8ACD = 1;
		}
		else if((loc_786D8ACB.wMinute != loc_786D8AD0.wMinute) && (loc_786D8ACB.wSecond < 60 - loc_786D8ACC)){
			loc_786D8ACD = 1;
		}
		else if((loc_786D8ACB.wSecond < 60-loc_786D8ACC) && (loc_786D8AD0.wSecond - loc_786D8ACB.wSecond >= loc_786D8ACC)){
			loc_786D8ACD = 1;
		}
		else if((loc_786D8ACB.wSecond == 60-loc_786D8ACC) && (loc_786D8AD0.wSecond >= 0)){
			loc_786D8ACD = 1;
		}
		else if((loc_786D8ACB.wSecond > 60-loc_786D8ACC) && ((loc_786D8AD0.wMinute == loc_786D8ACB.wMinute + 1) ||
		 (loc_786D8AD0.wMinute == 0 && loc_786D8ACB.wMinute==59)) && (loc_786D8AD0.wSecond >= loc_786D8ACB.wSecond-(60-loc_786D8ACC))){
			loc_786D8ACD = 1;
		}
    }

    DWORD loc_786D8AD1 = WaitForSingleObject(loc_786D8AC9, INFINITE); //wait
    loc_786D8AC8 = 1;
    ReleaseMutex(loc_786D8AC9);
    return 0;
```

- Vérifie l’heure et la date
- Si plus que 5 secondes alors le programme s’arrête

Anti Debug

```
int loc_401560283 = 0;

HANDLE loc_401780351 = GetCurrentProcess();

BOOL loc_4017896 = IsDebuggerPresent();

CheckRemoteDebuggerPresent(loc_401780351, &loc_401560283);

PEB *loc_401778245;

__asm{
    mov eax, fs:[0x30]
    mov loc_401778245, eax
}

if (loc_4017896 || loc_401560283 || loc_401778245->BeingDebugged) {
    exit(-1);
}
```

Fontions obfusquées

Ex: malloc (avec overlapping entre et décalage pour l'adresse de départ)

```
__asm {
    _emit 0x48
    _emit 0xb8
    _emit 0xff
    _emit 0xeb
    _emit 0x07
    _emit 0xff
    _emit 0x48
    _emit 0x31
    _emit 0xc0
    _emit 0xeb
    _emit 0xf8
    _emit 0xe8
}
loc_72000001 = (loc_7200001)(loc_401DD1[5] + loc_401DD1[7]);
__asm {
    _emit 0x48
    _emit 0xb8
    _emit 0xff
    _emit 0xeb
    _emit 0x07
    _emit 0xff
    _emit 0x48
    _emit 0x31
    _emit 0xc0
    _emit 0xeb
    _emit 0xf8
    _emit 0xe8
}
char* loc_725D687 = (char*) loc_72000001(200*sizeof(char)); //arg
```

Autre thread (toujours malloc)

```
loc_401DD1[5] = 0x78A5BF12;
while(loc_786D8AC1 > 0) {
    if (loc_786D8AC1 & 1) {
        loc_786D8AC1 = (loc_786D8AC1 * loc_786D8AC5) % loc_701B581;

    }
    loc_786D8AC5 *= loc_786D8AC5;
    loc_786D8AC5 %= loc_701B581;
    loc_786D8AC1 /= 2;
}
loc_401DD1[7] = 0x54321;
```

```
typedef int (*loc_720DA01)(char*, char*, int); //cc
typedef char* (*loc_7200A01)(time_t * loc_720D581 /*timer*/); //printff
typedef void* (*loc_7200001) (size_t loc_72000041 /*size*/); //strLen
```

Vigenère / César

-un simple Algo de César obfusqué avec la création de fichiers, l'Overlapping Instruction et le Code de "bourrage".

-Vigenère Algo avec une clé xorée auquelle on a appliqué un César, avec également les mêmes obfuscation que le césar.

```
int loc_401AC89[] = {97, 26, 65, 69}; //for Cesar and Vigenere Algo
```

Code Inutile pour obfusquer

⇒ Création de fichiers contenant des appels à divers appels de librairie non cachés (augmentation de l'arbre d'appels de fonctions)

⇒ Variables globales faisant du Array Aliasing sur des valeurs random, parfois le même tableau mais sur des cases du tableaux non utilisées pour la condition.

⇒ Threads contenant des fonctions inutiles, faisant des calculs aléatoires.

⇒ Du "isDebuggerPresent" Aléatoirement répartis dans le code, alors que les autres anti debug + discret sont placés à des endroits stratégiques.

⇒ Succession de conditions if avec des références aux tableaux globaux (ou pas)