

Forum:[Hack Forums](#)

Thread: [Junk code generator](#)

ctrlaltdestroy_ - September 30, 2013, 7:21 pm

Generates junk code that actually runs! just something fun I did, compiles with fasm and can be linked with C/C++. You might want to push esp ebp before using said code however, as there's nothing stopping it from generating esp/ebp as targets. Not that it wouldn't be easy to do, but I'll leave that up to you ;)

Code int gen_junk(__out_deref(cbOut) void* lpOut, size_t cbOut);
it returns the number of bytes made, which will be within 4 bytes of cbOut (5 bytes is the largest instruction it makes).

It can currently generate:

mov r8/32, imm8/32

logical math instructions:

op r8/32, r8/32

op [e]ax, imm8/32

Sample output:

Code CPU Disasm

Address	Hex dump	Command	Comments
00402040	BA A7163FAB	mov edx,AB3F16A7	
00402045	23EE	and ebp,esi	
00402047	B6 2F	mov dh,2F	
00402049	03E8	add ebp,eax	
0040204B	BE 6FBA7C5D	mov esi,5D7CBA6F	
00402050	02EE	add ch,dh	
00402052	B2 CF	mov dl,0CF	
00402054	21C9	and ecx,ecx	
00402056	BE 9F8BF6F8	mov esi,F8F68B9F	
0040205B	02ED	add ch,ch	
0040205D	B6 5B	mov dh,5B	
0040205F	22CA	and cl,dl	
00402061	BA C73ECCA0	mov edx,A0CC3EC7	
00402066	02EF	add ch,bh	
00402068	BA F788A4CF	mov edx,CFA488F7	

Code ; the assembler/composer:

```
;
;
; basic opcode structure:
;
;   unused
;   _| direction
;   |||
;   0 0 0 0 0 0 0
;   |_| | |
;   |||
;   operation | |
;   accumulator |
```

```

; wide
;
; this applies to all opcodes < 0100 (if opcode & 7 < 6)
;
; operation:
; specifies a logical operation from 0-7 (see constants section)

; accumulator:
; dest is accumulator, src is immediate after opcode, no modrm byte
; direction:
; if not set, src may be r/m, otherwise dest is r/m
; wide:
; if set, then operation is wide (r/m32), otherwise operation is 8 bit

```

```

;==
;   FuCkYoTraSh
;
; executable junk code generator
;   by ctrlaltdestroy_
;==

```

format MS COFF

include 'win32a.inc'

public gen_junk as '_gen_junk@8'

;== Constants ==

```

OP_ADD  = 0
OP_OR   = 1
OP_ADC  = 2
OP_SBB  = 3
OP_AND  = 4
OP_SUB  = 5
OP_XOR  = 6
OP_CMP  = 7

```

OP_MOV = \$b0

```

F_WIDE = $01
F_DIR  = $02
F_ACCUM = $04

```

;==

;== Code ==

section '.text' code readable executable

;==

```

; int gen_junk(__out_deref void* lpOut, size_t cbOut);
gen_junk:
    push ebx esi edi ebp

```

```
mov edi, [esp+4*4+4*1] ; lpOut
mov ebp, [esp+4*4+4*2] ; cbOut
mov esi, ebp
```

```
.gen_instr:
call gen_logical
sub ebp, ecx
cmp ebp, $5
jb .done
rdtsc
test eax, $4
jz .gen_instr
.gen_movinstr:
call gen_mov
sub ebp, ecx
cmp ebp, $5
jb .done
jmp .gen_instr
```

```
sub esi, ebp
mov eax, esi ; num bytes generated
.done: ; always (cbOut - n(<5))
pop ebp edi esi ebx
ret 4*2
```

```
; generates a mov r8/32, imm8/32
;
```

```
; in: edi - buffer to output to
; out: ecx - size of opcode
```

```
gen_mov:
push eax edx
```

```
stdcall rand,eax
pop edx
```

```
test edx, edx ; flag for 8 or 32
mov al, OP_MOV
mov ecx, 2 ; sizeof.mov r8, imm8
js .mov_8
.mov_32:
or al, $08
mov ecx, 5 ; sizeof.mov r32, imm32
.mov_8:
mov ah, dl
and ah, $07 ; get register code
add al, ah
stosb
```

```
.mov_regimm:
stdcall rand,eax
pop eax
test edx, edx
js .mov_8_imm
```

```
.mov_32_imm:
stosd
jmp .done
.mov_8_imm:
stosb
```

```
.done:
pop edx eax
ret
```

; generates a logical math operation

```
;
;
; in: edi - buffer to output to
; out: ecx - size of opcode
gen_logical:
```

```
push eax ebx edx
```

```
xor eax, eax
```

```
.invalid_op:
stdcall rand,eax
pop edx
and edx, $ffff0707
```

```
mov al, dl ; operation octet
shl al, 3
or al, dh ; settings octet
mov ah, al
and ah, $07
cmp ah, 5 ; if opcode > 5, then it's a push/pop segreg
ja .invalid_op ; or other non-logical instruction
```

```
stosb ; we've constructed the opcode
```

```
test al, F_ACCUM
jnz .accum_regimm
```

```
.logical_regreg:
stdcall rand,eax
pop edx ; generate new magic for regs
and edx, $07070707
```

```
shl eax, 8 ; ah contains opcode
mov al, dl ; set modrm.reg field
shl al, 3 ; shift into place
```

```
ror edx, 16 ; get next field
or al, dl ; set modrm.rm field
```

```
or al, $c0 ; 0x3 modrm.mod == reg-reg
```

```
stosb ; output modrm byte
mov ecx, 2
```

```
jmp .done
```

```
.accum_regimm:  
stdcall rand,eax  
pop edx ; generate a new imm
```

```
test al, F_WIDE  
jnz .accum_imm32
```

```
.accum_imm8:  
mov al, dl  
stosb  
mov ecx, 2  
jmp .done
```

```
.accum_imm32:  
mov eax, edx  
stosd  
mov ecx, 5  
jmp .done
```

```
.done:  
pop edx ebx eax  
ret
```

```
; creds to Knuth, Art of Computer Programming
```

```
rand: ; uint32_t __911call rand(void* dummy)
```

```
push eax edx  
; wat  
rdtsc  
not eax  
xor [rand_seed], eax  
; wat  
mov eax, [rand_seed]  
mov edx, $0019660D  
mul edx  
add eax, $3C6EF35F  
mov dword [rand_seed], eax  
mov [esp+4*2+4*1], eax
```

```
pop edx eax  
ret
```

```
==  
section '.data' data readable writeable  
==  
rand_seed dd $ff
```

_Spy - October 16, 2013, 4:25 pm

Nice work buddy pretty interesting :) .