Decompiler User Survey 2

April, 2021

1 User Survey 2

1.1 Self Assessment

Did you take part in the decompiler survey last year?

○ Yes

○ No

How much time (approx.) did you spend working with C code?

○ None

○ A few hours

○ Several days

○ More than a year

○ On a regular basis

How much time did you spend reversing executables before?

○ None

○ A few hours

○ Several days

○ More than a year

○ On a regular basis

1.2 Comprehension

Please consider the following decompiled function:

```
int sub_401000() {
    char i;
    char * var_0;
    char var_1;
    char var_3;
    int var_4;
    var_0 = malloc(16);
    var_1 = GetTickCount();
    GetSystemTime(lpSystemTime: var_0);
    for (i = 0; i < 8; i++) {
        var_3 = var_0[i];
        var_0[i] = ((unsigned int)(var_3 ^ var_1) % 24 & 0xff) + 'a';
    }
    var_4 = var_0 + 8;
    *var_4 = 0x6d6f632e;
    *(var_4 + 4) = 0x0;
    switch((((int) var_1) % 0x8) - 0x1) {
    case 0:
    case 5:
        *var_4 = *var_4 ^ 0x6d001700;
        break;
    case 1:
        *var_4 = *var_4 ^ 0x190a0d00;
        break;
    case 4:
    case 6:
        *var_4 = *var_4 ^ 0x6d1a1100;
        break;
    }
    return var_0;
}
```

What does this function return?
Please note that you cannot change the answer after selecting it.

○ No Idea

○ Manipulates the system time

○ Allows Virtual Machine Detection

○ Generates random domains

○ Encryptes memory regions

Please type all utilized top-level-domains (TLDs)

Which Top-Level domain will be utilized the most?

Which of these second-level domains can potentially be generated by the function?

	No	Yes
simpmpfp	○	○
xfbcbcic	○	○
facebook	○	○
squzuzfz	○	○
rlpmpmgmjdh	○	○

How many domains can potentially be generated by this function per day?

1.3 Comparison

In this part, we are going to show you how the decompiled function from before is decompiled by other popular decompilers.

Hex-Rays:

```
LPSYSTEMTIME sub_401000()
{
  LPSYSTEMTIME lpSystemTime; // [esp+4h] [ebp-Ch]
  WORD *p_wHour; // [esp+8h] [ebp-8h]
  char TickCount; // [esp+Eh] [ebp-2h]
  char i; // [esp+Fh] [ebp-1h]

  lpSystemTime = (LPSYSTEMTIME)malloc(16);
  TickCount = GetTickCount();
  GetSystemTime(lpSystemTime);
  for ( i = 0; i < 8; ++i )
    *((_BYTE *)&lpSystemTime->wYear + i) = (unsigned __int8)(TickCount ^ *((_BYTE
    ↪  *)&lpSystemTime->wYear + i)) % 24 + 97;
  p_wHour = &lpSystemTime->wHour;
  *(_DWORD *)&lpSystemTime->wHour = 1836016430;
  *(_DWORD *)&lpSystemTime->wSecond = 0;
  switch ( TickCount % 8 )
  {
    case 1:
    case 6:
      *(_DWORD *)p_wHour ^= 0x6D001700u;
      break;
    case 2:
      *(_DWORD *)p_wHour ^= 0x190A0D00u;
      break;
    case 5:
    case 7:
      *(_DWORD *)p_wHour ^= 0x6D1A1100u;
      break;
    default:
      return lpSystemTime;
  }
  return lpSystemTime;
}
```

Ghidra:

```
LPSYSTEMTIME FUN_00401000(void)
{
  LPSYSTEMTIME lpSystemTime;
  DWORD DVar1;
  uint *puVar2;
  uint uVar3;
  char local_5;

  lpSystemTime = (LPSYSTEMTIME)malloc(0x10);
  DVar1 = GetTickCount();
  GetSystemTime(lpSystemTime);
  local_5 = '\0';
  while (local_5 < '\b') {
    *(byte *)((int)&lpSystemTime->wYear + (int)local_5) =
         (byte)(*(byte *)((int)&lpSystemTime->wYear + (int)local_5) ^ (byte)DVar1) %
         ↪   0x18 + 0x61;
    local_5 = local_5 + '\x01';
  }
  puVar2 = (uint *)&lpSystemTime->wHour;
  *puVar2 = 0x6d6f632e;
  *(undefined4 *)&lpSystemTime->wSecond = 0;
  uVar3 = (int)(char)(byte)DVar1 & 0x80000007;
  if ((int)uVar3 < 0) {
    uVar3 = (uVar3 - 1 | 0xfffffff8) + 1;
  }
  switch(uVar3) {
  case 1:
  case 6:
    *puVar2 = *puVar2 ^ 0x6d001700;
    break;
  case 2:
    *puVar2 = *puVar2 ^ 0x190a0d00;
    break;
  case 5:
  case 7:
    *puVar2 = *puVar2 ^ 0x6d1a1100;
  }
  return lpSystemTime;
}
```

Binary Ninja:

```
int32_t sub_401000()

int32_t eax = malloc(0x10)
char eax_1 = GetTickCount()
GetSystemTime(lpSystemTime: eax)
char var_5 = 0
while (sx.d(var_5) s< 8)
```

```
    int32_t eax_6
    int32_t edx_4
    edx_4:eax_6 = sx.q(zx.d(*(eax + sx.d(var_5)) ^ eax_1))
    *(eax + sx.d(var_5)) = (mods.dp.d(edx_4:eax_6, 0x18)).b + 0x61
    int32_t ecx_1
    ecx_1.b = var_5
    ecx_1.b = ecx_1.b + 1
    var_5 = ecx_1.b
void* edx_8 = eax + 8
*(edx_8 + 0) = 0x6d6f632e
*(edx_8 + 4) = 0
int32_t edx_11 = sx.d(eax_1) & 0x80000007
if (edx_11 s< 0)
    edx_11 = ((edx_11 - 1) | 0xfffffff8) + 1
int32_t eax_9 = edx_11 - 1
if (eax_9 u<= 6)
    if (eax_9 == 4 || eax_9 == 6)
        *(edx_8 + 0) = *(edx_8 + 0) ^ 0x6d1a1100
    if (eax_9 == 0 || eax_9 == 5)
        *(edx_8 + 0) = *(edx_8 + 0) ^ 0x6d001700
    if (eax_9 == 1)
        *(edx_8 + 0) = *(edx_8 + 0) ^ 0x190a0d00
return eax
```

dewolf:

```
int sub_401000() {
    char i;
    char * var_0;
    char var_1;
    char var_3;
    int var_4;
    var_0 = malloc(16);
    var_1 = GetTickCount();
    GetSystemTime(lpSystemTime: var_0);
    for (i = 0; i < 8; i++) {
        var_3 = var_0[i];
        var_0[i] = ((unsigned int)(var_3 ^ var_1) % 24 & 0xff) + 'a';
    }
    var_4 = var_0 + 8;
    *var_4 = 0x6d6f632e;
    *(var_4 + 4) = 0x0;
    switch((((int) var_1) % 0x8) - 0x1) {
    case 0:
    case 5:
        *var_4 = *var_4 ^ 0x6d001700;
        break;
    case 1:
        *var_4 = *var_4 ^ 0x190a0d00;
        break;
    case 4:
    case 6:
        *var_4 = *var_4 ^ 0x6d1a1100;
        break;
    }
    return var_0;
}
```

Snowman:

```
struct s0* fun_401000() {
    struct s0* eax1;
    struct s0* v2;
    signed char al3;
    signed char v4;
    signed char v5;
    uint32_t edx6;
    int32_t edx7;
    uint32_t edx8;

    eax1 = reinterpret_cast<struct s0*>(malloc(16));
    v2 = eax1;
    al3 = reinterpret_cast<signed char>(GetTickCount());
    v4 = al3;
    GetSystemTime(v2);
    v5 = 0;
    while (static_cast<int32_t>(v5) < 8) {
        edx6 = static_cast<uint32_t>(*reinterpret_cast<unsigned
        ↪  char*>(reinterpret_cast<int32_t>(v2) + v5)) ^
        ↪  reinterpret_cast<uint32_t>(static_cast<int32_t>(v4));
        __asm__("cdq ");
        edx7 = reinterpret_cast<int32_t>(
        ↪  static_cast<uint32_t>(*reinterpret_cast<unsigned char*>(&edx6))) % 24 + 97;
        *reinterpret_cast<signed char*>(reinterpret_cast<int32_t>(v2) + v5) =
        ↪  *reinterpret_cast<signed char*>(&edx7);
        v5 = reinterpret_cast<signed char>(v5 + 1);
    }
    v2->f8 = 0x6d6f632e;
    (&v2->f8)[1] = 0;
    edx8 = reinterpret_cast<uint32_t>(static_cast<int32_t>(v4)) & 0x80000007;
    if (__intrinsic()) {
        edx8 = (edx8 - 1 | 0xfffffff8) + 1;
    }
    switch (edx8 - 1) {
    case 0:
    case 5:
        v2->f8 = v2->f8 ^ 0x6d001700;
        break;
    case 1:
        v2->f8 = v2->f8 ^ 0x190a0d00;
        break;
    case 4:
    case 6:
        v2->f8 = v2->f8 ^ 0x6d1a1100;
    case 2:
    case 3:
        goto 0x40112d;
    }
    return v2;
}
```

Retdec:

```
int32_t function_401000(int32_t a1) {
    int32_t * mem = malloc(16); // 0x401008
    int32_t result = (int32_t)mem; // 0x401008
    int32_t v1 = GetTickCount(); // 0x401014
    GetSystemTime((struct _SYSTEMTIME *)mem);
    for (int32_t i = 0; i < 8; i++) {
        char * v2 = (char *)(i + result); // 0x401046
        *v2 = (*v2 ^ (char)v1) % 24 + 97;
    }
    int32_t * v3 = (int32_t *)(result + 8); // 0x40107e
    *v3 = 0x6d6f632e;
    *(int32_t *)(result + 12) = 0;
    int32_t v4 = 0x1000000 * v1 / 0x1000000; // 0x401097
    int32_t v5 = v4 & -0x7ffffff9; // 0x40109b
    int32_t v6 = v5; // 0x4010a1
    if (v5 < 0) {
        // 0x4010a3
        v6 = (v4 + 7 | -8) + 1;
    }
    // 0x4010a8
    g18 = v6 - 1;
    switch (v6) {
        case 1: {
        }
        case 6: {
            // 0x4010e8
            *v3 = *v3 ^ 0x6d001700;
            // break -> 0x40112d
            break;
        }
        case 2: {
            // 0x40110c
            *v3 = *v3 ^ 0x190a0d00;
            // break -> 0x40112d
            break;
        }
        case 5: {
        }
        case 7: {
            // 0x4010c4
            *v3 = *v3 ^ 0x6d1a1100;
            // break -> 0x40112d
            break;
        }
    }
    // 0x40112d
    return result;
}
```

JEB:

```
LPSYSTEMTIME sub_401000() {
    LPSYSTEMTIME lpSystemTime = (LPSYSTEMTIME)malloc(16);
    DWORD v0 = GetTickCount();
    char v1 = (unsigned char)v0;

    GetSystemTime(lpSystemTime);
    for(char i = 0; i < 8; ++i) {
        *(char*)((int)i + (int)lpSystemTime) = (unsigned char)((unsigned
        ↪  int)*(char*)((int)i + (int)lpSystemTime) ^ (int)v1) % 24 + 97;
    }

    int* ptr0 = (int*)(lpSystemTime + 4);
    *ptr0 = 1836016430;
    *(ptr0 + 1) = 0;
    int v2 = (int)v1 & 0x80000007;
    if(v2 < 0) {
        v2 = ((v2 - 1) | 0xfffffff8) + 1;
    }

    unsigned int v3 = (unsigned int)(v2 - 1);
    switch((unsigned int)(v2 - 1)) {
        case 1: {
            *ptr0 = *ptr0 ^ 0x190a0d00;
            return lpSystemTime;
        }
        case 0:
        case 5: {
            *ptr0 = *ptr0 ^ 0x6d001700;
            return lpSystemTime;
        }
        case 4:
        case 6: {
            *ptr0 = *ptr0 ^ 0x6d1a1100;
            return lpSystemTime;
        }
        default: {
            return lpSystemTime;
        }
    }
}
```

Which aspects in the decompiled codes above are especially favorable to you?

Which aspects of the presented decompiled functions do you deem unhelpful or hamper your understanding of the code?

1.4 Feedback

Thank you for getting this far! If you would like to leave us any feedback about the survey, please use the lines below to help us improve ourselves.
Reminder: All replies in the complete study are going to be anonymized.

Thank you very much for your participation! List of utilized decompiler versions:

- JEB: 4.0-beta.3.202103090424
- Snowman: 0.1.3

- retdec: 4.0
- IDA: 7.6
- Ghidra: 9.2.2
- binaryninja: 2.3.2720-dev