一天学会 alpha 语言

更新记录

时间	版本	其他
2015 年 7 月 24 日	1.1	
2015 年 7 月 30 日	1.2	
2015 年 8 月 20 日	1.3	
2015 年 9 月 23 日	1.4	
2015 年 12 月	1.5	
2015 年 6 月 8 日	1.6	

目录

语言简介

Alpha 语言是 AlphaFuzzer Framework 的专用语言。学会 Alpha 以后可以快速在 AlphaFuzzer Framework 上构建自己的 fuzz 工具，来满足对目标的测试需求。

使用范围

该语言使用于如下人员
软件开发工程师
软件测试工程师
软件安全研究人员
计算机安全爱好者
只要目标为文件格式类型（如图片，视频，音频，图标，文档，自定义格式等均可）皆可以

用本语言来构造属于自己的 fuzzer，对目标进行安全测试。测试成功与否取决于两个方面，一是引擎是否够强大，二是使用者编写的脚本信息是否精确。

支持语句

_num

使用示例:_num,888,0,0,32;
用于表示一个数值.共有四个参数.
参数 1：表示这个数值的数.比如例子里面表示 888.
参数 2：表示该数值是否可变,0 为不可变,1 为可变.若可变则不对参数 1 进行解析.
参数 3：表示大小尾.0 表示小尾.1 表示大尾.（小尾指低位数据存放在低位地址上.大尾则表示低位数据存放在高位地址上).
参数 4：数值的大小.支持 8 位.16 位.32 位 3 种类型.也就是 1 字节.2 字节.4 字节.

_str

使用示例:_str,hello,1,0;
用于表示一个字符串.共有 3 个参数.
参数 1：字符串数值.直接填写字符串即可.
参数 2：字符串是否可变.0 为不变,1 为数据变长度_不变,2 为数据长度均可变.
参数 3：字符串的长度.如果是 0.则是输入字符串的实际长度
.该语句最多输出 127 个字符

_cal,addr

使用示例:_cal,addr,32,0,3;
用于计算一些数据的偏移.共有 3 个参数.
参数 1：表示计算结果的位数.可选值为 8.16.32.分别表示 1.2.4 字节.
参数 2：表示计算结果的大小尾.0 为小尾.1 为大尾.
参数 3：要计算的目标的函数序列号.上面的参数 3 是 3.代表计算第三个函数的偏移.

_cal,size

使用示例:_cal,size,32,0,4,6;
用于计算一些数据的大小.共有 4 个参数.
参数 1：表示计算结果的位数.可选值为 8.16.32.
参数 2：表示计算结果的大小尾.0 为小尾.1 为大尾.

参数 3：要计算的开始函数(函数序列号)
参数 4：要计算的结束函数(函数序列号)
如上示例.开始是 4.结束为 6.则表示计算函数序列号为 4 5 6 三条指令生成数据的总大小.

_cal,f+类(5 种)

使用示例:_cal,f+,32,0,3,5;
用于对_num 类型输出的数据作计算.共有 4 个参数.
参数 1：计算结果的大小.分别为 6.16.32.分别表示计算结果是 1.2.4 字节.
参数 2：计算结果的大小尾.0 表示小尾.1 表示大尾.
参数 3：用于计算的第一个函数的参数 ID.
参数 4：用于计算的第二个函数的参数 ID.
该函数所表示的值=第三个函数的数值+第四个函数的数值.
该类函数共用 5 种.分别为
_cal,f+ 结果=参数 3 指向函数数值+参数 4 指向函数数值
_cal,f- 结果=参数 3 指向函数数值-参数 4 指向函数数值
_cal,f* 结果=参数 3 指向函数数值*参数 4 指向函数数值
_cal,f/ 结果=参数 3 指向函数数值/参数 4 指向函数数值
_cal,f% 结果=参数 3 指向函数数值%参数 4 指向函数数值

_cal,n+类(5 种)

使用示例:_cal,n+,32,0,3,200.
用于计算一些数据的大小.共有 4 个参数.
参数 1：计算结果的大小.分别为 6.16.32.分别表示计算结果是 1.2.4 字节.
参数 2：计算结果的大小尾.0 表示小尾.1 表示大尾.
参数 3：用于计算的第一个函数的参数 ID.
参数 4：用于计算的第二个参数.该数是数值.
该函数所表示的值=第三个函数的数值+第四个参数值 200.
该类函数共用 5 种.分别为
_cal,n+ 结果=参数 3 指向函数数值+参数 4
_cal,n- 结果=参数 3 指向函数数值-参数 4
_cal,n* 结果=参数 3 指向函数数值*参数 4
_cal,n/ 结果=参数 3 指向函数数值/参数 4
_cal,n% 结果=参数 3 指向函数数值%参数 4

_cal,-n 类(3 种)

使用示例:_cal,0-n,32,0,200,3.

用于计算一些数据的大小.共有 4 个参数.
参数 1：计算结果的大小.分别为 6.16.32.分别表示计算结果是 1.2.4 字节.
参数 2：计算结果的大小尾.0 表示小尾.1 表示大尾.
参数 3：用于计算的第一个参数.该数是数值.
参数 4：用于计算的第二个函数的参数 ID.
该函数所表示的值=第三个参数 200-第四个函数的值.
该类函数共用 3 种.分别为
_cal,-n 结果=参数 3-参数 4 指向函数数值
_cal,/n 结果=参数 3/参数 4 指向函数数值
_cal,&n 结果=参数 3%参数 4 指向函数数值

_copy

使用示例:_copy,1,3,100,0;
用于对一段指令生成的数据进行复制.该函数有 4 个参数 如
参数 1 要复制的开始 ID 值.
参数 2.要复制的结束 ID 值.
参数 3.要复制的份数.
参数 4.优先级,可设为 0-15.
如上面指令 表示要把 1-3 条指令的数据.复制 100 次.
该函数要求：
如果参数 3 为 0.则表示 0-100 的随机数.

[<这里是注释信息>]

用于表示注释代码的信息.用[<>]来表示.填写在每一条函数的末尾（不得填写在最后一条函数的末尾.
注释信息不能超过 16 字节

_crc32

使用示例:_crc32,1,2,5;
用于计算一段数据的 CRC32 值.数据可以为 num 类型 也可以为长度不会变化的 STR 类型.
该函数一共有 3 个参数
参数 1：大小尾 0 表示大尾.1 表示小尾.
参数 2：数据开始的 ID
参数 3：数据结束的 ID
如上面指令表示第二条数据到第五条数据的 CRC32 值.占用空间 4 字节.

_crc16

使用示例:_crc16,1,2,5;
用于计算一段数据的 CRC16 值.用法和_CRC32 一样.只是占用空间为 2 字节.

_hex

使用示例:_hex,004010cc;
用于定义一段 HEX 数值.最多 64 字节.一个参数. 如_hex,004010cc;

_choice

使用示例:_choice,3,5;
选择语句.用来对一段指令序列随机选择.该函数一共有 2 个参数.
表示在 3-5 条语句中随机选择一条语句作为输出.
可用于指向语句的类型有：
_num
_str
_hex

_DelToEnd

使用示例:_DelToEnd,0;
删除语句.可以根据情况来选择是否删除该指令后续的全部代码.
该函数有 1 个参数.
如果该参数为 0.表示无条件删除后面的指令.
如果该参数非 0.则表示参数语句的 ID.如果指向 ID 参数语句值为 0.则删除_DelToEnd 语句后面的全部语句.否则不删除.
可用于指向语句的类型有：
_CountOfCopy
_num
_cal,addr
_cal,size
_cal,f+
_cal,n+
_cal,n-
注意：
1 不要用于指向无值意义的指令.如_str 类型.
2 被删除的语句参与计算.但是不参与输出.所以请小心使用该语句.

3 该语句可用于删除最后几条用于计算的零时语句.

_when

使用示例:_when,32,0,3,4,5;
有条件选择语句.该语句一共有 5 个参数.
参数 1：计算结果的大小.分别为 6.16.32.分别表示计算结果是 1.2.4 字节.
参数 2：大小尾.0 大尾 1 为小尾.
参数 3：判断语句的 ID.如果该 ID 值为 0.则输出参数 4.否则输出参数 5.
参数 4：语句 ID
参数 5：语句 ID
例子表示判断第三条指令的值是否为 0.如果为 0 输出参数 4 的值.否则输出参数 5 个值.输出结果为 4 字节.大尾格式.
参数 3-参数 5 指向的 ID 应为_NUM 或相似类型

_nop

使用示例:_nop:
_nop 语句.该语句没有参数.如：_nop;
该语句占用行数.但是不生成任何数据.
该指令不受到{}的约束。

_nopto

使用示例:_nopto,222;
_nopto 语句.该语句有一个参数.如
参数 1：表明从现在到 222 行都设置为 nop 语句.
该指令不受到{}的约束。

_RangeNum

使用示例:_RangeNum,111,222,1,32;
随机数语句.该语句有四个参数
参数 1：随机数最低值
参数 2：随机数最高值
参数 3：随机数大小尾 0 表示小尾 1 表示大尾
参数 4：随机数所占字节 8 16 或者 32
该语句表示生成一个 32 位.大尾.的随机数.随机数在[111,222)之间

_align

使用示例:_align,16,200;
对齐指令.该函数有 2 个参数.如：
参数 1：按照多少字节来对齐（1-256）.
参数 2：从多少条指令开始对齐.

_bit

使用示例:_bit,1,1,1,1,0,0,0,2;
二进制数据.该函数有 8 个参数.表示八位.
其中 0 表示该位 0.1 表示该位为 1.2 表示该位是可变的.如：

_array

使用示例:_array,2,100,100,1,1:
数组函数.用来生成一些随机数组数据十分方便.比如在构建 BMP 脚本的时候.
该函数可生成 4 维数组.该函数一共有 5 个参数.
参数 1：数组是否可变.值为 0-4,0 表示不可变.1 表示参数 2 可变. 如此类推.4 表示参数 2-参数 5 都可变.
参数 2-参数 5：表示四维数组.
该条语句表示最大为 100*100 的二维可变数组

_CountOfArray

使用示例:_CountOfArray,32,0,20.1;
取数组随机只的函数.该函数有 4 个参数.如：
参数 1：结果的大小.8.16.32 字节.
参数 2：结果的大小尾.
参数 3：指向函数 ID 值.只能为 Array 函数.
参数 4：指向哪一个维度的数组.可选为 1-4.

{}

相对偏移为“{}”该结构不占用代码行数.括号内的指令行数可以认为是正括号后的相对偏移.最多支持 16 对.

Block

使用示例：

```
block,A002;
{
    _str,BBBB,,;
}
```

结构函数.用来表示数据块.该函数有 1 个参数.为数据名字.函数后面用大括号表示.可以嵌套.

copy

使用示例:copy,A001,1,0,0;
对结构进行复制的函数.该函数有 4 个参数.
参数 1 结构名字.
参数 2 复制的次数.
参数 3 随机 0 或者 1.如果为 0 表示复制次数为[0,参数 2)的值.
参数 4 优先级.0-15.0 最先被处理.15 最后被处理.

CountOfCopy

使用示例:CountOfCopy,32,0,4,3;
取复制次数的函数.一共有 4 个参数 如
参数 1 结果的大小.位. 可为 8.16.32；
参数 2 结果的大小尾
参数 3 指向的 ID 值（必须为 Count）
参数 4 对结果进行加法运算的加数.

_DelToBracket

使用示例：_DelToBracket;
该函数无参数，表示删除该结构后面的指令。直到结构结束。
该函数必须写在 block 结构内。用于调试语句用。

size

使用示例：size,32,0,xxx.20;
该函数有 3 个参数，用来计算结构 block 的大小
参数 1，计算结果的大小，单位为位。可设置数值为 8，16，32；
参数 2，计算结果的大小尾。
参数 3，指向结构名称。

参数 4，对结果进行加法运算。

addr

使用示例：addr,32,0,xxx;
该函数有 3 个参数，用来计算结构 block 的偏移
参数 1，计算结果的大小，单位为位。可设置数值为 8，16，32；
参数 2，计算结果的大小尾。
参数 3，指向结构名称。
参数 4，对结果进行加法运算。

Choice

使用示例：choice,xxx;
该函数有 1 个参数，指向 block 结构。

crc32

使用示例：crc32,0,xxx,0;
该函数有 3 个参数。
参数 1，计算结果的大小尾。
参数 2，指向 block 结构。
参数 3，优先级。0-15。

Crc16

使用示例：crc16,0,xxx,0;
该函数有 3 个参数。
参数 1，计算结果的大小尾。
参数 2，指向 block 结构。
参数 3，优先级。0-15。

使用例子

第 1 课:HelloWorld

难度：简单
代码:_str,HelloWorld,0,0;
解析:要输出的代码可看成一个字符串，首先可以想到的就是字符串函数。字符串函数为_str,一共有 3 个参数.参数 2 为可变，我们这里输入不变的字符串，可以设置为 0，参数 3 为长度，0 为忽略。因此我们也可以设置为 0.由于 0 可以省去，因此这个字符串也可以表示成下面的样子:
_str,HelloWorld,,;
或者是下面的样子:
_str,HelloWorld,0,10;
也可以是下面的样子:
_str,Hello,0,0;
_str,World,0,0;
输出数据为:

Offset	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
00000000	48 65 6C 6C 6F 57 6F 72 6C 64	HelloWorld

第 2 课:字符串函数_str

难度：中等
使用频率：高
该函数用来输出字符串数据。
例 1:
_str,HelloWorld,0,20;
输出结果:

Offset	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
00000000	48 65 6C 6C 6F 57 6F 72 6C 64 00 00 00 00 00 00	HelloWorld
00000016	00 00 00 00	

分析:虽然已经强制生成长度为 20 字节的数据，而 HelloWorld 只有 10 字节，因此其他的长度用 0 补齐。
例 2:
_str,HelloWorld,0,5;
输出结果:

Offset	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
00000000	48 65 6C 6C 6F	Hello

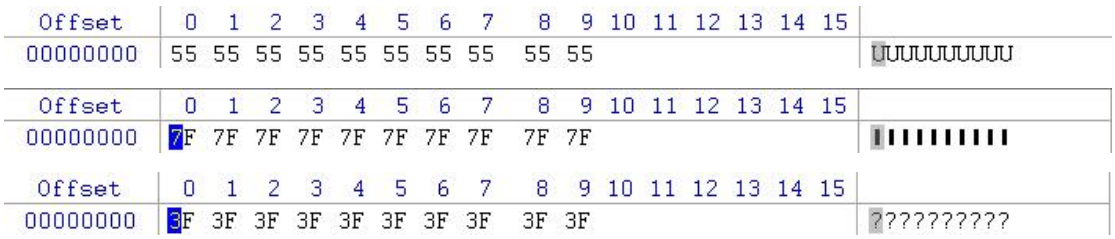
分析:虽然代码要求输入 10 字节的 HelloWorld,但是第四个参数已经指定只可以输入 5 字节，因此最终结果只输出了字符串的前 5 字节.
该例子说明，参数 3 的优先级大于参数 1 的优先级

例 3:

_str,HelloWorld,1,0;

和前面例子不同的是，此次输出了大量的文件，因为参数 2 为 1 表示可变的。

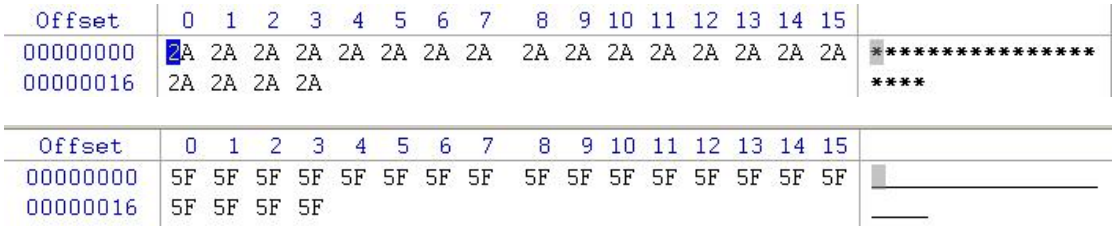
输出结果;



例 4:

_str,,1,20;

输出结果:

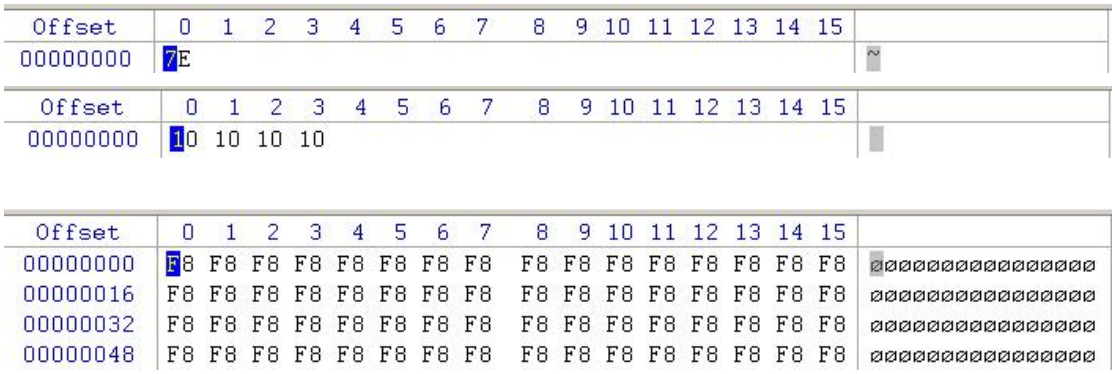


表示输出一条字符串，内容可变，但是长度为 20。

例 5:

_str,0,2,0;

输出结果



代码解析：由于可变参数值为 2,表示该数据无论长度还是内容都是可以变化的。因此生成上面数据。

提示：

1：可变值有 3 个选项，分别是 0，1，2。

2：非变异情况下该语句最多只能输出 127 个字符，如果字符更多，请用多条语句。

第 3 课:数字函数_num

难度：中等

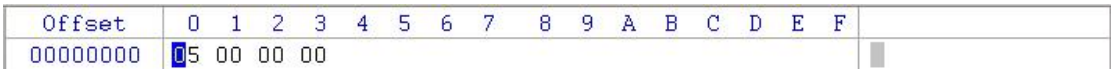
使用频率：高

该函数用来输出数值数据。

例 1:

_num,5,0,0,32;

输出结果：



例 2:

_num,5,0,1,32;

输出结果：

解析：此两条语句不同之处在于参数 3，例 1 为 0，表示小尾，例 2 为 1，表示大尾。32 则表示 32 字节。

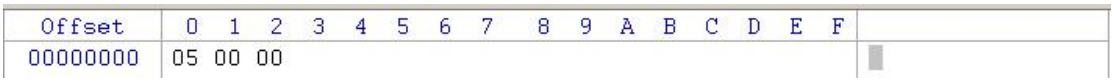
经验：一般情况下文件格式小尾的居多。

例 3：

_num,5,0,0,16;

_num,0,0,0,8;

输出数据：

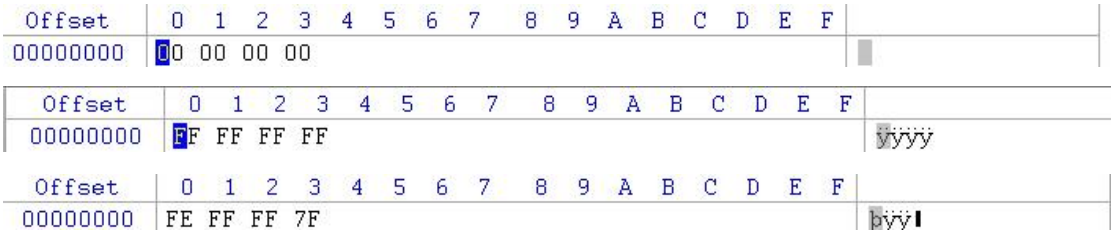


提示：24 字节的数据可以由 8 字节和 16 字节两条数据组成。

例 4：

_num,0,1,0,32;

输出数据：



解析：第二个参数为 1 表示随机。表示输出可能让程序出问题的值。但是注意在确定随机数之前先依据文件格式规则确定该结构的大小尾。

就是说以下 2 条语句的挖掘效果是截然不同的。

_num,0,1,0,32;

_num,0,1,1,32;

提示：

1.注意数据大小尾，即使数据需要变异。

第 4 课:十六进制函数_hex

难度：简单

使用频率：较高

该函数用于输出 hex 类型数据，最多可输出 63 字节。主要补充一些情况下_str 语句无法满足的情况，如逗号，换行符等等。

例 1:

_hex,33332c0d0a;
输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	33	33	2C	0D	0A												33,

提示：
1 在_str 类型无法满足需求的时候，可以考虑一下_hex 语句，或许问题就会迎刃而解。
2 该语句不支持变异，如果需要变异请使用_str 语句。

第 5 课:二进制函数_bin

难度：简单
使用频率：较低
该函数用来输出可变的二进制数据。此类数据特点是数据按照位来表示一些数值。某些位是固定的，某些位是可变的。该函数一共 8 个参数，2 表示可变，0 和 1 表示具体数值。
例 1：
_bit,1,1,1,1,0,0,0,1;
输出结果：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	F1																ñ

解析：16 进制的 F1 换成 2 进制就是 11110001，由于不可变，因此只输出这一个结果
例 2：
_bit,2,1,1,1,0,0,0,2;
输出结果:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	70																p

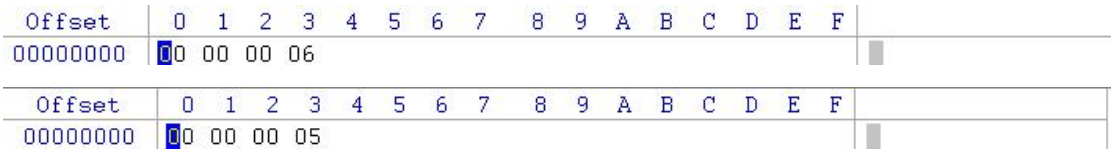
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	F1																ñ

解析，该函数有 2 个字节存在变异，因此可以出现 4 种数据。
分别为：
11110001(F1)
11110000(F0)
01110001(71)
01110000(70)
提示：如果某结构可以用本语句表达，那就使用吧。

第 6 课:范围内的随机数_RangeNum

难度：中等
使用频率：较高
作用：用来表示某指定范围的随机数。
例 1：
_RangeNum,5,8,1,32;
解析：表示输出 5-8 内的随机数，要求 32 字节，大尾形式。

输出结果：



提示，某些情况下_num 不能满足需求的时候，可使用该函数。

例 2：请考虑下面代码

```
_str,我的名字叫,,;
_RangeNum,65,90,0,8;
_RangeNum,97,122,0,8;
_RangeNum,97,122,0,8;
_str, ,,;
_str,我今年,,;
_RangeNum,48,57,0,8;
_RangeNum,48,57,0,8;
_str,岁,,;
```

输出结果：

提示信息：

机器码 48-57 范围为 0-9.

65-90 范围为 A-Z.

65-90 范围为 a-z.

提示：在_num 函数不能准确对文件格式进行建模的时候，那就考虑下_RangeNum 函数吧。

第 7 课:对齐指令函数_align

难度：中等

使用频率：低

作用：一些格式处理时候为了保证速度，采取了按照某字节对齐来计算。如某四字节对齐。

例 1：

```
_str,A,,;
_str,B,,;
_str,C,,;
_align,16,1;
```

输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	41	42	43	00	00	00	00	00	00	00	00	00	00	00	00	00	ABC

解析：第 1 行代码偏移为 0.第 4 行代码表示按照第 1 行代码偏移做 16 字节对齐。因此数据为 16 字节。

例 2：

```
_str,A,,;
_str,B,,;
_str,C,,;
_align,16,3;
```

输出数据:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	41	42	43	00	00	00	00	00	00	00	00	00	00	00	00	00	ABC
00000010	00	00															

解析：第 3 行代码偏移为 2.第四行代码表示按照第一行代码偏移做 16 字节对齐。因此数据为 2+16=18 字节。

提示：

1 有时候在对自己建模脚本进行调试的时候，使用对齐函数可以事半功倍。

2 对齐 ID 值必须在本语句所占 ID 值的前面。否则无效。

第 8 课:数据校验指令(_crc16,_crc32)

难度：简单

使用频率：低

说明：_crc32 是对数据进行 crc32 计算的指令，值为 32 位（4 字节），_crc16 是对数据进行 crc16 计算的指令，值为 16 位（2 字节）。

例 1：

```
_crc32,0,2,5;
_str,AAAA,,;
_str,BBBB,,;
_str,CCCC,,;
_str,DDDD,,;
```

输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	10	B2	78	59	41	41	41	41	42	42	42	42	43	43	43	43	²xYAAAABBBBCCCC
00000010	44	44	44	44													DDDD

解析，参数 2=2，参数 3=5，表示计算 2-5 条数据生成的 CRC32 值，值以小尾（第一个参数是 0）的形式保存。如输出数据，该 CRC32 的值为 5978B210H。

提示：

1 在 zip 等格式建模时候，可以用该指令来躲避程序对校验值的检测。

2 仅用于对数据类格式进行解析，不应对无意义的结构进行解析。

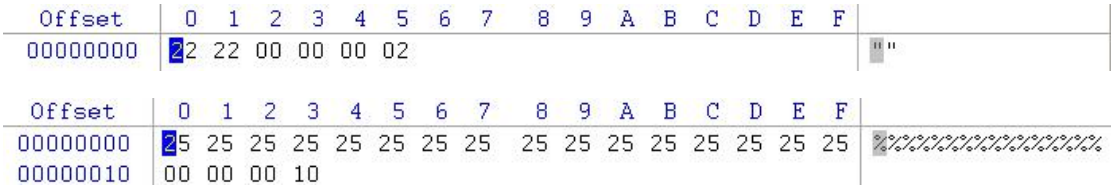
3 解析的范围不能包括自身。

第 9 课:地址计算指令_cal,addr

难度：中等
使用频率：高
说明：该函数用于对数据地址进行计算。
例 1：

```
_str,,2,;
_cal,addr,32,1,2;
```

输出数据：



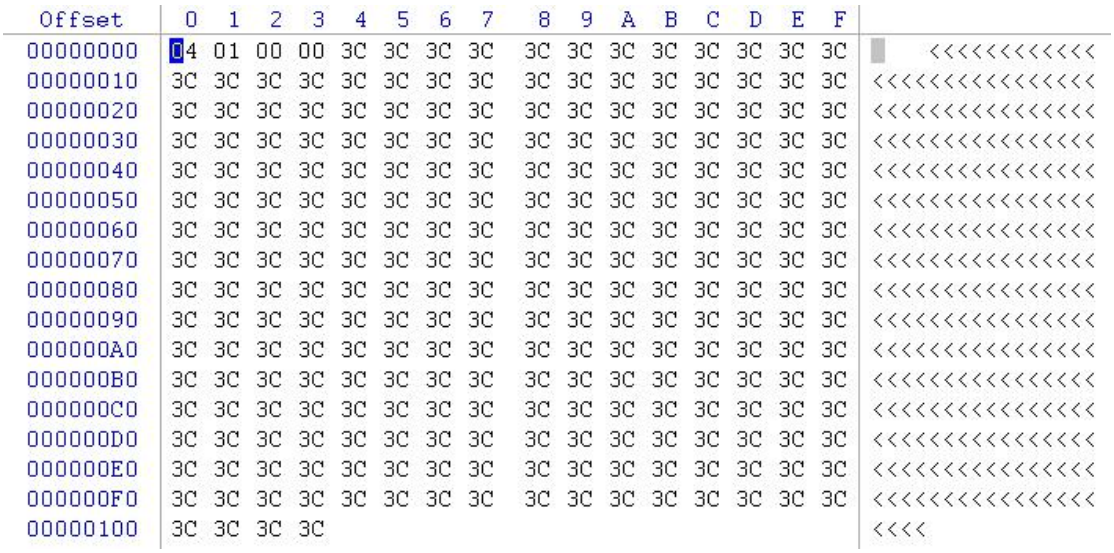
解析：由于第一条语句是可变语句，长度可变的，因此第二条语句的偏移也是可变的。该语句是输出第二条语句的偏移地址，以 32 位，大尾的形式输出。
提示：请尽量多的使用该语句。

第 10 课:数据大小计算指令_cal,size

难度：中等
使用频率：高
说明：该函数用于对数据大小进行计算。
例 1：

```
_cal,size,32,0,1,2;
_str,,2,;
```

输出数据：



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	04	01	00	00	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<
00000010	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
00000020	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
00000030	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
00000040	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
00000050	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
00000060	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
00000070	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
00000080	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
00000090	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
000000A0	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
000000B0	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
000000C0	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
000000D0	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
000000E0	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
000000F0	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	3C	<<<<<<<<<<<<<<<<
00000100	3C	3C	3C	3C													<<<<

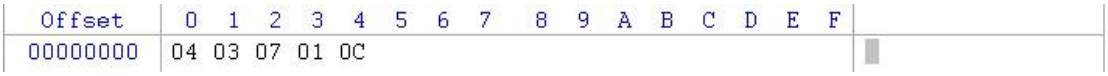
解析：计算 1-2 行数据的大小，以小尾的形式表示出来.
这个文件一共只有 2 条数据。也就是说，第一条指令输出该文件的总大小。
提示：请尽量多的使用该语句。

第 11 课:计算类指令_cal,f

难度：中等
使用频率：低
说明：该类指令一共有 5 条指令分别为
_cal,f+
_cal,f-
_cal,f*
_cal,f/
_cal,f%
例 1：

```
_num,4,,,8;
_num,3,,,8;
_cal,f+,8,0,1,2;
_cal,f-,8,0,1,2;
_cal,f*,,0,1,2;
```

输出结果：



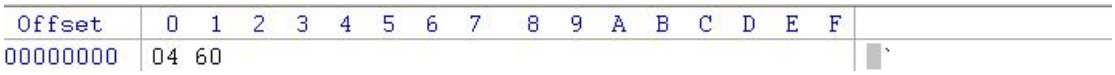
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	04	03	07	01	0C												

解析：
第三个参数和第四个参数都指向函数 ID
第一条数据为 4.第二条数据为 3
第三条数据=第一条数据+第二条数据。所以为 7.
第四条数据=第一条数据-第二条数据。所以为 1
第五条数据=第一条数据*第二条数据。所以为 C（12）

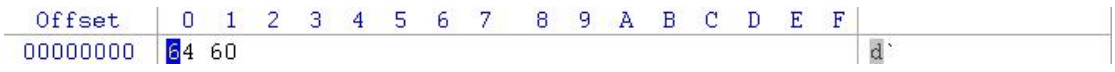
提示：在计算除法语句时候，请避免除数为 0.

第 12 课:计算类指令_cal,n

难度：中等
使用频率：低
说明：与_cal,f 不同的是，参数 3 和参数 4 只有一个参数是函数 ID，一个参数是实际值。而_cal,f 两个参数都是 ID 值
该类指令一共有 8 条指令，分两组，他们是
第一组：_cal,n+ 一共有 5 条指令
_cal,n+,32,0,3,200;
_cal,n-,32,0,3,200;
_cal,n*,32,0,3,200;
_cal,n/,32,0,3,200;
_cal,n%,32,0,3,200;
其中第三个参数表示 ID 值，第四个参数表示具体数值，
第二组：_cal,-n 一共有三条指令
_cal,-n.32,0,200,3;
_cal,/n,32,0,200,3;
_cal,%n,32,0,200,3;
其中第三个参数表示具体值，第四个参数表示 ID 值。
例 1
_num,4,,,8;
_cal,-n,8,0,100,1;
输出数据：



解析：ID2 表示 100-参数 1 的值。
ID1=4，100-4=96，也就是 16 进制的 60.
例 2
_num,100,,,8;
_cal,n-,8,0,1,4;



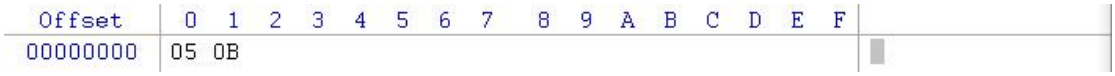
解析：
ID2 表示 ID1 值-4.
ID1=100，100-4=96，也就是 60H

第 13 课:删除语句指令_DelToEnd

难度：中等
使用频率：中
说明：在处理_cal,n，_cal,f 类指令的时候经常会需要一些临时数据(用于其他数据计算但是不用于输出的数据) 因此可以把临时数据写在_DelToEnd 后面。
例 1：

```
_num,5,,,8;
_cal,n+,32,0,4,1;
_DelToEnd,0;
_cal,n*,32,0,1,2;
```

输出数据:



解析：
ID1=5
ID4=ID1*2
ID2=ID4+1
第 4 条数据为临时数据，我们不需要输出，因此 ID3=_DelToEnd,0;
0 为强制删除，因此只输出 1-2 行数据。

第 14 课:空指令_nop _nopto

难度：中等
使用频率：低
说明：该空指令不输出任何数据，不对任何数据做计算，但是会占用行数。
在对一些绝对偏移的脚本做删除的时候，由于不能更改行数信息（否则一些绝对偏移计算地址都会改变） 可以用_nop 来补充。或者在不使用相对偏移方法进行编码的时候，可以使用_nopto 方法来确定后面函数所在的 ID 行。
例 1：

```
_cal,size,32,0,201,201;
[<这里可以填写少量其他代码，而不影响 ID1 的计算>]
_nopto,200;
_str,AAAAAA,,;
```

第 15 课:选择指令_choice

难度：较难
使用频率：低
说明，用来从几个数据里面随机选择。

_choice,3,5;
_DelToEnd,0;
_str,AAAAAA,,;
_str,BBBBBB,,;
_str,CCCCCC,,;
输出数据

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	43	43	43	43	43	43											CCCCCC

解析：该条指令表示从 ID3-ID5 中随机选择一个作为输出，如上图选择了 CCCCCC。

第 16 课:选择指令_when

难度：较难
使用频率：低
说明，用来从 2 个数据里面做选择。
例 1
_when,8,1,3,4,5;
_DelToEnd,0;
_num,0,,,8;
_num,255,,,8;
_num,254,,,8;
输出数据

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	FF																ÿ

解析：由于第三条指令非 0，所以输出第四条数据，为 FF
例 2
_when,8,1,3,4,5;
_DelToEnd,0;
_num,0,,,8;
_num,255,,,8;
_num,254,,,8;
输出数据

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	FE																þ

解析：由于第三条指令非 0，所以输出第四条数据，为 FD

第 17 课:复制指令_copy

难度：困难
使用频率：高
说明：用来复制数据块的指令。

例 1：

```
_str,A,,;
_copy,1,1,7,0;
_copy,1,2,7,1;
```

输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000010	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000020	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000030	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA

解析：

ID2 表示 ID1 复制 7 次（加上自己就是 8 次）

ID3 表示 ID1+ID2 然后输入 7 次（加上自己是 8 次）

也就是说该条语句表示输出 8*8 个 A，如图。

提示：为了生成有效的数据，请合理控制优先级的使用。

例 2:

```
_str,A,,;
_copy,1,1,0,0;
```

输出数据

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000010	41	41	41	41	41	41	41	41									AAAAAAAA

为什么这里输出了 24 次呢？因为次数为 0 则表示输出次数随机，随机值为[0，99]

第 18 课:计算复制指令次数指令_CountOfCopy

难度：中等

使用频率：高

说明：用于计算复制指令的次数，一般与复制指令_Copy 一起使用

例 1

```
_str,A,,;
_copy,1,1,0,0;
_CountOfCopy,8,0,2,3;
```

输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000010	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000020	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000030	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000040	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000050	41	41	41	41	41	41	41	41	5A								AAAAAAAAZ

解析：一共有 58H 个 A，也就是说复制了 57H 次，（减去原来的一次）_CountOfCopy 的第四个参数为 3，即+3，因此为 5A。

提示：指向的指令一定是_Copy 类型.

第 19 课:数组指令_array

难度：困难
使用频率：高
说明：该指令可以定义固定数组，随机数组，最高支持 4 维数组。
例 1：
生成一个[20][10]的数组
代码：

```
_array,2,20,10,1,1;
```

输出代码：

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	04	01	48	08	7E	FC	F0	46	3E	04	78	4E	3C	02	50	59
00000016	44	01	48	57	7E	82	70	65	3E	04	78	5D	40	FF	90	59
00000032	44	00	48	08	7E	FC	F0	65	3E	04	78	4E	3C	02	50	59
00000048	44	FE	48	57	42	7F	70	46	41	80	78	F8	55	7C	50	10
00000064	52	84	00	08	7E	FC	70	65	3E	04	78	5D	40	FF	90	59
00000080	84	00	48	08	7E	FC	F0	46	3E	04	78	4E	3C	02	50	59
00000096	44	01	48	08	42	7F	70	65	3E	80	2F	F8	4E	40	50	10
00000112	59	44	00	88	57	7E	7F	70	65	3E	04	78	4E	40	FF	90
00000128	52	84	01	48	08	7E	7F	70	F0	56	81	2F	F8	4E	40	FF
00000144	90	59	44	01	88	57	42	7F	F0	56	41	04	78	4E	40	FF
00000160	50	59	44	01	48	88	54	82	22	F0	46	41	82	22	F0	46
00000176	41	2F	78	5D	3C	02	90	59	3F	FE	48	57	7E	FC	70	46
00000192	41	04	78	4E	40	FF	90	52								

解析：输入固定的数据，所以 4 个数组参数都不应该变异，所以变异选项设置为 0。由于是二维数组，因此后 2 维度设置为 1 即可。
例 2：生成某格式图片数据，长为 0-40 像素，宽为 0-50 像素；一个像素占用 3 个字节。
代码：

```
_array,2,40,50,3,1;
```

解析：长度和宽度是应该变异的，所以变异选项设置为 0，总大小是长度*宽度的三倍，所以第四个参数设置为 3，第五个设置为 1.

提示：该指令可理解为_copy 指令的补充。
如下面代码

```
_num,0,1,0,8;
_copy,1,1,0,0;
_copy,1,2,0,0;
_copy,1,3,0,0;
_copy,1,4,0,0;
```

可以写成

```
_array,4,100,100,100,100;
```

第 20 课:计算数组指令次数指令_CountOfArray

难度：中等
使用频率：高
说明：用来计算数组指令生成的次数。
例 1

```
_CountOfArray,32,0,20,1;
_CountOfArray,32,0,20,2;
_array,2,10,10,1,1;
```

输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	00	00	00	06	00	00	00	06	08	65	41	80	78	55	02	90
00000016	52	FE	88	54	82	F0	56	04	78	4E	02	90	52	FE	88	57
00000032	82	70	56	04	78	4E	02	50	59	FE	48	57				

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	00	00	00	02	00	00	00	04	56	52	84	00	88	57	42	7F

解析:id1 id2 的值分别是数组的一维度的值和二维度的值。
提示：所指向的语句必须为_array

第 21 课:结构模块指令 block

难度：困难
使用频率：高
说明：用来定义数据结构格式。大括号内的偏移是相对于”{”相对偏移。该结构支持最多 16 次的嵌套。
例 1:

```
block,val1;
{
_str,AAAA,,;
    block,val2;
    {
        _str,BBBB,,;
    }
    block,val3;
    {
        _str,CCCC,,;
    }
}
```

输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	41	41	41	41	42	42	42	42	43	43	43	43					AAAABBBBCCCC

解析：本例子讲述了 block 的用法

例 2

```
_str,AA,,;
    block,val;
    {
        _str,BBBB,,;
        _cal,size,32,,1,1;
    }
```

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	41	41	42	42	42	42	04	00	00	00							AABBBB

解析：_cal 指令在括号内，所以他指的 ID1 指令的长度是”BBBB”的长度，而不是”AA”

提示：

1 block 不能单独使用，后面一定要紧跟{}使用。

2 {}可以单独使用，表示偏移为相对偏移的数据块。

3 再定义复杂结构的时候，block 结构可起到事半功倍的作用。

第 22 课:结构复制指令 copy

难度：困难

使用频率：高

说明：用来对 block 指令进行复制。

例 1

```
copy,val1,10,0,10;
_DelToEnd,0;
block,val1;
{
_str,A,,;
copy,val2,2,0,0;
block,val2;
{
_str,B,,;
}
}
```

输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	41	42	42	42	41	42	42	42	41	42	42	42	41	42	42	42	ABBBABBBABBBABBB
00000016	41	42	42	42	41	42	42	42	41	42	42	42	41	42	42	42	ABBBABBBABBBABBB
00000032	41	42	42	42	41	42	42	42									ABBBABBB

提示：

1 复制的区间代码不要有_DelToEnd 语句。

2 注意使用优先级，否则无法得到想要的结果。

第 23 课:计算结构复制指次数的指令 CountOfCopy

难度：中等
使用频率：高
说明：用来对 block 指令进行复制。
例 1

```
CountOfCopy,8,0,2,1;
copy,val,100,1,0;
block,val;
{
_str,A,,;
}
```

输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	1C	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAA
00000016	41	41	41	41	41	41	41	41	41	41	41	41	41				AAAAAAAAAAAAA

第 24 课_DelToBracket 指令

难度：中等
使用频率：高
说明：用来删除结构内后面的指令。不影响结构外的数据。
声明：该指令必须在 block 结构内使用。用于调试语句用

例 1：

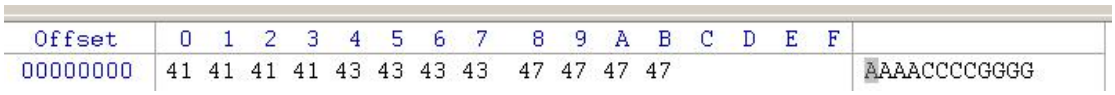
```
block,all;
{
    block,1;
    {
     _str,AAAA,,;
    _DelToBracket;
    _str,BBBB,,;
    }

    block,2;
    {
    _str,CCCC,,;
    _DelToBracket;
    _str,DDDD,,;
    }
```

```
    _str,GGGG,,;
    _DelToBracket;

    block,3;
    {
    _str,EEEE,,;
    _DelToBracket;
    _str,FFFF,,;
    }
}
```

输出数据:



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	41	41	41	41	43	43	43	43	47	47	47	47					AAAACCCCGGGG

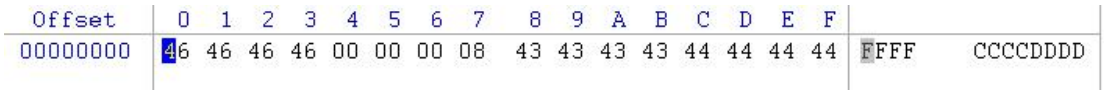
说明：红色为被删除指令。

第 25 课 addr 指令

难度：中等
使用频率：高
说明：用来计算 block 结构的偏移
例 1：

```
block,BBB;
{
    _str,FFFF,,;
}
addr,32,1,AAA,0;
block,AAA;
{
    _str,CCCC,,;
    _str,DDDD,,;
}
```

输出数据：



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	46	46	46	46	00	00	00	08	43	43	43	43	44	44	44	44	FFFF CCCCDDDD

第 26 课 size 指令

难度：中等
使用频率：高
说明：用来计算 block 结构的偏移
例 1：

```
block,BBB;
{
    _str,FFFF,,;
}
size,32,1,CCCC,0;
block,AAA;
{
    _str,CCCC,,;
    block,CCCC;{_str,DDD,,;}
}
```

输出数据：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	46	46	46	46	00	00	00	03	43	43	43	43	44	44	44	

第 27 课 choice 指令

难度：中等
使用频率：高
说明：从 block 结构内随机选去一个 block 子结构。
要求必须指向 block 结构，并且 block 必须有子结构

例 1

```
choice,sum;
block,sum;
{
    block,A;
    {
        _str,A,,;
    }
    block,B;
    {
        _str,BB,,;
    }
    block,C;
    {
        _str,CCC,,;
    }
    block,D;
    {
        _str,DDDD,,;
    }
    block,E;
```

```
    {
        _str,EEEEE,,;
    }
}
```

输出数据

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	42	42	41	42	42	43	43	43	44	44	44	44	45	45	45	45	BBABBCCCDDDDEEEE
00000010	45																E

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	45	45	45	45	45	41	42	42	43	43	43	44	44	44	44	45	EEEEEABBCCCDDDDE
00000010	45	45	45	45													EEEE

第 28 课 CRC32,CRC16 指令

难度：中等

使用频率：中等

说明：对结构数据进行 crc32 计算，支持嵌套。

(CRC16 使用方法类似)

例 1：

```
block,002;
{
crc32,0,001,0;
block,001;
{
_hex,CCCCCCCCCCCCCCCC;
}
}
crc32,0,002,1;


block,003;
{
_hex,B88BEA58CCCCCCCCCCCCCCCC;
}
crc32,0,003,0;
```

输出内容：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	B8	8B	EA	58	CC	CC	CC	CC	CC	CC	CC	CC	A9	71	75	89	¸‹êXÌÌÌÌÌÌÌÌ©qu‰
00000010	B8	8B	EA	58	CC	CC	CC	CC	CC	CC	CC	CC	A9	71	75	89	¸‹êXÌÌÌÌÌÌÌÌ©qu‰

代码编写实战

第一课 MIDI 格式

了解 MIDI 文件格式：
下面用来编写一个 mid 的文件格式的脚本。

mid 文件格式简介：
一个 MIDI 文件基本上由 2 部分组成，头块和轨道块。

头块：
头块出现在文件的开头，头块看起来一般是这样的：
4D5468640000 0006 FFFF NNNN DDDD
4D5468640000 表示头块的表示值
FFFF 是文件格式，有三种格式。
NNNN 是 MIDI 文件中的轨道数。
DDDD 是每个四分音符节奏数

轨道块：
4D54726B XXXXXXXX AAAAAAAAAAAAAAAAAAAAAA
4D54726B 表示轨道块的标识值
AAAAAAAAAAAAAAAAAAAAAA 表示轨道块
XXXXXXXX 表示轨道快的大小。

对文件格式建模，编写解析代码
下面是我们编写的解析代码
_str,MThd,,6,,;[<头信息为 MThd，长度为 6 的字符串类型>]
_cal,size,16,1,1,1;[<第一行代码解析字符串的长度，大尾，16 位>]
_num,,1,,16;[<一个 16 位的可变数据>]
_num,,1,,16;[<如上>]
_num,,1,,16;[<如上>]
_str,MTrk,,4,,;[<类似第一行>]
_cal,size,16,1,8,8;[<取缔把行代码生成数据的大小>]
_str,,1,,,;[<一个会变异的字符串，用来表示轨道块>]

第二课 BMP 格式（1/2）

和 midi 格式相比，BMP 格式较为复杂，具体文件格式请自行学习，下面是一段 BMP 格式的解析脚本：

```
_str,BM,,;
_cal,size,32,,1,20;[<all size>]
```

```
_num,0,,,32;
_cal,addr,32,,16;
_cal,size,32,,5,15;
_CountOfCopy,32,0,20,1;[<x>]
_num,16,,,32;[<y>]
_num,1,,,16;
_num,24,,,16;
_num,0,,,32;
_cal,size,32,,16,20;
_num,4,,,32;
_num,4,,,32;
_num,0,,,32;
_num,1,,,32;
_num,,1,,8;[<16>]
_num,,1,,8;
_num,,1,,8;
_copy,16,18,15;
_copy,19,19,0;
```

该段脚本一共有 20 行，
第 1-4 行为一个结构，我们定义为结构 A
第 5-15 行结构 B
第 16-20 行表示结构 C
第 2 行表示文件的总大小。
第 4 行则表示结构 C 的偏移。
第 5 行表示结构 B 的总大小。
第 11 行表示结构 C 的大小.
第 6 行表示第 20 行出现的次数+1
第 19 行则表示第 18 行复制 15 次（加上 18 行就是 16 次，也就是宽度 16 像素）
其中第 20 行表示第 19 行复制 n 次，n∈[0,99]
也就是说，第 9 行表示长度像素值。值为[1-100]

第三课 BMP 格式（2/2）

上一课的 BMP 格式，维护起来不是很方便，因为 3 个结构都是绝对偏移。
这一课就来讲解如何把 alpha 代码写的漂亮一些.
既然是 3 个结构，我们就先定义这 3 个结构为 structA,structB,structC

```
block,structA;
{}
block,structB;
{}
block,structC;
{}
```

编译下，成功，可以继续写。首先写 C 结构。

block,structC;
{
_array,2,16,100,3,1;
}
编译下，没错误，我们继续写 B 结构，因为 BC 之间关联较大。我们把 BC2 个结构放一起。这里主要优化一些计算语句，普通的 copy 用数组来代替。

```
block,structBC;
{
_cal,size,32,,1,11;
_CountOfArray,32,0,12,1;
_CountOfArray,32,0,12,2;
_num,1,,,16;
_num,24,,,16;
_num,0,,,32;
_cal,size,32,,12,12;
_num,4,,,32;
_num,4,,,32;
_num,0,,,32;
_num,1,,,32;
_array,1,100,16,3,1;
}
```

编译一下，没错误。这时候我们再处理 A 结构，只主要把第二条语句行数修改即可。

```
block,structA;
{
_str,BM,,;
_cal,size,32,,1,16;[<all size>]
_num,0,,,32;
_cal,addr,32,,16;
}
```

编译一下 没问题，这时候我们就把他们组装到一起。

```
block,structA;
{
_str,BM,,;
_cal,size,32,,1,18;[<all size>]
_num,0,,,32;
_cal,addr,32,,18;
}
block,structBC;
{
_cal,size,32,,1,11;
_CountOfArray,32,0,12,1;
_num,16,,,32;
_num,1,,,16;
```

```
_num,24,,,16;
_num,0,,,32;
_cal,size,32,,12,12;
_num,4,,,32;
_num,4,,,32;
_num,0,,,32;
_num,1,,,32;
_array,1,100,16,3,1;
}
```

常见问题

问：为什么有的函数前面有“_”有的没有？

答：一般语句前面都有_来表达，但是对结构进行操作的语句，名称前面是没有_的。

问：平时提示某行语句错误，一般从哪里去排错呢？

答：可以从如下方面进行排错。

1 语句名称写的是否正确？是否注意了大小写？

2 语句之间的标点符号是否是半角符号？

3 参数数量是否正确？

4 各个参数值是否有效。

问：强制生成样本数是什么意思？意义大不大？

答：程序生成样本数只是对样本逻辑进行粗略的计算生成，如果你的时间充足，建议勾选强制生成样本数，并且根据自己的时间，以及设备配置最大限度的设置样本生成量。

问：为什么样本最大之支持 1M 呢？

答：样本太大会影响 fuzz 过程的速度，而几乎所有的样本都不需要超过 1M 的容量

附录 1 约束类指令优先级

从高到低分别是：

_shoice

_cal,size

_cal,addr

size

addr

_cal,f 系列 n 系列

_crc16

_crc32

_when

_copy

choice
copy

优先级高的指令是不能对优先级低的指令做运算！

附录 2:NPFUZZ 模块的控制指令

Recv;

接受指令：用于接受远程发来的信息，该指令没有参数。

SendAll;

发送指令:对数据建模窗口生成的数据进行发送，每发送一次，生成的数据变异一次，该指令没有参数。

SendHex;

二进制数据发送指令，用于发送指定的二进制数据。该函数一个参数，用于表示要发送的二进制数据（16 进制形式编写），二进制数据大小应小于 128 字节。
如：
SendHex,313233343536;

SendTxt;

文本数据发送指令，用于发送指定的二进制数据。该函数一个参数，用于表示要发送的文本数据，文本数据应小于 128 字节。

SendXss;

Xss 数据发送指令，用于随机发送 Xss 数据。该函数没有参数。

Sleep;

休眠指令，该函数有一个参数，单位为毫秒，区间要求：[0,3600000]。可用于指令间的暂停。
如
Sleep,200;
休眠 200 毫秒。

Msg;

消息指令，用于弹出指定的消息框，该函数一个参数，用于表示消息的内容。如：
Msg,Hello;

SendSql;

SQL 数据发送指令，用于随机发送 SQL 数据。该函数没有参数。

SendArray;

发送数据[Array 窗口]指令。Array 窗口数据为 C 数组类型。方便 Wireshark 抓包工具对数组的导出。也可以通过导入文件格式的形式导入数据。
该函数没有参数，数据大小不得大于 1KB。

SendArrayVary;

对数组数据进行变异再发出，变异方法类似于 filefuzz 通用 fuzz 模块的引擎 1。数据大小应该保持在 4 字节-10k 字节，该函数没有参数。
不适合有校验结构的数据使用。

SendArrayVaryS;

对数组数据进行变异再发出，变异方法类似于 filefuzz 通用 fuzz 模块的引擎 2。数据大小应该保持在 128 字节-10k 字节，该函数没有参数。
不适合有校验结构的数据使用。

SendArrayVaryDiy;

对数组数据进行自定义变异再发出，该函数有 2 个参数，分别表示开始变异地址和结束变异地址。只在地址区间进行变异，非区间不进行变异。其他和 SendArrayVary 相同。
不适合有校验结构的数据使用。

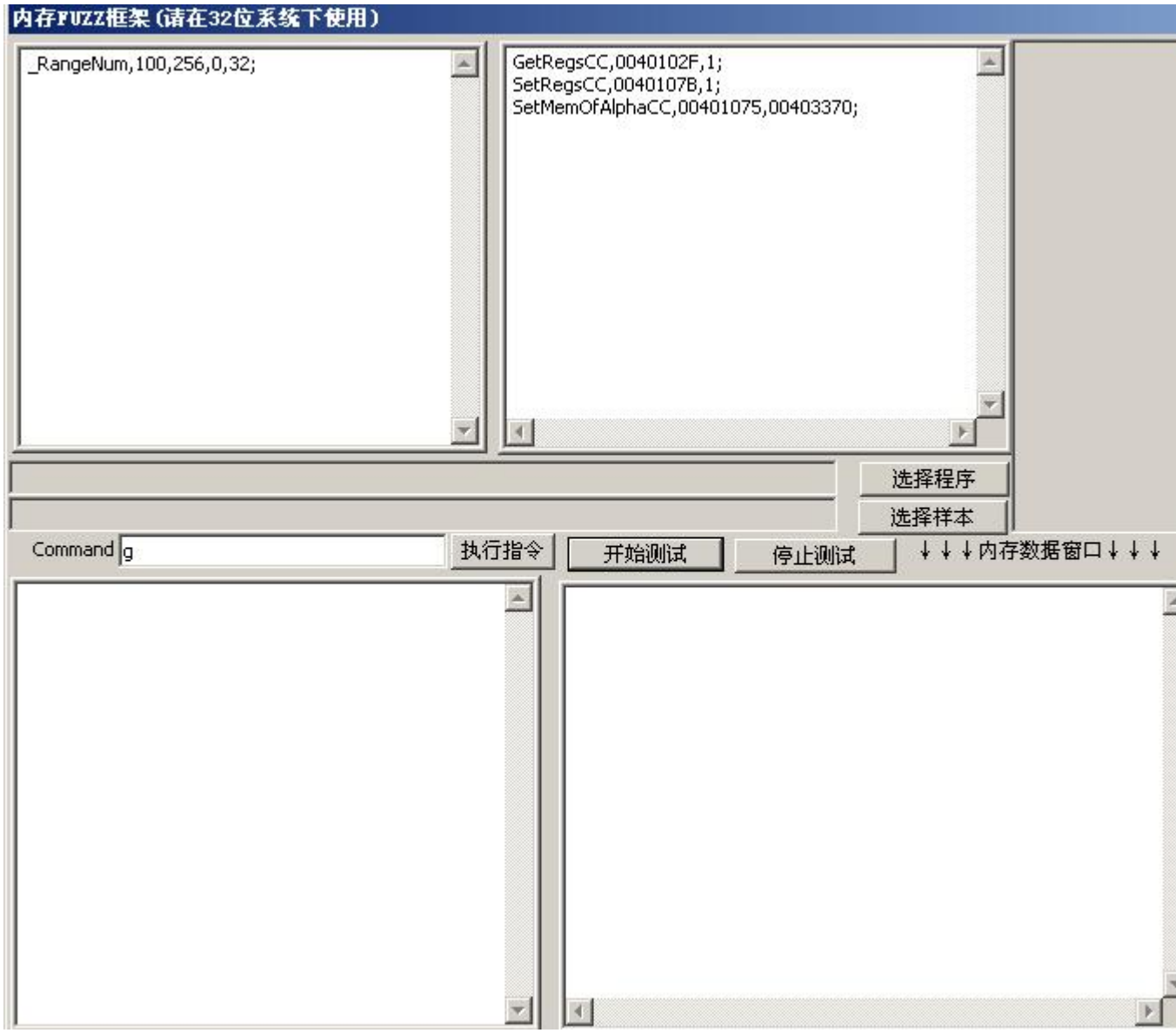
SendArrayVarySDiy;

对数组数据进行自定义变异再发出，该函数有 2 个参数，分别表示开始变异地址和结束变异地址。只在地址区间进行变异，非区间不进行变异。其他和 SendArraySVary 相同。
不适合有校验结构的数据使用。

SendArrayVaryAll;

对数组数据进行自定义变异再发出（引擎 3），该种变异方法需要大量的变异，需要较长的 fuzz 时间，一般情况下不建议使用。
不适合有校验结构的数据使用。

附录 3:内存 FUZZ 模块的 Command 指令



Command 指令填写在 command 后的编辑框中，填写后点击执行指令方可执行。截止到 1.6 版本，Command 支持如下 command 指令。

run

运行指令，表示让程序运行。该指令无参数。

g

让程序处于调试状态并且跑起来的指令。该指令无参数。

kill

结束指令，用于杀死当前线程。该指令无参数。

bp

下断点指令，用于在指定地址下断点，该指令 1 个参数，表示要下的断点地址，如 bp 004010cc。

bl

列出所有断点的指令，用于列出当前测试环境下的所有断点，该指令无参数。

DelLog

删除 Log 日志指令，用于删除当前 Log 窗口内的日志信息。

bc

删除断点指令，用于删除指定的断点，该指令一个参数，用于表示要删除的断点信息。
如：bc 004010cc.

bcAll

删除所有断点指令，用于删除当前测试环境内的所有断点。该指令没有参数。

so

单步步过指令，单步指令，遇到 call 走过（不进入）。该指令没有参数。

si

单步步入指令，单步指令，遇到 call 进入。该指令没有参数。

trace

跟踪指令，用于跟踪程序。内存 Fuzz 测试时候必用。该指令没有参数。

untrace

停止跟踪指令。用于跟踪指令的终止。该函数没有参数。

附录 4:内存 FUZZ 模块的内存控制指令

用于对内存进行控制，通过对内存控制指令的组合，即可完成高效的内存 fuzz。
使用方法：
1 在内存控制窗口填写内存控制指令系列。如果有需要，在 alpha 窗口填写 alpha 数据生成指令，或在样本窗口加入要变异的样本.
2 点击开始，跟踪进入准备状态
3 在 command 处输入 trace 指令，点击确定，测试开始。

Stop

说明:停止指令，当条件满足时，跟踪状态 停止。
举例:Stop,eax,01000000,>,20;
解析:
参数 2-4：分别表示满足条件
参数 2 表示寄存器，填写小写形式。分别可以为如下寄存器。
eax
ecx
ebx
edx
esi
edi
esp
ebp
eip
参数 3：要进行比较的值，8 位 16 进制的形式，不需要添加”0x”前缀和”H”后缀。
参数 4：比较方法，一共支持 6 种方法，包括：
等于：==
大于：>
小于：<
不等于：!=
大于等于>=
小于等于<=

参数 5：表示满足次数，如该指令表示满足 20 次跟踪才会停止。

StopRange

说明：当指定的寄存器在一某区间内达到一定次数的时候，跟踪停止。
举例：StopRange,eax,01000000,02000000,20;
解析：
该指令有 4 个参数。
参数 1 表示寄存器，同上可选 9 个寄存器。
参数 2 表示寄存器的最小值
参数 3 表示寄存器的最大值。
参数 4 表示满足的次数。
该条指令表示当 eax 的值低 20 次满足[01000000,02000000]的时候，跟踪停止。

StopReg

说明：当某 2 个寄存器满足一定条件后，跟踪停止。
举例：StopReg,eax,ebx,>,20;
解析：
该函数有 4 个参数
参数 1 寄存器名称，可表示 9 个寄存器。
参数 2 寄存器名称，可表示 9 个寄存器。
参数 3 寄存器满足条件，共 6 种。
参数 4 满足条件次数。
该条指令表示，eax>ebx 为满足一次，当程序第 20 次满足条件时候，跟踪停止。

Go

说明：该函数表示跟踪指定步数。
举例：Go,100;
解析：
该函数有一个参数，表示要跟踪指定的步数。达到这个步数自动停止。
如上一条指令表示跟踪 100 步后程序自动停止。

StopChangeReg

说明：该函数表示在指定位置，寄存器发生变化的时候跟踪停止。
举例：StopChangeReg,004010cc,eax,20;
解析：
该函数有 3 个参数
参数 1，寄存器的值。只有当寄存器满足该值的时候，监控指令才会判断与执行。

参数 2，寄存器。表示要监控的寄存器，该寄存器可以表示 8 种寄存器（eip 除外，因为无意义）
参数 3，满足条件次数，只有满足条件达到这个次数，跟踪才会停止。
该函数表示在 004010cc 处，当 eax 改变次数达到 20 次的时候，跟踪停止。

StopRegLp4

说明：该函数表示在指定位置，寄存器发生变化的时候跟踪停止。
举例：StopRegLp4,eax,12345678,>,20
解析：
该函数共有 4 个参数。
参数 1 表示寄存器，寄存器地址指向的数据[4 字节]为要对比的值。
参数 2 表示要和参数 1 对比的值。
参数 3 表示 6 种对比的方式
参数 4 表示满足次数。
例子表示 eax 指向的数据[4 字节]第 20 次大于 12345678H 的时候,程序停止。

StopRegLp2

说明：该函数表示在指定位置，寄存器发生变化的时候跟踪停止。
举例：StopRegLp4,eax,12345678,>,20
解析：
该函数共有 4 个参数。
参数 1 表示寄存器，寄存器地址指向的数据[2 字节]为要对比的值。
参数 2 表示要和参数 1 对比的值。
参数 3 表示 6 种对比的方式
参数 4 表示满足次数。
例子表示 eax 指向的数据[2 字节]第 20 次大于 12345678H 的时候,程序停止。

StopRegLp1

说明：该函数表示在指定位置，寄存器发生变化的时候跟踪停止。
举例：StopRegLp4,eax,12345678,>,20
解析：
该函数共有 4 个参数。
参数 1 表示寄存器，寄存器地址指向的数据[1 字节]为要对比的值。
参数 2 表示要和参数 1 对比的值。
参数 3 表示 6 种对比的方式
参数 4 表示满足次数。
例子表示 eax 指向的数据[1 字节]第 20 次大于 12345678H 的时候,程序停止。

StopMemLp4

说明：该函数表示在指定位置，寄存器发生变化的时候跟踪停止。
举例：StopMemLp4,00401000,12345678,>
解析：
该函数共有 3 个参数。
参数 1 表示一个地址，该地址指向的数据[4 字节]为要对比的值。
参数 2 表示要和参数 1 对比的值。
参数 3 表示 6 种对比的方式
例子表示 00401000 指向的数据[4 字节]大于 12345678H 的时,程序停止。

StopMemLp2

说明：该函数表示在指定位置，寄存器发生变化的时候跟踪停止。
举例：StopMemLp4,00401000,12345678,>
解析：
该函数共有 3 个参数。
参数 1 表示一个地址，该地址指向的数据[2 字节]为要对比的值。
参数 2 表示要和参数 1 对比的值。
参数 3 表示 6 种对比的方式
例子表示 00401000 指向的数据[2 字节]大于 12345678H 的时,程序停止。

StopMemLp1

说明：该函数表示在指定位置，寄存器发生变化的时候跟踪停止。
举例：StopMemLp4,00401000,12345678,>
解析：
该函数共有 3 个参数。
参数 1 表示一个地址，该地址指向的数据[1 字节]为要对比的值。
参数 2 表示要和参数 1 对比的值。
参数 3 表示 6 种对比的方式
例子表示 00401000 指向的数据[1 字节]大于 12345678H 的时,程序停止。

MemShow

说明：该函数用来显示内存的值，显示大小为 256 字节。
举例：MemShow,00401000;
解析：
该函数共有 1 个参数。

参数 1 为一个地址，表示要显示的内存地址。
例子表示数据窗口不间断的显示 00401000 处的数据。

SetMemOfAlpha

说明：该函数用来将生成的 alpha 数据覆盖到指定内存。
举例：SetMemOfAlpha,004010cc,eax,>,22222222;
解析：
该函数共有 4 个参数。
参数 1 为一个地址，表示把数据覆盖到的内存地址。
参数 2-参数 4 表示覆盖的满足条件，其中：
参数 2 表示寄存器的名字。
参数 3 表示满足方法
参数 4 表示一个值。
该例子表示当 eax>22222222 的时候，将 Alpha 代码生成的数据覆盖到地址 004010cc 处。
注意：如若使用该函数，需要输入 alpha 生成代码。

Sleep

说明：该函数一般用来调试用，表示程序休眠一定的时间。
举例：Sleep,50;
解析：
该函数共有 1 个参数。
参数 1 为一个地址，表示要休眠的时间，单位为毫秒。
该例子表示每跟踪一条代码，程序休眠 100 毫秒。

SetFixedMemOfAlpha

说明：该函数用来将生成的 alpha 数据覆盖到指定内存。
举例：SetFixedMemOfAlpha,004010cc,Reg,>,22222222,256;
解析：
该函数共有 5 个参数。
参数 1 为一个地址，表示把数据覆盖到的内存地址。
参数 2-参数 4 表示覆盖的满足条件，其中：
参数 2 表示寄存器的名字。
参数 3 表示满足方法
参数 4 表示一个值。
参数 5：最大覆盖数。
该例子表示当 eax>22222222 的时候，将 Alpha 代码生成的数据覆盖到地址 004010cc 处。
覆盖数不得超过 256 字节。
注意：如若使用该函数，需要输入 alpha 生成代码。

SetSampleData

说明：该函数用来将样本变异后放到指定内存地址上。
举例：SetSampleData,00401000,eax,>,22222222,1;
解析：
该函数共有 5 个参数。
参数 1 为一个地址，表示把数据覆盖到的内存地址。
参数 2-参数 4 表示覆盖的满足条件，其中：
参数 2 表示寄存器的名字。
参数 3 表示满足方法
参数 4 表示一个值。
参数 5：变异方法。1-3，一共 3 种变异方法。
该例子表示当 eax>22222222 时，将样本的数据按照第一种变异方法变异后，存放到指定地址。
注意：如若使用该函数，需要输入 alpha 生成代码。

SetReg

说明：该函数用来设置寄存器的值。
举例：SetReg,Eip（Reg）,004010cc,Reg,>,22222222;
解析：
该函数共有 5 个参数。
参数 1 为寄存器名，表示要设置的寄存器。
参数 2 表示要设置的寄存器的值。
参数 3-参数 5 表示满足条件
该例子表示当条件满足时，设置 eip=004010cc。

SetRegToMemOfAlpha

说明：条件满足时，在指定寄存器的地址存放 alpha 代码生成的数据。
举例：SetRegToMemOfAlpha,eax(指向内存的寄存器),ebx,>,22222222;
解析：
该函数共有 4 个参数。
参数 1 为寄存器名，该寄存器指向的地址用来存放 alpha 生成的数据。
参数 2-参数 4 表示满足条件
该例子表示当条件满足时（ebx>22222222 的时候），把 alpha 代码生成的数据存放到 eax 指向的地址。

SetFixedRegToMemOfAlpha

说明：条件满足时，在指定寄存器的地址存放 alpha 代码生成的数据。
举例：SetRegToMemOfAlpha,eax(指向内存的寄存器),ebx,>,22222222,256;
解析：
该函数共有 5 个参数。
参数 1 为寄存器名，该寄存器指向的地址用来存放 alpha 生成的数据。
参数 2-参数 4 表示满足条件
参数 5 数据字节的限制。
该例子表示当条件满足时（ebx>22222222 的时候），把 alpha 代码生成的数据存放到 eax 指向的地址，数据最大不得超过 256 字节、如果超过 256 字节，就复制 256 字节数据。

GetRegs

说明：条件满足时，获取寄存器系列。
举例：GetRegs,3,Reg,>,33333333;
解析：
该函数共有 4 个参数。
参数 1 保存寄存器数据的位置，0-3。
参数 2-参数 4 表示满足条件
该例子表示当条件满足时获取寄存器组的数据，保存到 3 号变量内。
该函数使用频率较大。

SetRegs

说明：条件满足时，设置寄存器系列。
举例：SetRegs,3,Reg,>,33333333;
解析：
该函数共有 4 个参数。
参数 1 设置寄存器数据的位置，0-3。
参数 2 -参数 4 表示满足条件
该例子表示当条件满足时设置寄存器组的数据，数据来源是 3 号变量。
该函数使用频率较大。

Goto

说明：条件满足时，执行到指定地址
举例：Goto，004010cc,Reg,>,33333333;
解析：

该函数共有 4 个参数。
参数 1 要执行到的地址
参数 2 -参数 4 表示满足条件
该例子表示当条件满足时执行到指定地址 004010cc。
如果该地址不会被执行，程序将会跑起来。

Msg

说明：条件满足时，弹出信息框
举例：Msg,Reg,>,33333333;
解析：
该函数共有 3 个参数。
参数 1-参数 3 表示满足条件
该例子表示当条件满足时弹出信息框。

SleepPro

说明：条件满足时，执行 Sleep 指令。
举例：SleepPro,1000,Reg,>,33333333;
解析：
该函数共有 4 个参数。
参数 1 表示要 Sleep 的时间，单位为毫秒
参数 2-参数 4 表示满足条件
该例子表示当条件满足时 Sleep 1000 毫秒。

SetRegToMemOfSample

说明：条件满足时,对样本进行变异并且存放在指定寄存器指向的地址。
举例：SetRegToMemOfSample,eax(指向地址存放样本),Reg,>,22222222，1(1-3);
解析：
该函数共有 5 个参数。
参数 1 为寄存器表示要存放变异样本的地址。
参数 2-参数 4 表示满足条件
参数 5 表示变异方法，1-3;
该例子表示当条件满足时，对样本进行第一种方式变异，变异后的样本存放在 eax 指向的地址中。

SetFixedRegToMemOfSample

说明：条件满足时,对样本进行变异并且存放在指定寄存器指向的地址。
举例：SetRegToMemOfSample,eax(指向地址存放样本),Reg,>,22222222，1(1-3),256;
解析：
该函数共有 6 个参数。
参数 1 为寄存器表示要存放变异样本的地址。
参数 2-参数 4 表示满足条件
参数 5 表示变异方法，1-3;
参数 6 表示数据大小的上限。
该例子表示当条件满足时，对样本进行第一种方式变异，变异后的样本存放在 eax 指向的地址中，变异后的数据不得超过 256 字节，如果超过只赋值 256 字节。

CC 类指令

前面的控制指令都是在每一步跟踪的基础上进行调试监控的。虽然使用上较为灵活，但是效率有时候较为低下，为了提高效率可以采用 CC 类指令。
如何识别 CC 类指令？如果指令名称以 CC 结尾的话，那一般就是 CC 类指令。CC 类指令之间不采用跟踪形式，因此速度较快。
前面的内存控制指令是在跟踪模式上进行监控的，CC 指令是在断点模式上进行监控的。CC 指令后面的第一个参数为下 CC 断点的地址，后面不一一介绍。
为了稳定，请不要在一条 CC 类指令后面紧跟其他 CC 类指令，也尽量不写在函数开头与结尾。

GetRegsCC

两个参数，表示获取寄存器组，参数 2 为存放寄存器组的变量位置，为 1-4。
GetRegsCC,33333333,1;

SetRegsCC

两个参数，表示设置寄存器组，参数 2 为存放寄存器组的变量位置，为 1-4。
SetRegsCC,33333333,1;

SetMemOfAlphaCC

两个参数，表示 alpha 生成的数据存放到内存中，参数 2 表示要存放的内存地址。
SetMemOfAlphaCC,004010cc[eip],00403000[存放 alpha 数据的地址];
使用该指令需要填写 alpha 代码。

SetFixedMemOfAlphaCC

三个参数,表示 alpha 生成的数据存放到内存中，参数 2 表示要存放的内存地址,参数 3 表示最大可覆盖数。
SetFixedMemOfAlphaCC,004010cc[eip],00403000,4,256;[存放 alpha 数据的地址];
使用该指令需要填写 alpha 代码。

SetRegToMemOfSampleCC

三个参数，表示把样本变异后存放在内存中。参数 2 寄存器指向的地址表示要存放的内存地址，参数 3 表示样本的变异种类，1-3 三种。
SetRegToMemOfSampleCC,004010cc[EIP],Reg(指向地址存放样本),1(1-3);
使用该指令需要添加样本数据。

SetFixedRegToMemOfSampleCC

四个参数，表示把样本变异后存放在内存中。参数 2 寄存器指向的地址表示要存放的内存地址，参数 3 表示样本的变异种类，1-3 三种。参数 4 表示最大可覆数
SetFixedRegToMemOfSampleCC,004010cc[EIP],Reg(指向地址存放样本),1(1-3),256;
使用该指令需要添加样本数据。

GetMemRangeCC

四个参数，表示得到内存数据段，最大 1M.
参数 2 表示读取数据的变量，0-3.
参数 3 表示读取内存的起始地址.
参数 4 表示要读取内存的大小
GetMemRangeCC,004010CC,1[0-3],00401000,00003000;

SetMemRangeCC

四个参数，表示设置内存的数据段，最大 1M
参数 2 表示设置数据的变量，0-3.
参数 3 表示设置内存的起始地址.
参数 4 表示设置内存的大小
SetMemRangeCC,004010CC,1[0-3],00401000,00003000;

SleepCC

两个参数，表示睡眠的时间。
参数 2 表示要睡眠的时间，单位毫秒。
如 SleepCC,33333333,100;

MsgCC

四个参数，表示弹出对话框。
参数 2-参数 4 表示满足条件

附录 5：内存 fuzz 测试例子

内存 fuzz 简介：

按照 fuzz 形式而言，传统的 fuzzer 工具分为二种，一种是文件格式的 fuzz 测试， 一种是网络协议的 fuzz 测试。Alphafuzzer 已经同时支持了这 2 种形式的 fuzz。但是在有些时候，传统的 fuzz 是应用效率并不是很高。比如：
当程序对输入次数有限制（比如输入错误 3 次程序就自动关闭）。我们无法对程序持续的进行 fuzz 测试的时候。
当程序运行需要解压，载入很多很庞大的库文件，我们无法高效的进行 fuzz 测试时候。
当程序（如网络协议）传输过程有数据加密，而我们并不知道加密方式。无法构造有效的数据包进行 fuzz 测试的时候。
当目标程序有非常多的函数，我们只关注某一个函数，而又不想在其他函数上浪费时间的时候。
当我们受不了文件 fuzz 缓慢的速度的时候。
越来越多的特殊情况，导致传统的 fuzz 测试无法有效进行，所以，alphafuzzer 在 1.5 版本开始，增加了内存 fuzz 框架。
内存 fuzz 优点：
适用面广：无论是文件格式的 fuzz 还是网络协议的 fuzz，最终都是在内存内处理，只要我们控制了内存，就可以控制所有。
方便快捷，容易深入：如今各种文件格式，网络协议越来越复杂，各种校验越来越多。传统的 fuzz 很多数据都没有通过校验，造成了资源和时间的浪费。内存 fuzz 测试可以躲避这些校验，直接深入系统函数的最内层进行 fuzz。更精确，更深入，更快捷。
完美的控制程序：去 fuzz 自己喜欢的函数，放弃自己不感兴趣的函数，完美的控制程序。用自己写的内存控制代码，做自己要做的事情。
高效：传统的文件 fuzz 效率缓慢，因为 cpu 执行了大多数对我们来说没有用处的指令。科学的使用内存 fuzz 可以让 fuzz 速度提高几倍，几十倍甚至近百倍（因目标程序而异）。
内存 fuzz 缺点：

对使用者要求较高：想使用内存 fuzz，必须对调试器，软件逆向等有一定的了解，因此门槛较高，对使用者有一定的要求。
对 CPU 要求较高：跟踪测试时候需要较高的 cpu 资源，因此使用者应该有一个强大的 CPU。

例 1：简单的密码破解


MemFuzzTest1.zip

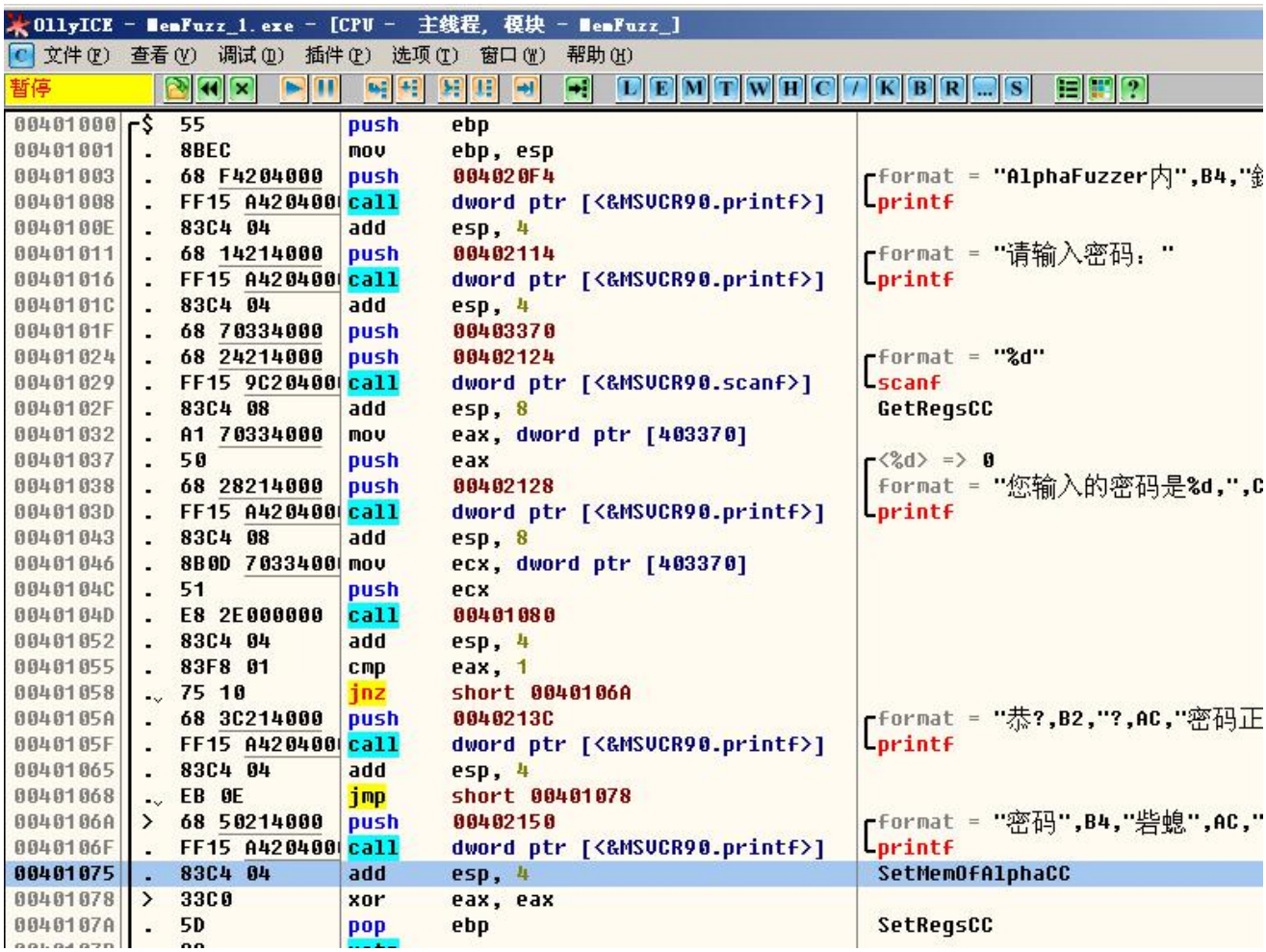
源码：

```
int _tmain(int argc, _TCHAR* argv[])
{
    printf("AlphaFuzzer 内存 fuzz 演示例\r\n");
    printf("请输入密码： ");
    scanf("%d",&sn);
    printf("您输入的密码是%d,\r\n",sn);
    if(cal(sn)==1) printf("恭喜，密码正确\r\n");
    else           printf("密码错误，程序退出\r\n");
    return 0;
}
int cal(int k)
{
    if(k==58)    return 1;
    Else     return 0;
}
```

解析：

该程序运行后，会验证用户输入的密码，密码是 58.输入正确返回正确然后退出。输入错误返回错误信息，然后退出。

逆向分析：

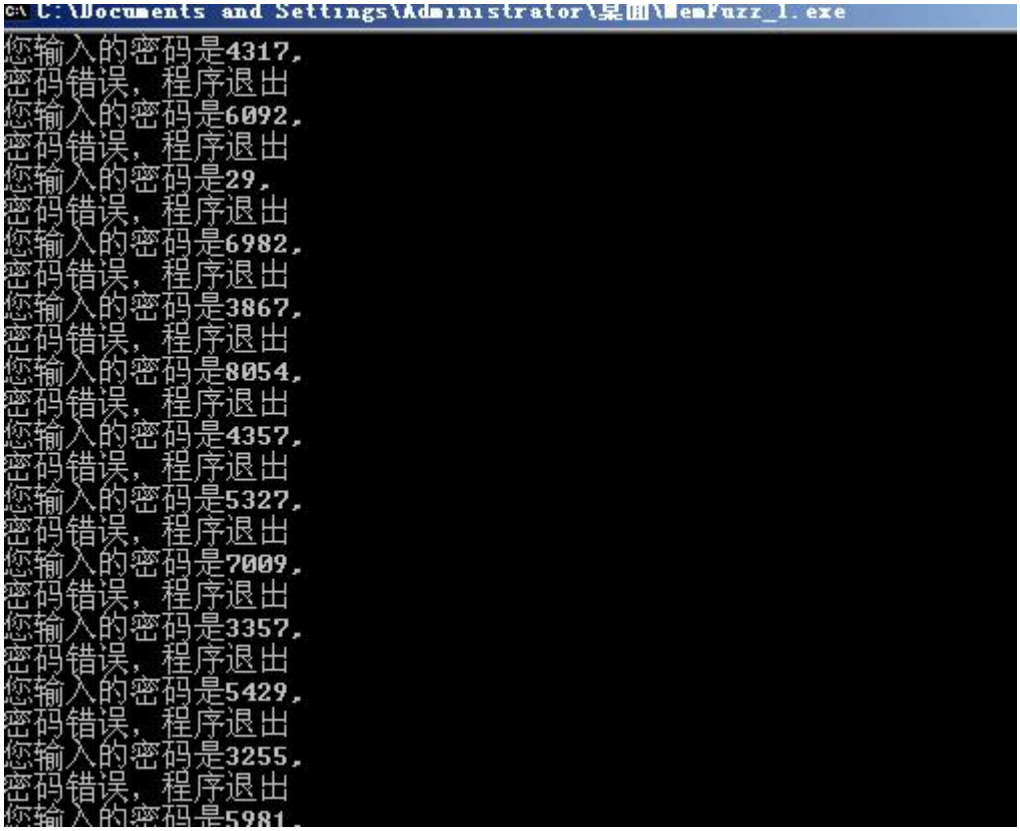


代码编写（性能版）：

内存控制代码：
GetRegsCC,0040102F,1;
SetRegsCC,0040107A,1;
SetMemOfAlphaCC,00401075,00403370;
Alpha 数据生成代码：
_RangeNum,1,10000,0,32;

测试：

第一步：载入程序，填写测试代码。
第二步：点击开始测试
第三步：command 窗口输入 trace 点击确定

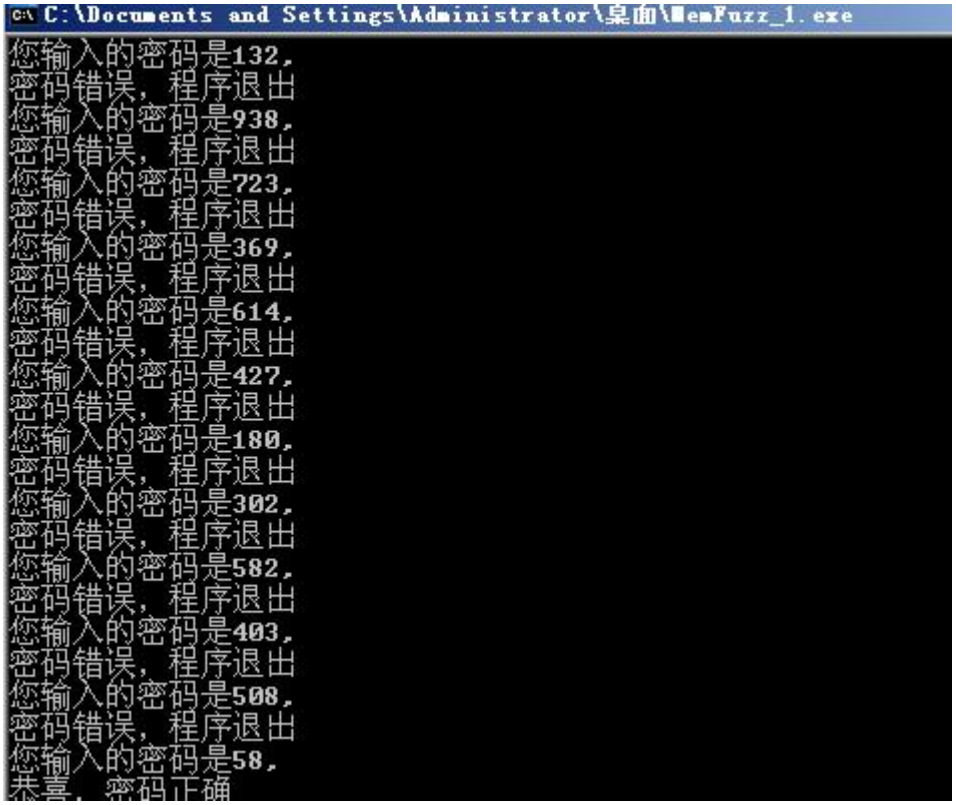


代码编写（功能版）：

GetRegs,,eip,==,0040102f;
SetFixedMemOfAlpha,00403370,eip,==,00401075,4;
SetRegs,,eip,==,0040107A;
MemShow,00403370;
Stop,eip,00401065,==,1;
Alpha 数据生成代码：
_RangeNum,1,1000,0,32;

测试:

可以勾选右下角的跳过 xxx 指令选项，这样可以更快更高效的测试，但是稳定性有所下降。



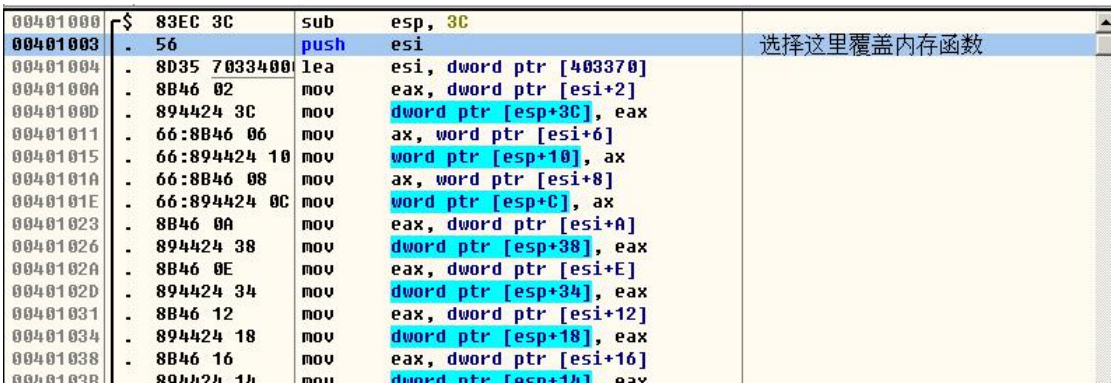
例 2：用 alphafuzzer 内存 fuzz 模拟的图片解析程序

MemFuzzTest2.zip

程序解析：打开文件 C:\sample.bmp 并进行简单解析，解析后输出部分解析结构和图片像素值。

反汇编分析：

```
004011BD   .  50            push    eax                              ┌n
004011BE   .  51            push    ecx                              │src => NULL
004011BF   .  68 70334000   push    00403370                         │dest = MemFuzz_.00403370
004011C4   .  E8 25080000   call    <jmp.&MSVCR90.memcpy>            └memcpy
004011C9   .  E8 32FEFFFF   call    00401000                          解析函数
004011CE   .  68 DC224000   push    004022DC                          选择这里恢复
004011D3   .  FFD6          call    esi
004011D5   .  8B15 7033500  mov     edx, dword ptr [503370]
004011DB   .  83C4 10       add     esp, 10
004011DE   .  52            push    edx                              ┌hObject => NULL
004011DF   .  FF15 1420400  call    dword ptr [<&KERNEL32.CloseHand   └CloseHandle
004011E5   .  A1 78335000   mov     eax, dword ptr [503378]
004011EA   .  50            push    eax                              ┌BaseAddress => NULL
004011EB   .  FF15 1820400  call    dword ptr [<&KERNEL32.UnmapView   └UnmapViewOfFile
004011F1   .  33C0          xor     eax, eax
004011F3   .  5E            pop     esi
004011F4   L. C3            retn
```



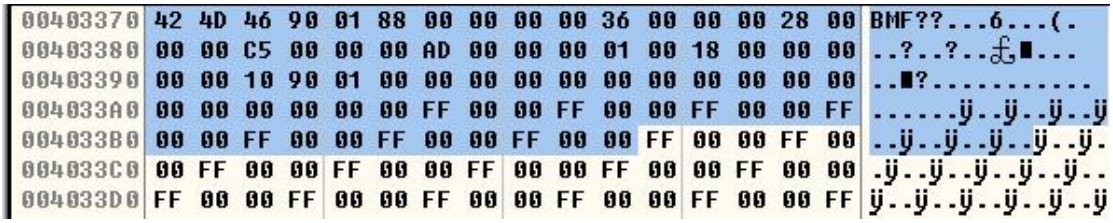
为了稳定性，多个内存覆盖函数尽量不要挨着。因此我们选择 00401003 来作为内存覆盖函数.

代码编写：

1 让程序循环执行。我们选择在 004011C9 选择保存点，在 004011CE 进行恢复。
也就是如下指令：

```
GetRegsCC,004011C9,1;
SetRegsCC,004011CE,1;
```

2 我们构造畸形的 BMP 图片数据之间插入内存中。经过简单分析，内存 00403370 处即为 BMP 样本数据。而我们选择解析函数较为开始的地方，00401003 处进行内存数据的覆盖。



我们可以用下面一条指令表示：

```
SetMemOfAlphaCC,00401003,00403370;
```

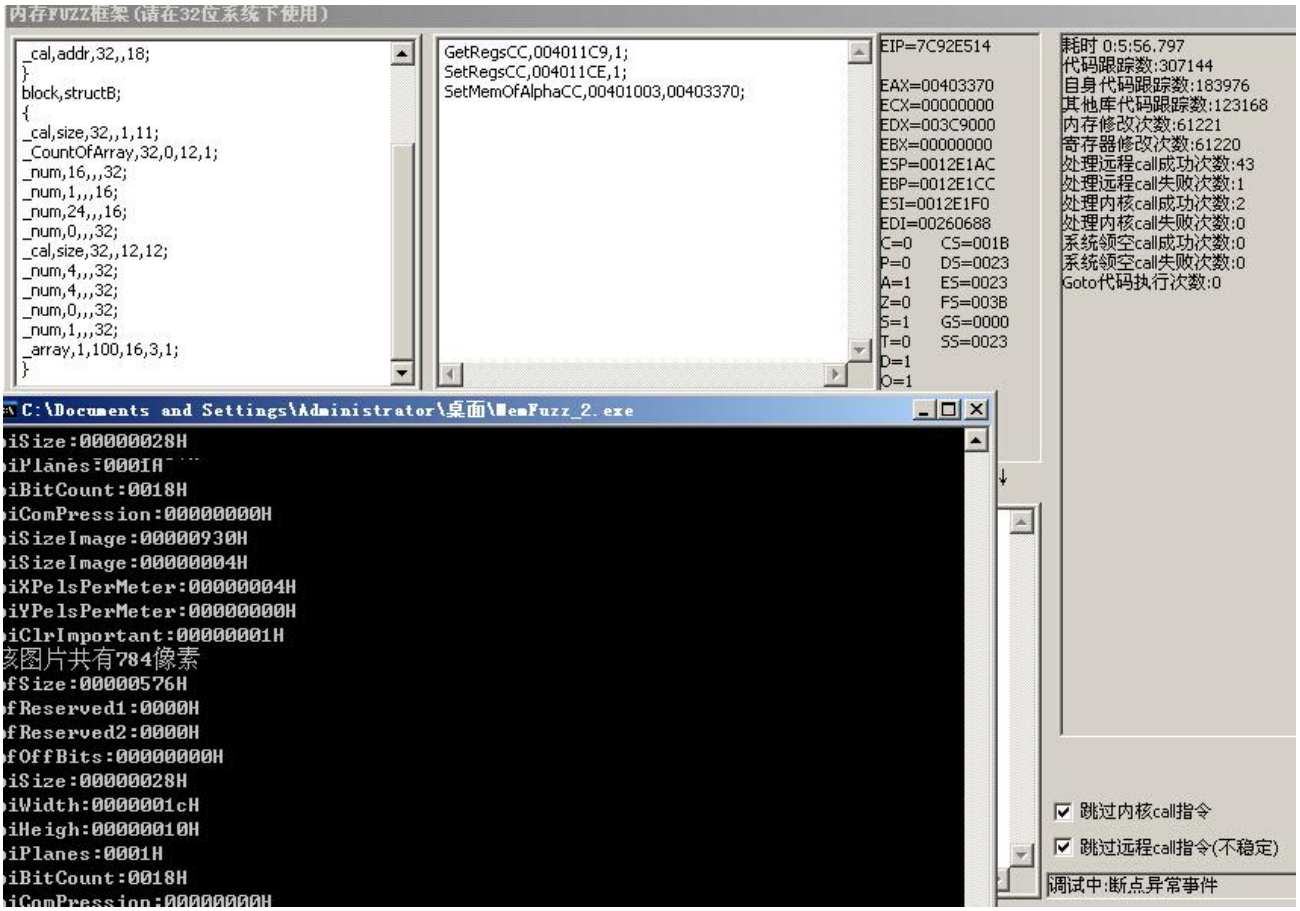
3 我们来输入构造 bmp 数据的 alpha 指令。

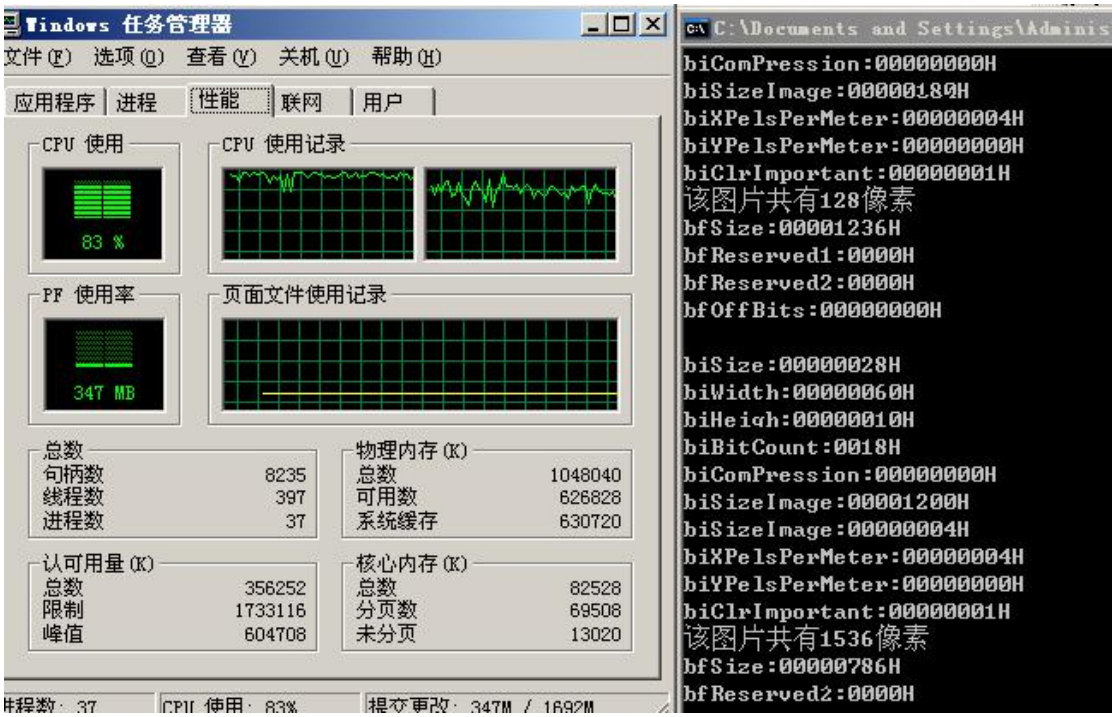
```
block,structA;
{
_str,BM,,;
_cal,size,32,,1,18;[<all size>]
_num,0,,,32;
_cal,addr,32,,18;
}
block,structB;
{
_cal,size,32,,1,11;
_CountOfArray,32,0,12,1;
_num,16,,,32;
_num,1,,,16;
_num,24,,,16;
```

```
_num,0,,,32;
_cal,size,32,,12,12;
_num,4,,,32;
_num,4,,,32;
_num,0,,,32;
_num,1,,,32;
_array,1,100,16,3,1;
}
```

测试：

开始测试-输入指令 trace-执行指令
这时候内存 fuzz 已经开始测试
测试在 xp 虚拟机内执行,传统的文件格式 fuzz，每秒可能只能完成 2-5 个测试。而内存 fuzz 如图所示不到 6 分钟测试了 61221 个测试用例。1 秒测试数量近 200 个。效率是普通文件 fuzz 的几十倍。当然，内存 fuzz 除了可靠的内存控制代码外，还需要较大的 cpu 资源.

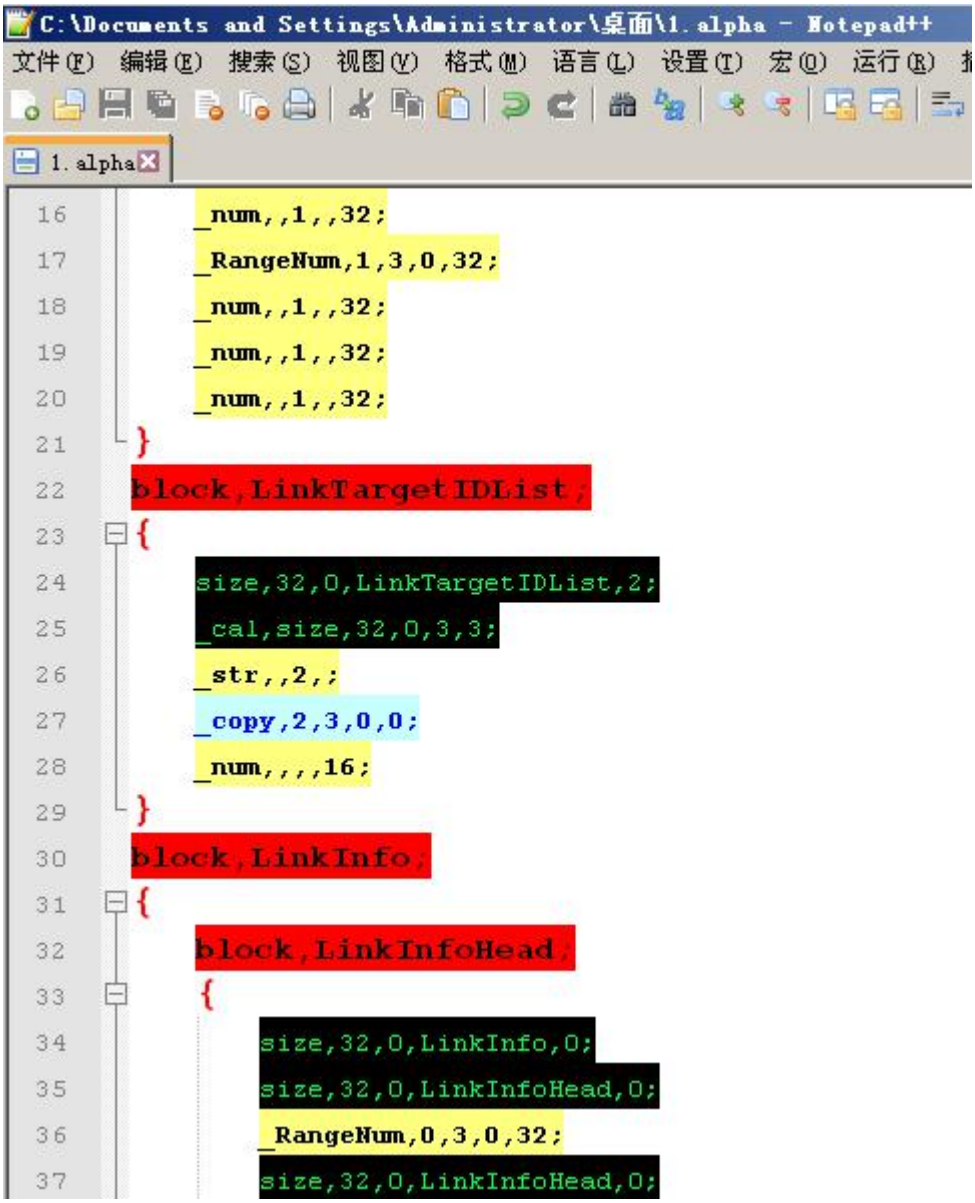


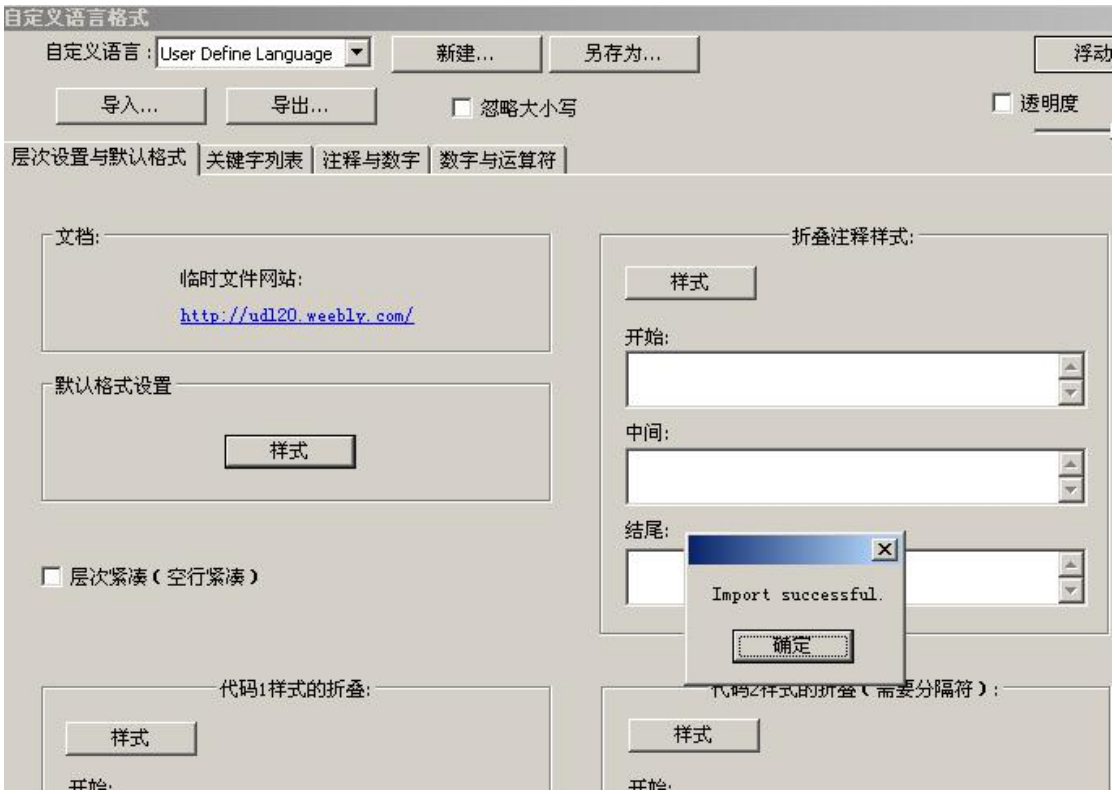


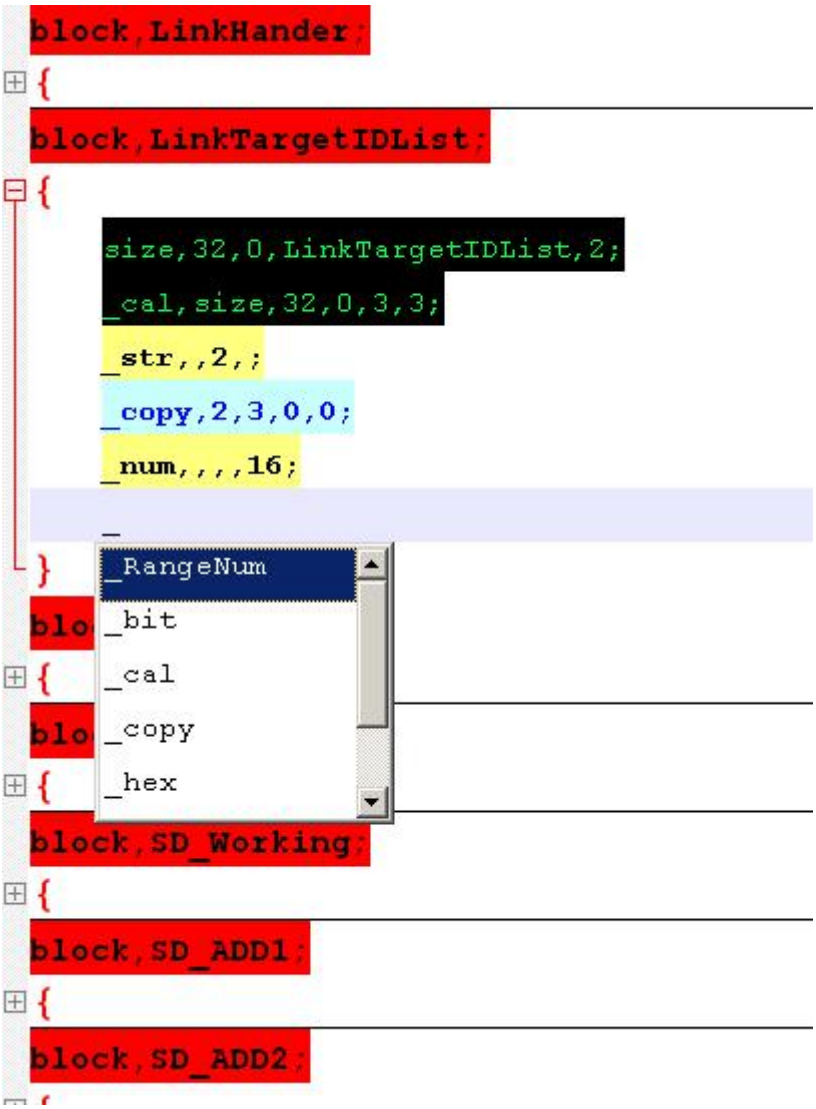
附录 6：Alpha 语言 Notepad++解析脚本（demo 版）

为了方便 alpha 语言的开发速率，因此从 1.6 开始提供一个 Notepad++的 alpha 语言解析脚本。使用它可以快速的进行 alpha 语言开发。用户可以根据自己爱好来修改颜色。

配置方法：语言-自定义语言格式-导入-选择附件中的 alpha.xml 即可。
Alpha 语言保存后后缀名为.alpha


C:\Documents and Settings\Administrator\桌面\1.alpha - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 格式(M) 语言(L) 设置(T) 宏(O) 运行(R)
1.alpha
16 _num,,1,,32;
17 _RangeNum,1,3,0,32;
18 _num,,1,,32;
19 _num,,1,,32;
20 _num,,1,,32;
21 }
22 block,LinkTargetIDList;
23 {
24 size,32,0,LinkTargetIDList,2;
25 _cal,size,32,0,3,3;
26 _str,,2,;
27 _copy,2,3,0,0;
28 _num,,,,16;
29 }
30 block,LinkInfo;
31 {
32 block,LinkInfoHead;
33 {
34 size,32,0,LinkInfo,0;
35 size,32,0,LinkInfoHead,0;
36 _RangeNum,0,3,0,32;
37 size,32,0,LinkInfoHead,0;


自定义语言格式
自定义语言：User Define Language
新建...
另存为...
浮动
导入...
导出...
忽略大小写
透明度
层次设置与默认格式
关键字列表
注释与数字
数字与运算符
文档:
临时文件网站:
http://udl20.weebly.com/
默认格式设置
样式
层次紧凑（空行紧凑）
折叠注释样式:
样式
开始:
中间:
结尾:
Import successful.
确定
代码1样式的折叠:
样式
样式



附录 7：模拟鼠标脚本模块

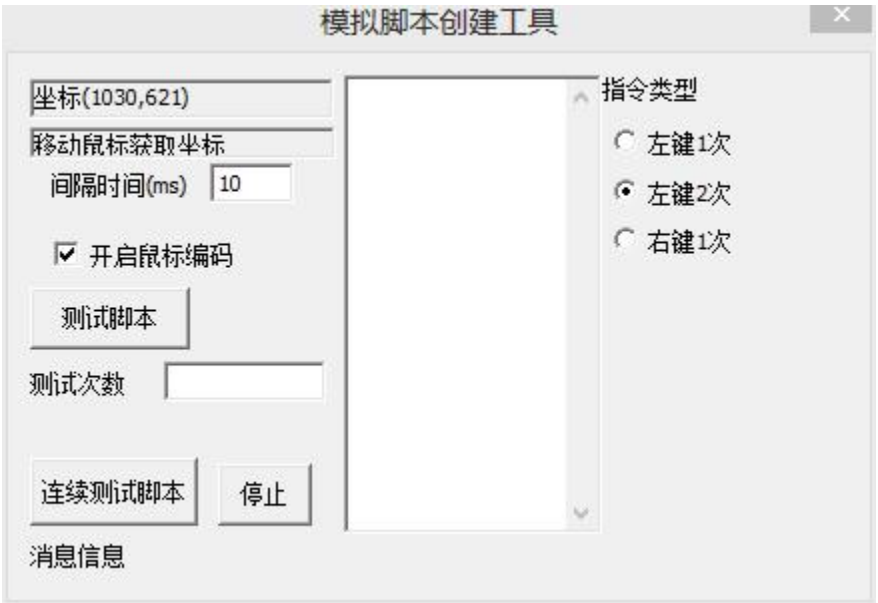
在一些情况下，测试过程需要鼠标点击才可以完成。模拟鼠标脚本实现了这个功能。
工具提供了 2 种鼠标解析脚本，编写代码创建和鼠标创建。目前该功能还属于 demo 版本，只支持四种方法，分别为：
1 鼠标左键单击事件：
如：MouseL1,400,300;
该函数有 2 个参数，分别代表 x 坐标和 y 坐标。表示在(x,y)这个坐标上面鼠标左键 1 次。
2 鼠标左键双击事件：
如：MouseL2,400,300;
该函数有 2 个参数，分别代表 x 坐标和 y 坐标。表示在(x,y)这个坐标上面鼠标左键 2 次。
3 鼠标右键单击事件：
如：MouseR1,400,300;
该函数有 2 个参数，分别代表 x 坐标和 y 坐标。表示在(x,y)这个坐标上面鼠标左键 1 次。

4 Sleep 事件：

如：Sleep,500;
休息的时间，表示按键直接的间隔。

鼠标录制脚本的方法：

如上图所示，如需要鼠标编码需要勾选 2 个地方，一个是左边鼠标编码的选项框，一个是右边指令类型的选项框。够点击后会发现坐标下面有一个倒计时的数字，从 4000 毫秒开始倒计时。如果发现鼠标移动了那么重新倒计时。用户可在倒计时结束之前把鼠标移动到需要录制的坐标，当鼠标 4000 毫秒时间结束后，会有 1000 毫秒的录制时间。这时候用户只需要移动鼠标坐标，即可完成对该地址的录制。



录制结束后可用在文件格式监控引擎里面使用该脚本。该版本最多支持 100 条指令的解析。

下面用一个简单的例子来实验该模块的使用。实验为模拟鼠标打开并关闭我的电脑。

如上图所示，“我的电脑”的关闭坐标是（878,130） 我们可以用下面的代码来表示我的电脑的打开，关闭脚本。

```
MouseL2,37,106;
Sleep,400;
MouseL2,878,130;
Sleep,400;
```