贪吃蛇
反汇编代码分析报告

THINCT

December 9, 2023

SnakeGame::update

EBX 代替当前的函数栈底

```
004079C0    push    ebx
004079C1    mov     ebx, esp
004079C3    sub     esp, 8
004079C6    and     esp, -8
004079C9    add     esp, 4
004079CC    push    ebp
004079CD    mov     ebp, [ebx+4]
004079D0    mov     [esp+4], ebp
004079D4    mov     ebp, esp
```

1. 当 eip 在.text:004079C0 处，esp 所指向的是 ret addr.
2. 当 eip 在.text:004079C1 处，ebx 所指向的是 esp-4. 此时:
   - ebx+4 指向的是 ret addr
   - ebx+8 指向的是第一个参数

EBX 代替当前的函数栈底

```
004079C3    sub     esp, 8
004079C6    and     esp, 0FFFFFFF8h
004079C9    add     esp, 4
004079CC    push    ebp
```

esp 实现了向下最近的 8 的倍数取证。比如 12 取整就是 8，16 取整就是 16，18 取整就是 16. 因为是针对栈结构地址取整，所以越是往小的方向越安全，因为对于栈结构来讲，越小的地址是没有用过的地址。所以后面的 ebp,esp, ebp 只能作为局部变量的索引，而对于参数的索引，用 ebx 比较合适。

总结:

对于这个函数来讲，并不是按照套路 *ebp* 作为局部变量和函数参数的唯一参考.

operator += 传参

```
004079FF    mov     eax, [ebx+8]
00407A02    mov     ecx, [eax+4]
00407A05    push    ecx
00407A06    mov     edx, [eax]
00407A08    push    edx
00407A09    mov     eax, [ebp-2Ch]
00407A0C    add     eax, 28h ; '('
00407A0F    push    eax
00407A10    call    sf::operator+=(sf::Time
    &,sf::Time)
```

- 从 0x00407A09 到 0x00407A0F 是第一个参数, 已知 [ebp-2Ch] 为 this, 所以第一个参数为 this->offset28h, 并且为 sf::Time 引用类型. 所以 ***sf::Time* this->offset28h***.

operator += 传参

```
004079FF    mov     eax, [ebx+8]
00407A02    mov     ecx, [eax+4]
00407A05    push    ecx
00407A06    mov     edx, [eax]
00407A08    push    edx
00407A09    mov     eax, [ebp-2Ch]
00407A0C    add     eax, 28h ; '('
00407A0F    push    eax
00407A10    call    sf::operator+=(sf::Time
    &,sf::Time)
```

- 0x004079FF 已推导出为当前函数的第一个参数, 而 0x00407A02 到 0x00407A08 是连续的内存, 从 call 得知这个连续的内存是 sf::Time 类型, 所以推导出 [ebx+8] 是 sf::Time* 类型, 即 *sf::Time* [ebx+8]*

operator += 传参

总结:

operator += 第一个参数是传地址, 第二个参数是传值, 只不过 sf::Time 的内存是 8 个字节, 所以从起始地址连续压栈 2 次. 本重载函数主要需要掌握的是: **不能根据 *push* 来判断函数的参数个数**.