

fluff.c – a kernel/driver/app scanning utility

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-06/1929.html>

cutaway_at_bellsouth.net

Date: 06/08/05

To: <linux-kernel@vger.kernel.org>
Date: Wed, 8 Jun 2005 11:55:27 -0400

Scans an (uncompressed) kernel image (or any other app) trying to estimate how much space is being wasted from a compiler injecting gratuitous alignments on message strings. It also looks for potential x86 register operation savings.

It displays a bit of context and offset in image file being scanned when it thinks it has found something.

This is undoubtedly chock full of bugs (it was whipped together in about an hour) and could certainly be refined further in its discrimination of what to hit on or not, but even in its current state it does appear to emit some useful and revealing rather information. ex. typical x86 kernels are wasting well over 20K of space on gratuitous zero byte alignment padding on message strings that have no performance bearing what so ever.

This is NOT GPL'd – I release this as 100% public domain code. Anyone is 100% free to do ANYTHING with it as they please. If it causes cancer in lab rats, makes your hair fall out, or greases your hard drive(s), I'd be interested in hearing about it, but I can't be held responsible. Caveat Emptor.

Tony

----- CUT HERE -----

/*

Brute force scan of binary files to look for gratuitous alignment of text strings that are porking up the binary image.

Also scans for potential register op savings

Usage: fluff <filename>

Displays total & context of suspect gaseous bytes encountered

*/

#include <stdio.h>

#include <stdlib.h>

Linux–Kernel: fluff.c – a kernel/driver/app scanning utility

```
#include <string.h>
#include <limits.h>
#include <ctype.h>

/*#define SHOW_LOW_GRADE_OPPORTUNITY*/

#define MAXFILELEN 10000000 /* 10M – enough for any Linux kernel */

static char buf[MAXFILELEN];

size_t flen;

unsigned long gas = 0L;
unsigned long reggas = 0L;

static char *ScanForZero(char *p)
{
    while (*p != 0) p++;
    return p;
}

#define CHECKCHARS 4

static int StrCheck(char *p)
{
    {
        int rc = 1;
        char cbuf[CHECKCHARS];
        int i;

        for (i=0; i<CHECKCHARS; i++)
            cbuf[i] = *(p+i);

        for (i=0; i<CHECKCHARS; i++)
        {
            if ((!isascii(cbuf[i])) || (!isprint(cbuf[i])))
            {
                rc = 0;
                break;
            }
        }
        return rc;
    }
}

static void ZapNL(char *p)
{
    {
        while (*p)
        {
            if (*p == '\n')
                *p = ' ';
            p++;
        }
    }
}
```

```

}

static char *CheckPadHere(char *p)
{
    unsigned long ZeroCount = 0;
    char scratch[100];

    p = ScanForZero(p);

    while (*p++ == 0)
        ZeroCount++;

    if (StrCheck(p))
    {
        if ((ZeroCount > 2) && (ZeroCount < 32))
        {
            strncpy(scratch, p-1, 30);
            scratch[30] = 0;
            ZapNL(scratch); /* For display readability */
            printf("Near [%08X] ['%30s...'] preceded by %ld zeros\n",
                p-buf, scratch, ZeroCount);
            gas += ZeroCount;
        }
    }
    return p;
}

static void ScanPadding(void)
{
    char *p = buf;

    buf[MAXFILELEN-1] = 0xff;
    flen--;

    while (p < &buf[MAXFILELEN-1])
        p = CheckPadHere(p);
}

static int RegOpportunity(unsigned char OpCode, unsigned int imm32, char *p)
{
    int opp = 0;
    char *reg;

    switch (imm32)
    {
        /* Might convert to XOR reg32,reg32 */
        case 0 :
            opp = 3;
            break;
        /* Might convert to XOR/INC reg32,reg32 */
        case 1 :

```

Linux–Kernel: fluff.c – a kernel/driver/app scanning utility

```
    opp = 2;
    break;
/* Might convert to OR reg32,-1 */
case 0xffffffff :
    opp = 2;
    break;
#ifdef SHOW_LOW_GRADE_OPPORTUNITY
/* Might convert to XOR reg32,reg32, MOV reg,imm8, saves 1 byte */
default :
    if ( (OpCode == 0xb8) || (OpCode == 0xbb) ||
        (OpCode == 0xb9) || (OpCode == 0xba))
        opp = 1;
    break;
#endif
}

switch (OpCode)
{
    case 0xb8 : reg = "EAX"; break;
    case 0xbb : reg = "EBX"; break;
    case 0xb9 : reg = "ECX"; break;
    case 0xba : reg = "EDX"; break;
    case 0xbd : reg = "EBP"; break;
    case 0xbe : reg = "ESI"; break;
    case 0xbf : reg = "EDI"; break;
    default:
        printf("Internal error, bad REG opcode\n");
        exit(-1);
}
if (opp)
    printf("Near [%08X] Suspected MOV %s,%08x\n", p-buf, reg,imm32);
return opp;
}

static char *CheckRegHere(char *p)
{
    unsigned char OpCode = (unsigned char)*p;
    unsigned int imm32;

    switch (OpCode)
    {
        case 0xb8 : /* MOV EAX,imm32 */
        case 0xbb : /* MOV EBX,imm32 */
        case 0xb9 : /* MOV ECX,imm32 */
        case 0xba : /* MOV EDX,imm32 */
        case 0xbd : /* MOV EBP,imm32 */
        case 0xbe : /* MOV ESI,imm32 */
        case 0xbf : /* MOV EDI,imm32 */
            p++;
            imm32 = *((unsigned int *)p);
    }
}
```

Linux–Kernel: fluff.c – a kernel/driver/app scanning utility

```
    if ((imm32 == 0) || (imm32 == 0xffffffff) ||
        ((imm32 >= 1) && (imm32 <= 255)))
    {
        reggas += RegOpportunity(OpCode, imm32, p-1);
        p += 4;
    }
    break;
default:
    p++;
    break;
}
return p;
}

static void ScanRegOp(void)
{
    char *p = buf;

    buf[MAXFILELEN-1] = 0xff;
    flen--;

    while (p < &buf[MAXFILELEN-1])
    {
        p = CheckRegHere(p);
    }
}

static void usage(void)
{
    printf("usage: fluff <filename>\n");
    exit(-1);
}

int main(int argc, char *argv[])
{
    FILE *f = fopen(argv[1], "rb");
    int rc = 0;

    if (argc != 2)
        usage();

    f = fopen(argv[1], "rb");
    if (f)
    {
        flen = fread(buf, 1, MAXFILELEN, f);
        fclose(f);
        if (flen)
        {
            printf("%ld bytes read from %s\n", (long)flen, argv[1]);
            printf("\n----- SCANNING FOR PAD
ZEROS ----- \n\n");
        }
    }
}
```

Linux-Kernel: fluff.c – a kernel/driver/app scanning utility

```
    ScanPadding();
    printf("\n----- SCANNING FOR REG
S -----\n\n");
    ScanRegOp();

    printf("\n\nTOTAL SUSPECTED ZERO PADDING = %ld bytes\n", gas);
    printf("POTENTIAL REGISTER OPS SAVINGS = %ld bytes\n", reggas);
}
else
{
    printf("Opened %s, but read nothing!\n", argv[1]);
    rc = -1;
}
}
else
{
    printf("Can't open %s\n", argv[1]);
    rc = -1;
}
return rc;
}
```

–

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>