SSE指令由浅入深

原创

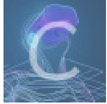
R4dish 于 2019-06-19 20:44:47 发布 460 收藏 2

分类专栏： 逆向分析

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/qq_40934487/article/details/91618214

版权

 逆向分析 专栏收录该内容

2 篇文章 0 订阅

订阅专栏

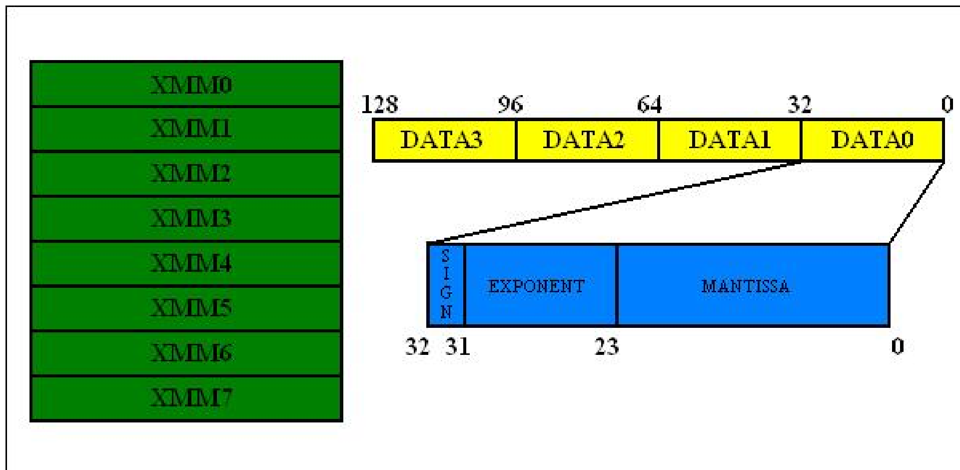
最近在CTF比赛中遇到过几次SSE指令，之前没怎么了解这个，然后抽时间总结了一些这方面的知识

介绍

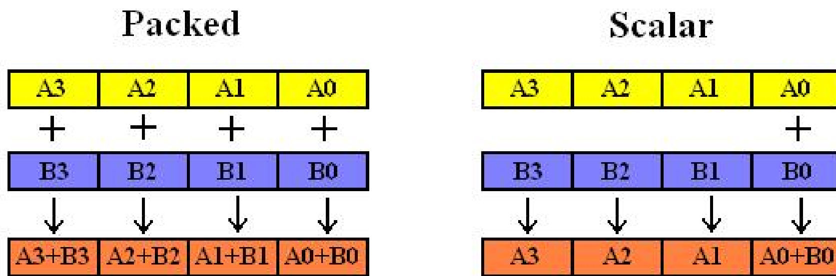
SSE是一组Intel CPU指令，是继MMX的扩充指令集，也是流化SIMD扩展，提供有70条新指令和一组暂存器，用于处理128位打包数据。在Linux下可以使用`cat /proc/cpuinfo`来查看CUP支持的指令集。

```
一筐萝卜 → Desktop  cat /proc/cpuinfo | grep flags
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_pe
rfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movb
e popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_pe
rfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movb
e popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_pe
rfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movb
e popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_pe
rfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movb
e popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_pe
rfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movb
e popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_pe
rfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movb
e popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_pe
rfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movb
e popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_pe
rfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movb
e popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
一筐萝卜 → Desktop
```

SSE新增的八个新的128位暂存器，这些128位元的暂存器可以用存放32位源的单精度浮点数，SSE的浮点数运算指令就是使用这些暂存器，下面是SSE新增的暂存器的示意图



SSE的浮点数运算指令可以分成两类：Packed和Scalar；Packed指令是一次对XMM暂存器中4个浮点数（DATA0！DATA3）都进行计算，而Scalar则只对XMM暂存器中的DATA0进行运算。如下图所示：



常用SSE指令和解释

浮点指令

- 赋值

```
movaps   把4个对齐的单精度值传送到xmm寄存器或者内存
movups   把4个不对齐的单精度值传送到xmm寄存器或者内存
movlps   把2个单精度值传送到内存或者寄存器的低四字![](imgs/20190612-205642.png)
movhps   把2个单精度值传送到内存或者寄存器的高四字
movlhps  把2个单精度值从低四字传送到高四字
movhlps  把2个单精度值从高四字传送到低四字
```

- 数学运算

```
addps    将两个打包值相加
subps    将两个打包值相减
mulps    将两个打包值相乘
divps    将两个打包值相除
rcpps    计算打包值的倒数
sqrtps   计算打包值的平方根
rsqrtps  计算打包值的平方根倒数
maxps    计算两个打包值中的最大值
minps    计算两个打包值中的最小值
```

- 比较

```
cmpps   比较打包值
cmpss   比较标量值
comiss  比较标量值并且设置eflags寄存器
ucomiss 比较标量值（包括非法值）并设置eflags寄存器
```

- unpack和shuffle

```
SHUFPS  交错的结果对存储在xmm1中
UNPCKHPS 从xmm1和xmm2 / m128的高四字中解包和交错单精度浮点值
UNPCKLPS 从xmm1和xmm2 / m128的低四字中解包和交错单精度浮点值。
```

- 逐位逻辑运算

```
andps   计算两个打包值的按位逻辑与
andnps  计算两个打包值的按位逻辑非
orps    计算两个打包值的按位逻辑或
xorps   计算两个打包值的按位逻辑异或
```

整数指令

```
pavgb 计算打包无符号字节整数的平均值
pavgw 计算打包无符号字整数的平均值
pextrw 把一个字从mmx寄存器复制到通用寄存器
pinsrw 把一个字从通用寄存器复制到mmx寄存器
pmaxub 计算打包无符号字节整数的最大值
pmaxsw 计算打包有符号字整数的最大值
pminub 计算打包无符号字节整数的最小值
pminsw 计算打包有符号字整数的最小值
pmulhuw 将打包无符号字整数相乘并且存储高位结果
psadbw 计算无符号字节整数的绝对差的总和
```

SSE浮点数运算操作

```
addps/addss _mm_add_ps/_mm_add_ss 加法
subps/subss _mm_sub_ps/_mm_sub_ss 减法
mulps/mulss _mm_mul_ps/_mm_mul_ss 乘法
divps/divss _mm_div_ps/_mm_div_ss 除法
sqrtps/sqrtss _mm_sqrt_ps/_mm_sqrt_ss 平方根
maxps/maxss _mm_max_ps/_mm_max_ss 逐项取最大值
minps/minss _mm_min_ps/_mm_min_ss 逐项取最小值
```

MMX指令和解释

MMX指令分为一下几种：

- 数据传送：movd, movq
- 数据转换：packsswb, packssdw, packuswb, punpckhbw, punpckhwd, punpckhdq, punpcklbw, punpcklwd, punpckldq
- 并行算术：paddb, paddw, paddd, paddsb, paddsw, paddusb, paddusw, psubb, psubw, psubd, psubsb, psubsw, psubusb, psubusb, psubusw, pmulhw, pmullw, pmaddwd
- 并行比较：pcmpeqb, pcmpeqw, pcmpeqd, pcmpgtb, pcmpgtw, pcmpgtd
- 并行逻辑：pand, pandn, por, pxor
- 移位与旋转：psllw, pslld, psllq, psrlw, psrld, psrlq, psraw, psrad
- 状态管理：emms

下面只列出一些常用的解释，如果有需要的话可以到传送门 去查询

```
movd  变量a的32位拷贝到MMX寄存器的低32位，高32位置零
movd  MMX变量的低32位拷贝到int类型中
punpcklbw MM,MM/m64   把目的寄存器与源寄存器的低32位按字节交错排列放入目的寄存器
punpcklwd MM,MM/m64   把目的寄存器与源寄存器的低32位按字交错排列放入目的寄存器
pshufd XMM,XMM/m128,imm8(0~255)   将源存储器的4个双字由imm8指定选入目的寄存器,内存变量必须对齐内存16字节
movaps 把4个对准的单精度值传送到xmm寄存器或者内存
pmulld 对128位寄存器的每32位做整形乘法运算，形成一个64位的立即数，然后取立即数的低32位到目的寄存器的对应bit位中
movups  将压缩单精度浮点值从 xmm2/m128 移到 xmm1
paddd   对128位寄存器的每32位做整形加法运算
pxor    对 xmm2 同 xmm1 执行逐位“异或”运算
```

例子0x01

以2019强网杯Just re作为例子，具体获取flag的方式不再说了，只谈一下涉及到SSE、MMX指令的

在check1中存在一段这样的指令：

```
.text:00401728                 movsx   ecx, dh
.text:0040172B                 movd    xmm0, ecx
.text:0040172F                 punpcklbw xmm0, xmm0
.text:00401733                 punpcklwd xmm0, xmm0
.text:00401737                 pshufd  xmm0, xmm0, 0
.text:0040173C                 movaps  xmmword ptr [ebp-20h], xmm0
.text:00401740                 test    esi, esi
.text:00401742                 jz      loc_401891
.text:00401748                 xor     esi, esi
.text:0040174A                 cmp     dword_4053C4, 2
.text:00401751                 jl      loc_40180B
.text:00401757                 movd    xmm0, dword ptr [ebp-20h]
.text:0040175C                 mov     esi, 10h
.text:00401761                 movaps  xmm3, ds:xmmword_404340
.text:00401768                 pmovzxbd xmm4, xmm0
.text:0040176D                 pmulld  xmm4, ds:xmmword_404380
.text:00401776                 movups  xmm0, xmmword_405018
.text:0040177D                 movaps  xmm1, xmm4
.text:00401780                 movups  xmm2, xmm3
.text:00401783                 paddd   xmm1, xmm0
.text:00401787                 paddd   xmm2, xmm5
.text:0040178B                 movups  xmm0, xmmword_405028
.text:00401792                 pxor    xmm2, xmm1
.text:00401796                 movaps  xmm1, xmm4
.text:00401799                 movups  xmmword_405018, xmm2
.text:004017A0                 movaps  xmm2, ds:xmmword_404350
.text:004017A7                 paddd   xmm1, xmm0
.text:004017AB                 movups  xmm0, xmmword_405038
.text:004017B2                 paddd   xmm2, xmm3
.text:004017B6                 paddd   xmm2, xmm5
.text:004017BA                 pxor    xmm2, xmm1
.text:004017BE                 movaps  xmm1, xmm4
.text:004017C1                 movups  xmmword_405028, xmm2
.text:004017C8                 movaps  xmm2, ds:xmmword_404360
.text:004017CF                 paddd   xmm1, xmm0
.text:004017D3                 movups  xmm0, xmmword_405048
.text:004017DA                 paddd   xmm2, xmm3
.text:004017DE                 paddd   xmm2, xmm5
.text:004017E2                 paddd   xmm4, xmm0
.text:004017E6                 pxor    xmm2, xmm1
.text:004017EA                 movaps  xmm1, ds:xmmword_404370
.text:004017F1                 paddd   xmm1, xmm3
.text:004017F5                 paddd   xmm1, xmm5
.text:004017F9                 pxor    xmm1, xmm4
.text:004017FD                 movups  xmmword_405038, xmm2
.text:00401804                 movups  xmmword_405048, xmm1
```

对于这类题，用X32dbg动态调试最容易理解，经过动态调试分析，不难发现这段汇编代码的意思和下面的一样

```
do
{
  *(&xmmword_405018 + v20) = (v20 + v3) ^ (0x1010101 * v11 + *(&xmmword_405018 + v20));
  ++v20;
}
while ( v20 < 24 );
```

例子0x02

SSE不仅运用在CTF逆向题目中，在PWN题目中也有时出现过

首先程序只开启了PIE保护，载入IDA中查看程序流程

```
signed __int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  char **v4; // [rsp+0h] [rbp-40h]
  unsigned int v5; // [rsp+14h] [rbp-2Ch]
  _BYTE *v6; // [rsp+18h] [rbp-28h]
  void *s; // [rsp+20h] [rbp-20h]
  int v8; // [rsp+2Ch] [rbp-14h]
  int v9; // [rsp+2Ch] [rbp-14h]
  unsigned int v10; // [rsp+3Ch] [rbp-4h]

  v4 = a2;
  v8 = sub_ADB("./flag.txt");                   // 返回文件内字符串的长度
  if ( v8 == -1 )
    return 0xFFFFFFFFLL;
  v9 = v8 + 33;
  s = malloc(v9);
  if ( s == -1LL )
    return 0xFFFFFFFFLL;
  memset(s, 0, v9);
  sub_B1C(s);
  sub_BBD("./flag.txt", s + 32, s + 32);
  v10 = 7 * sub_930(s);
  do
  {
    printf("%d> ", v10, v4);
    fflush(stdout);
    v6 = sub_CA0();
    if ( !v6 )
      return 0xFFFFFFFFLL;
    v5 = sub_94F(s, v6);
    if ( !v5 )
    {
      puts("Well done.");
      return 0LL;
    }
    printf("Ah ah ah, you didn't say the magic word! [%d]\n", v5);
    --v10;
  }
  while ( v10 );
  return 0LL;
}
```

首先程序使用`__xstat`函数对flag.txt的大小进行了查询，然后分配了`文件大小+33`的空白内存

`sub_B1C`函数是通过`/dev/urandom`来生成32位字符串

```
1  int __fastcall sub_B1C(__int64 a1)
2  {
3    int v1; // eax
4    int buf; // [rsp+10h] [rbp-10h]
5    unsigned int v4; // [rsp+14h] [rbp-Ch]
6    int fd; // [rsp+18h] [rbp-8h]
7    int v6; // [rsp+1Ch] [rbp-4h]
8
9    fd = open("/dev/urandom", 0);
10   v6 = 0;
11   v4 = sub_930("qwertyuiop[]\\asdfghjkl;'zxcvbnm,./`1234567890--=~!@#$%^&*()_+QWERTYUIOP{}|ASDFGHJKL:\"ZXCVBNM<>?");
12   while ( v6 <= 31 )
13   {
14     do
15     {
16       read(fd, &buf, 4uLL);
17       buf = buf;
18     }
19     while ( buf >= v4 );
20     v1 = v6++;
21     *(a1 + v1) = aQwertyuiopAsdf[buf];
22   }
23   return close(fd);
24 }
```

这个函数中存在SSE指令的是`sub_930`函数

```
.text:0000000000000930 sub_930         proc near               ; CODE XREF: main+A0↓p
.text:0000000000000930                                         ; sub_B1C+33↓p
.text:0000000000000930                 mov     rax, 0FFFFFFFFFFFFFFF0h
.text:0000000000000937                 mov     rdx, rdi
.text:000000000000093A                 pxor    xmm0, xmm0
.text:000000000000093E
.text:000000000000093E loc_93E:                                ; CODE XREF: sub_930+19↓j
.text:000000000000093E                 add     rax, 10h
.text:0000000000000942                 pcmpistri xmm0, xmmword ptr [rdx+rax], 8
.text:0000000000000949                 jnz     short loc_93E
.text:000000000000094B                 add     rax, rcx
.text:000000000000094E                 retn
.text:000000000000094E sub_930         endp
.text:000000000000094E
```

代码也很简单，用到了`pcmpistri`指令，意思是执行字符串数据与隐式长度的打包比较，生成索引，并将结果存储在`ECX`中；这个函数的意思就是确保从`/dev/urandom`读取的字符串长度小于`0x5f`

继续往下看，在`sub_94F`函数中也存在SSE指令

```
1  signed __int64 __fastcall sub_94F(__int64 a1, __int64 a2)
2  {
3    __int64 v2; // rax
4    const __m128i *v3; // rsi
5    __m128i v4; // xmm0
6    unsigned __int8 v5; // cf
7    __int64 v6; // rcx
8    signed __int64 result; // rax
9
10   v2 = a1 - a2;
11   v3 = (a2 - 16);
12   do
13   {
14     ++v3;
15     v4 = _mm_loadu_si128(v3);
16     v6 = _mm_cmpistri(v4, *(v3 + v2), 24);
17     v5 = _mm_cmpistrc(v4, *(v3 + v2), 24);
18   }
19   while ( !(v5 | _mm_cmpistrz(v4, *(v3 + v2), 24)) );
20   if ( v5 )
21     result = (v6 + 1) * (((*(v3->m128i_i64 + v2 + v6) - *(v3->m128i_i64 + v6)) >> 63) | 1);
22   else
23     result = 0LL;
24   return result;
25 }
```

经过动态调试分析出来该函数的意思：如果我们输入的字符串的ascii码小于内存中的字符串的话，函数返回1，如果大于则返回`-1`，如果相等的话返回`(index+1)%16`

那么我们可以不断的猜测得到内存中的字符串，也就是flag

Exp：

```
from pwn import *
context.log_level = 'debug'
context.terminal = ['deepin-terminal', '-x', 'sh' ,'-c']
index_1 = "qwertyuiop[]\\asdfghjkl;'zxcvbnm,./`1234567890--=~!@#$%^&*()_+QWERTYUIOP{}|ASDFGHJKL:\"ZXCVBNM<>?"
index_1 = ''.join(sorted(index_1))
flag = ""
r = process("./pwn")
k = 2
def guess(string, l, x):
    r.recvuntil("> ")
    mid = (l+x)/2
    #gdb.attach(r)
    r.sendline(flag + string[mid])
    res = r.recvline()[:-1]
    res = int(res.split("[")[1].split("]")[0])
    if res == k:
        return string[mid]
    elif res < 0:
        return guess(string, l, mid - 1)
    return guess(string, mid + 1, x)
for i in xrange(32):
    a = guess(index_1, 0, len(index_1))
    flag += a
    k+=1
    if k > 16:
        k = 1
index_2 = " !\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~"
for i in xrange(20):
    a = guess(index_2, 0, len(index_2))
    flag += a
    k+=1
    #print flag
    if k > 16:
        k = 1
log.info("flag  ---> " + flag)
```

总结

通过做一些存在SSE指令的题目可以发现，动态调试更容易理解程序到底是干什么的。

本文如有不妥之处，敬请斧正。

参考文献：

x86 and amd64 instruction reference
Ping-Che Chen