~~2007年 11月 3日 (土)~~
~~Real UNIX MAGAZINE Day~~

2010年 5月 8日 (土)
第四回 カーネル／YUREX 探検隊

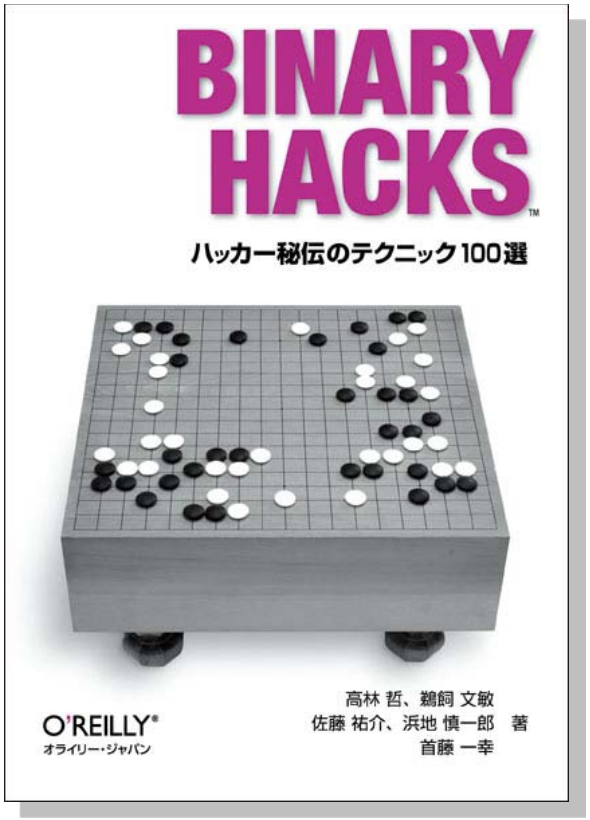
いまどきの Binary Hacks 前編

首藤 一幸

Binary Hacks

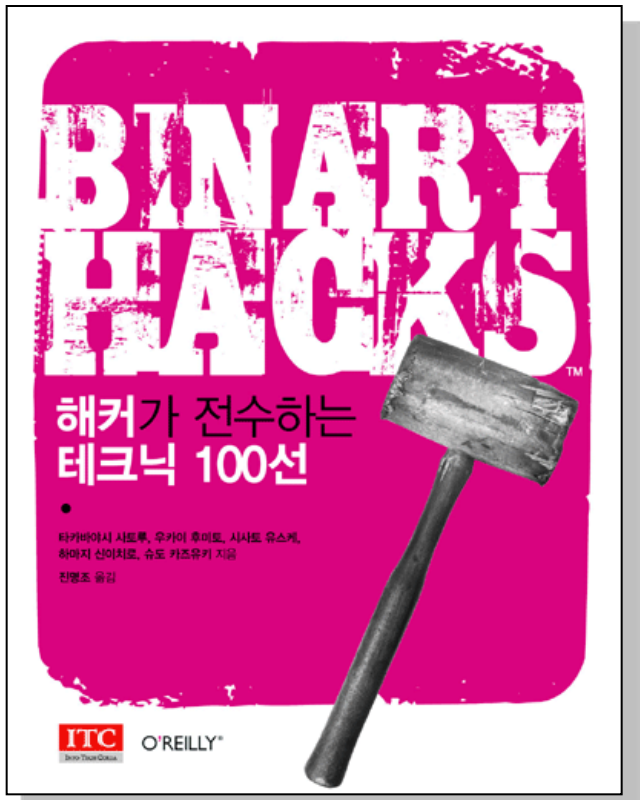
- 大好評！

日本語版



2006年 11月

韓国語訳



2007年 7月

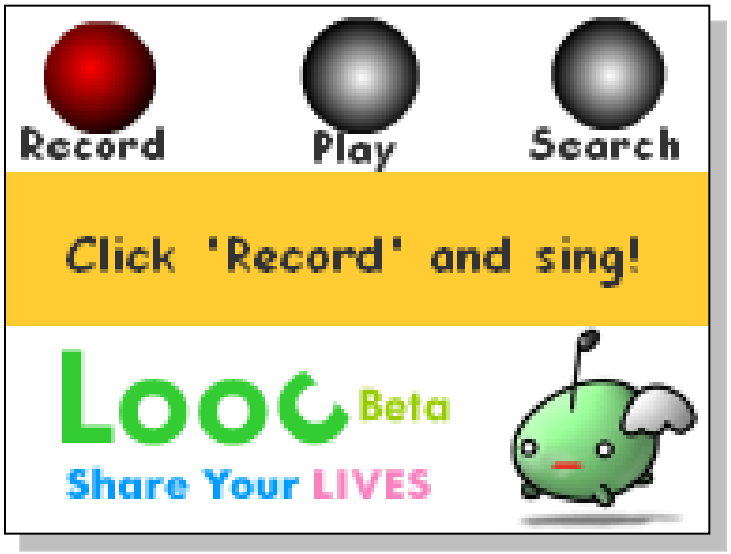
著者 (韓国語):
타카바야시 사토루
우카이 후미토시
사토 유스케
하마지 신이치로
슈도 카즈유키

日本語訳 (by OCN)
高林サトル
右カイ・フミトシ
佐藤ユウスケ
カバ紙シンイチロウ
首藤カズユキ

日本語訳 (by Excite)
他カバー夜店Satoru
鵜飼フミトシ
佐藤Yusuke
下馬だShinichiで
シューもKazuyuki

ウェブ系：うたごえ検索 ウタゴエ(株)

- はなうた歌って曲を検索。タタタ～♪
- PC 版のユーザ側は Flash で作ってある。



PC 版 うたごえ検索

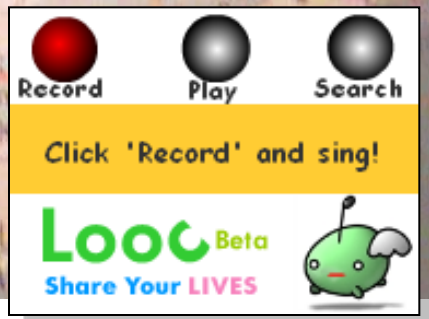
録音した音声を
サーバ側で解析

UTA GOE
UNIVERSAL GRID

ウェブ系を支えるバイナリ技術

オールドタイプの皆様に
癒しを御提供

Flash の音声フォーマット



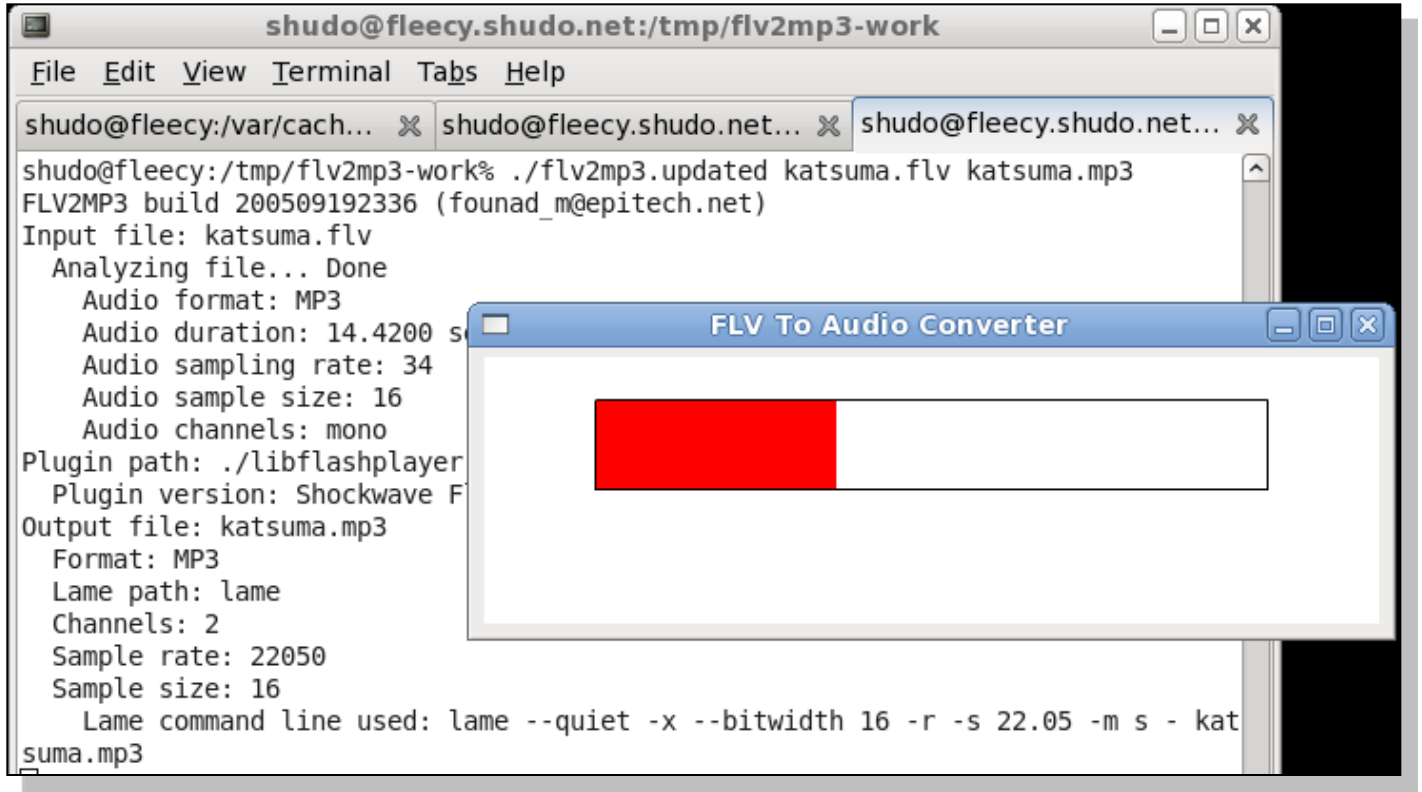
- うたごえ検索: ブラウザ側の Flash Player で録音した音声を、サーバ側で解析。
  - 録音結果ファイル: ○○.flv (Flash Video)

- Nerrymoser という CODEC でエンコードされている。解析のために要デコード。
  - ffmpeg も未サポート。
  - 数千ドル (!!!) のコンバータを買うのか？？？

- Linux 用の flv2mp3 が見つかった！
  これでデコードできるか！？

flv2mp3

- Debian GNU/Linux for x86 用のバイナリ。ソースなし。
- アクロバティックな作り
  - Adobe が配布している Flash Player の共有ライブラリ libflashplayer.so を使ってデコード。
    音声デバイスへの出力を横取りする。



flv2mp3

- Fedora Core 6 では落ちる！
  - 僕らは Red Hat 系列, Fedora に慣れてるので、Fedora で使いたい！

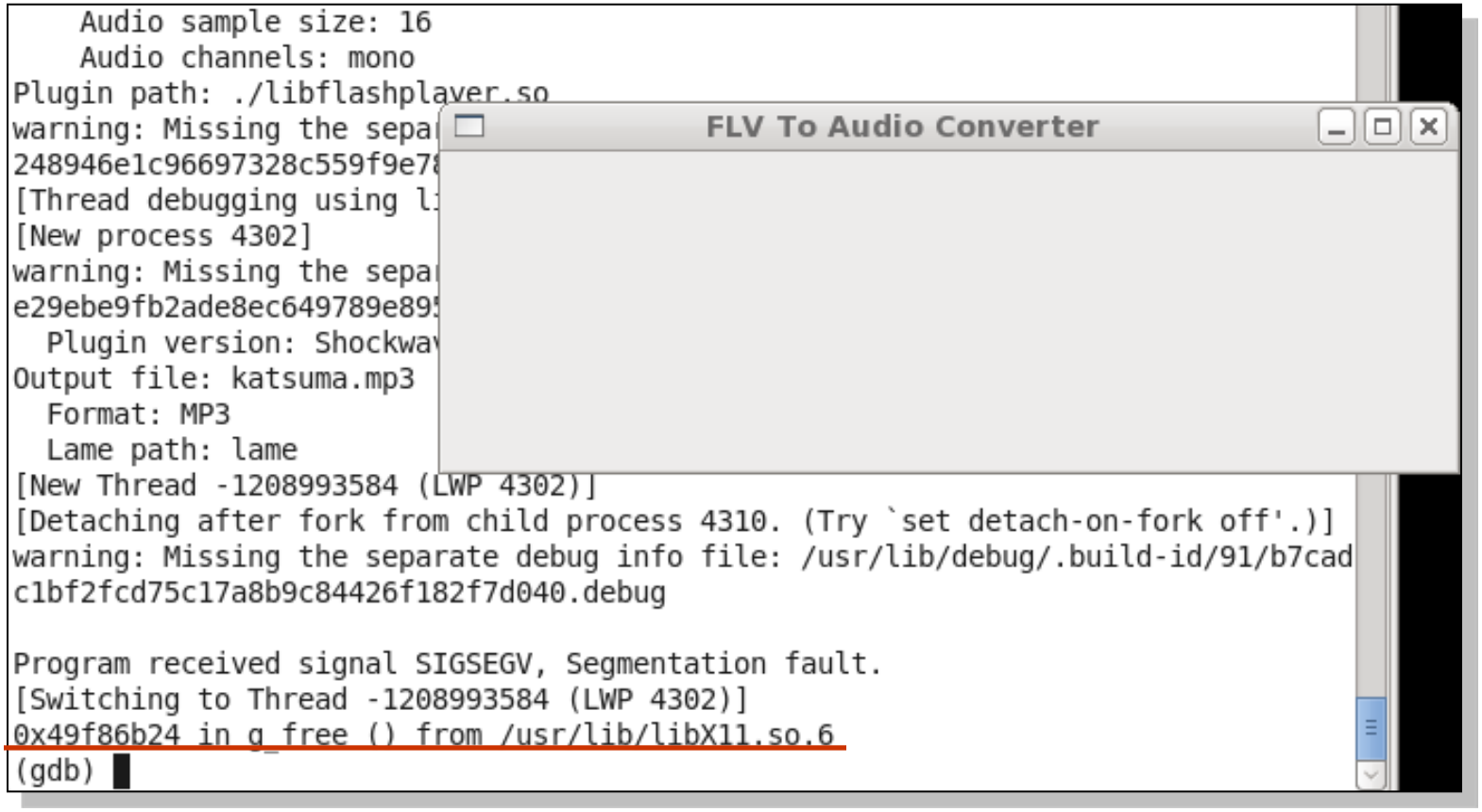
このへん

```
shudo@fleecy:/tmp/flv2mp3-work% ./flv2mp3 katsuma.flv katsuma.mp3
FLV2MP3 build 200509192336 (founad_m@epitech.net)
Input file: katsuma.flv
  Analyzing file... Done
    Audio format: MP3
    Audio duration: 14.4200 secs
    Audio sampling rate: 34
    Audio sample size: 16
    Audio channels: mono
Plugin path: ./libflashplayer.so
  Plugin version: Shockwave Flash 7.0 r69
Output file: katsuma.mp3
  Format: MP3
  Lame path: lame
Segmentation fault
shudo@fleecy:/tmp/flv2mp3-work%
```

- バイナリアン や old type にとって、「Segmentation Fault は友達！」

なにはともあれ GDB

- ん？ libX11.so の中で落ちてる？
  - libX11: X Window System の中心的なライブラリ



このへん

逆アセンブル

- % objdump –d /usr/lib/libX11.so.6

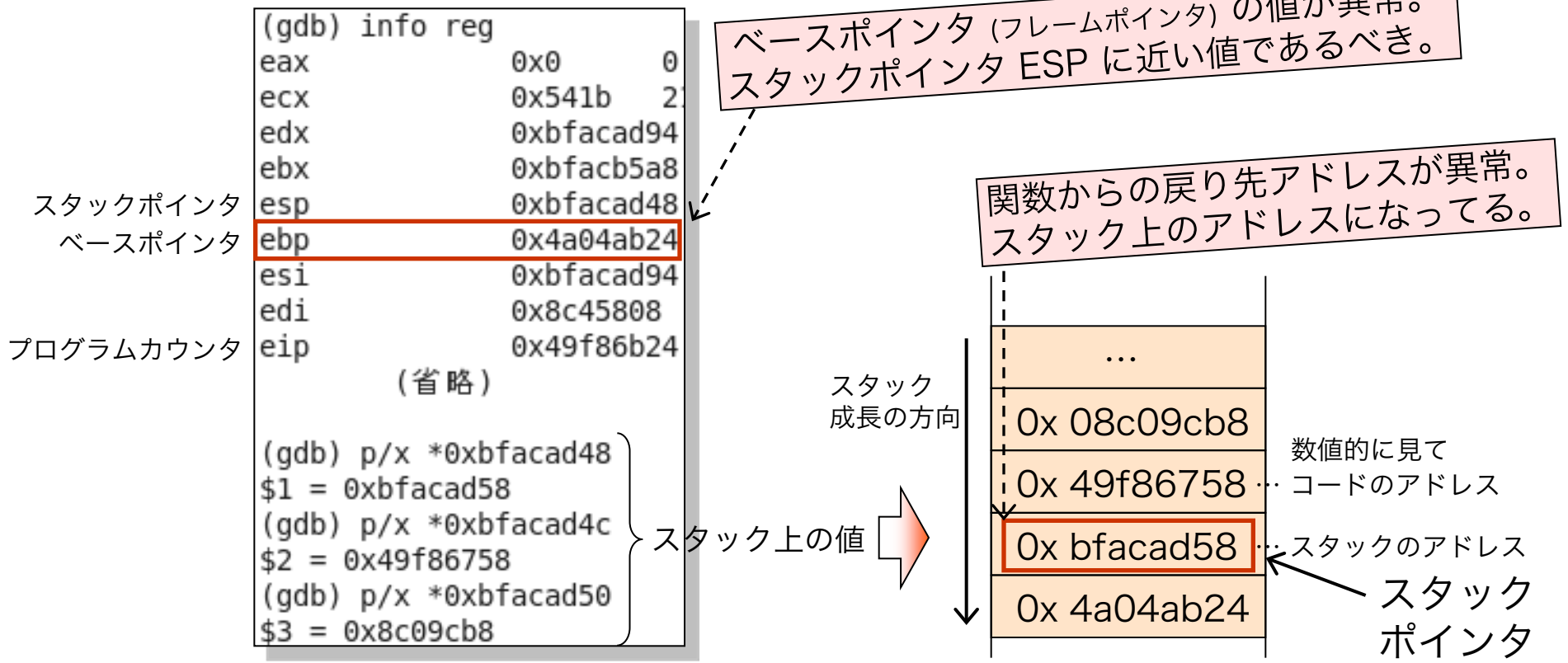
1つの関数

```
49f86af0:       55                              push   %ebp
49f86af1:       89 e5                           mov    %esp,%ebp
49f86af3:       53                              push   %ebx
49f86af4:       83 ec 14                        sub    $0x14,%esp
49f86af7:       8b 45 0c                        mov    0xc(%ebp),%eax
49f86afa:       e8 58 a5 fd ff                  call   49f61057 <geteuid@plt+0xe3>
49f86aff:       81 c3 25 40 0c 00               add    $0xc4025,%ebx
49f86b05:       c7 44 24 04 1b 54 00    movl   $0x541b,0x4(%esp)
49f86b0c:       00
49f86b0d:       89 44 24 08                     mov    %eax,0x8(%esp)
49f86b11:       8b 45 08                        mov    0x8(%ebp),%eax
49f86b14:       8b 40 10                        mov    0x10(%eax),%eax
49f86b17:       89 04 24                        mov    %eax,(%esp)
49f86b1a:       e8 95 8c fd ff                  call   49f5f7b4 <ioctl@plt>
49f86b1f:       83 c4 14                        add    $0x14,%esp
49f86b22:       5b                              pop    %ebx
49f86b23:       5d                              pop    %ebp
49f86b24:       c3                              ret
```

ret 命令で落ちてる

ret 命令で落ちるということは

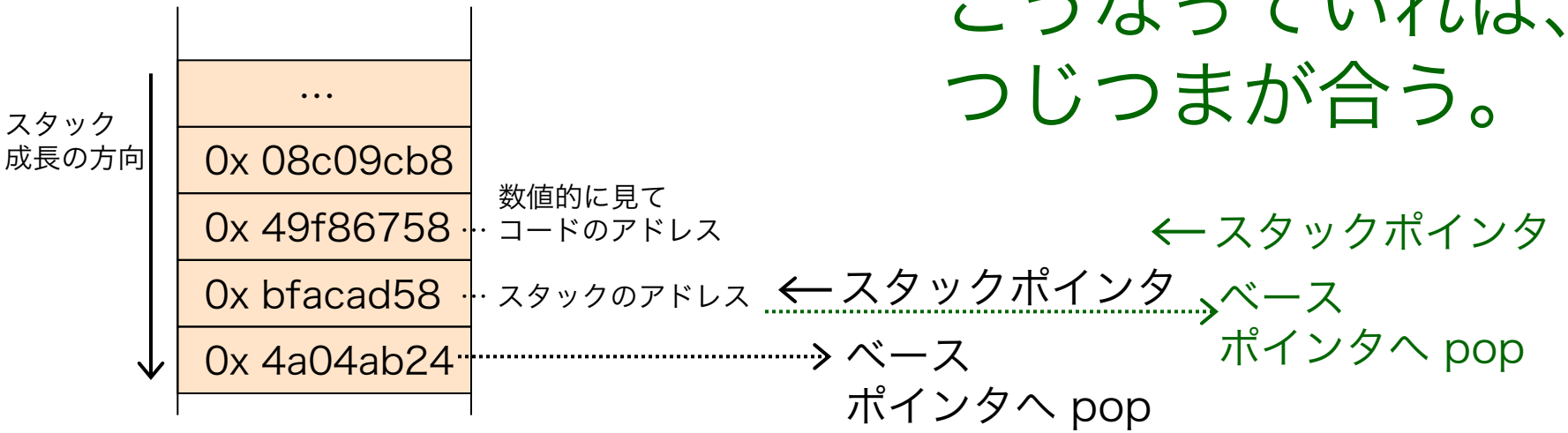
- 戻り先のアドレスが不正になっている。
  - 戻り先アドレスは、スタックから pop する。
- GDB から、スタックポインタの値と、スタック上の値を調査。



スタックポインタのズレ？

```
49f86b22:       5b                              pop    %ebx
49f86b23:       5d                              pop    %ebp
49f86b24:       c3                              ret
```

ここで落ちてる



こうなっていれば、
つじつまが合う。

- まままさか、スタックポインタがズレてる！？

スタックポインタをズラしてみる

```
49f86b1a:       e8 95 8c fd ff          call   49f5f7b4 <ioctl@plt>
49f86b1f:       83 c4 14                add    $0x14,%esp
49f86b22:       5b                      pop    %ebx
49f86b23:       5d                      pop    %ebp
49f86b24:       c3                      ret <- ここで落ちてる
```

0x18 に変更

- ちょうど、直前でスタックポインタの値を調整している。
- 調整幅を変更
  - 1要素分多めに: 0x14 → 0x18

libX11.so をイジる

- 変更したい個所の、ファイル中の位置を調べる。
  - % objdump –hd /usr/lib/libX11.so.6.2.0

```
Disassembly of section .text:

49f60f90 <XAllocColorCells-0xd0>:
49f60f90:       55                      push   %ebp
        (省略)
49f86b1a:       e8 95 8c fd ff          call   49f5f7b4 <ioctl@plt>
49f86b1f:       83 c4 14                add    $0x14,%esp
49f86b22:       5b                      pop    %ebx
49f86b23:       5d                      pop    %ebp
49f86b24:       c3                      ret
```

変更したい個所

0x49f86b21

.text セクション中の位置:
0x49f86b21 – 0x49f60f90 =
0x25b91

```
/usr/lib/libX11.so.6:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
        (省略)
  9 .text         0008fb04  49f60f90  49f60f90  00014f90  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
```

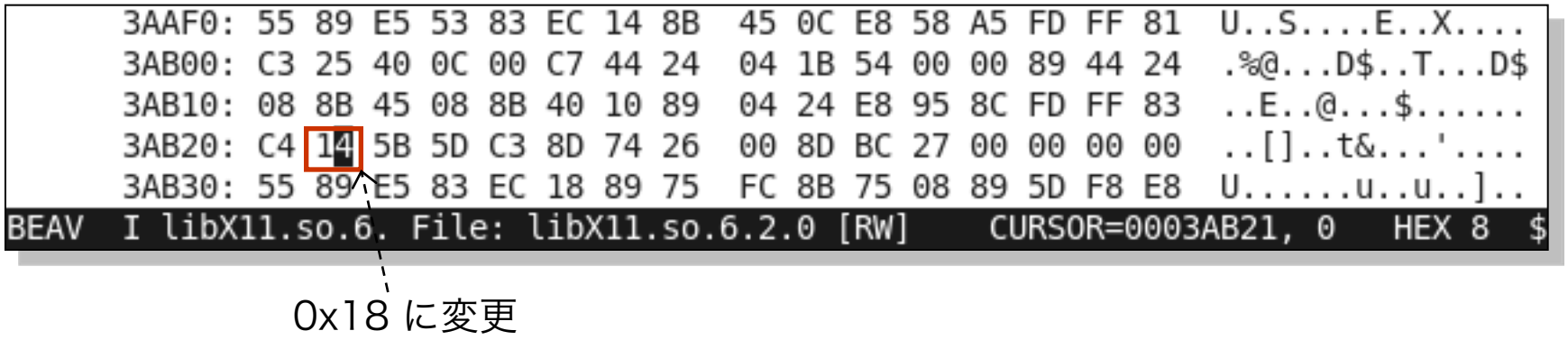
.text セクション自体の
ファイル中の位置: 0x14f90

変更したい個所のファイル中の位置:

0x14f90 + 0x25b91 = 0x3ab21

libX11.so を ×××

- お好みのバイナリエディタで。
  - Emacs の hexl モードとか beav とか。

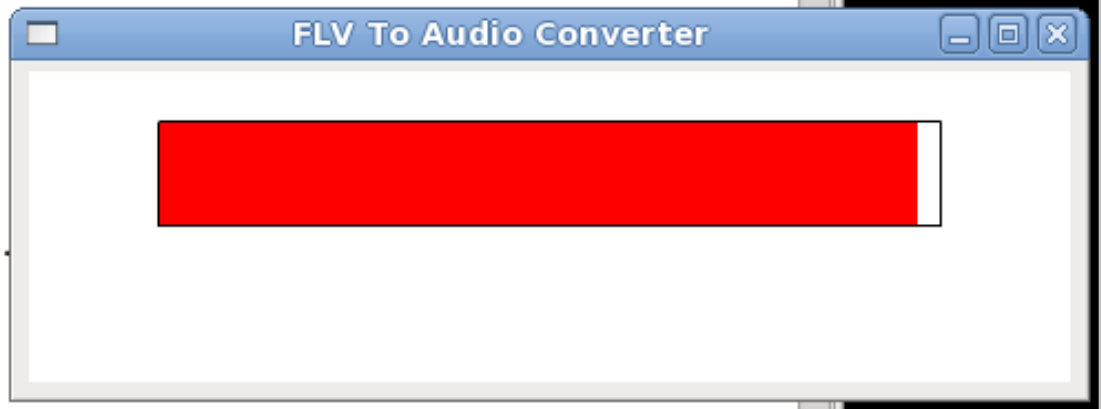


- イジった libX11 を使う。
  - /usr/lib/libX11.so… を置き換え → 運が悪いと X が起動しない。
  - 環境変数 LD_PRELOAD を指定して、変更版 libX11 を preload。
    - env LD_PRELOAD=…/libX11.so.6 flv2mp3 …

flv2mp3 が動いた！

```
shudo@fleecy:/tmp/flv2mp3-work% env LD_PRELOAD=./libX11.so.6.2.0 ./flv2mp3 katsu
ma.flv katsuma.mp3
FLV2MP3 build 200509192336 (founad_m@epitech.net)
Input file: katsuma.flv
  Analyzing file... Done
    Audio format: MP3
    Audio duration: 14.4200 secs
    Audio sampling rate: 34
    Audio sample size: 16
    Audio channels: mono
Plugin path: ./libflashplayer.so
  Plugin version: Shockwave Flash 7.
Output file: katsuma.mp3
  Format: MP3
  Lame path: lame
  Channels: 2
  Sample rate: 22050
  Sample size: 16
    Lame command line used: lame --quiet -x --bitwidth 16 -r -s 22.05 -m s - kat
suma.mp3
```

イジった libX11 を使用



- めでたしめでたし！

Beijing
Lanzhou
Tianjin
Huang He
NORTH
Pyongyang
KOREA
Seoul
SOUTH
KOREA
JAPAN
Honshu
Tokyo

ん？

libX11.so を ××× って、どうよ？

- きちんとコンパイルされて、整合がとれているはずの libX11.so を壊している。
  - 通常の使用では、かえって問題が起きるはず。

- そもそも、
  なんでスタックポインタの値がズレるの？
  - ありえない。

Beijing
Lanzhou
Tianjin
Huang He
NORTH
Pyongyang
KOREA
Seoul
SOUTH
KOREA
JAPAN
Honshu
Tokyo

後編 ～ 真の解決へ ～ に続く

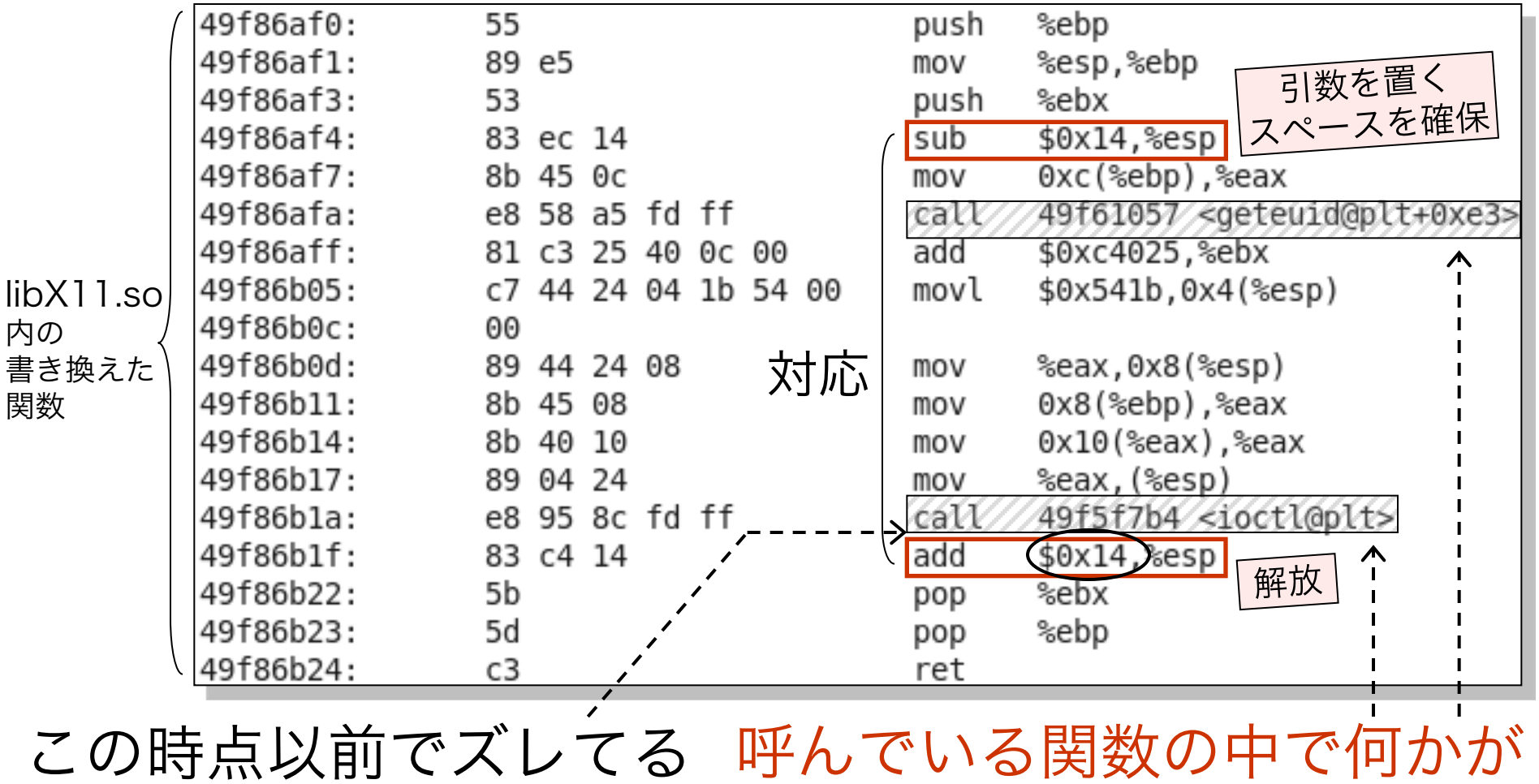
2010年 5月 8日 (土)
第四回 カーネル／YUREX 探検隊

いまどきの Binary Hacks 後編

首藤 一幸

なぜスタックポインタがズレるのか？

- この関数自体には、問題なさげ。



この時点以前でズレてる　呼んでいる関数の中で何かが

スタックポインタの変化を調査

- ブレークポイントをところどころに設定。

```
49f86b1a:       e8 95 8c fd ff          call   49f5f7b4 <ioctl@plt>
49f86b1f:       83 c4 14                add    $0x14,%esp
```

PLT (Procedure Linkage Table) 中の ioctl 関数用コードのアドレス

```
(gdb) b *0x49f86b1a
Breakpoint 1 at 0x49f86b1a
(gdb) b *0x49f86b1f
Breakpoint 2 at 0x49f86b1f
(gdb) run katsuma.flv katsuma.mp3
        (省略)
Breakpoint 1, 0x49f86b1a in g_free () from /usr/lib/libXi.so.6
(gdb) info reg
eax            0x4      4
ecx            0x4a04b6c0       1241822912
edx            0x8a66cb8        145124536
ebx            0x4a04ab24       1241819940
esp            0xbfc744d0       0xbfc744d0
        (省略)
Breakpoint 2, 0x49f86b1f in g_free () from /usr/lib/libXi.so.6
(gdb) info reg
eax            0x0      0
ecx            0x541b   21531
edx            0xbfc74534       -1077459660
ebx            0xbfc744c8       -1077459768
esp            0xbfc744cc       0xbfc744cc
```

ioctl 呼び出し直前の状態

ioctl 呼び出し直後の状態

4 減ってる

あってはいけないこと

call 命令の前後でスタックポインタの値が変化してる！

ioctl 関数の中で何が起きるのか

- とりあえず、GDB から逆アセンブル。

```
(gdb) disas ioctl
Dump of assembler code for function ioctl:
0x0804dfd0 <ioctl+0>:   sub    $0x1c,%esp
0x0804dfd3 <ioctl+3>:   mov    0x20(%esp),%eax
0x0804dfd7 <ioctl+7>:   mov    %ebx,0x18(%esp)
```

メモリ上

- よくよく眺めるために、
  コマンドラインから libc を逆アセンブル。
  - 注: ioctl(2) はシステムコールだが、glibc は関数として提供する。

```
% objdump -d /lib/libc.so.6 | less
        (省略)
009cb130 <ioctl>:
  9cb130:       53                      push   %ebx
  9cb131:       8b 54 24 10             mov    0x10(%esp),%edx
  9cb135:       8b 4c 24 0c             mov    0xc(%esp),%ecx
```

ライブラリ内

ん？

flv2mp3 のプロセス空間 (メモリ) と libc (ライブラリ) で、
ioctl 関数の内容が**食い違っている！？**

では、flv2mp3 が使う ioctl 関数はどこから来るのか？

- プロセス空間内で、ioctl は 0x804dfd0 に置かれている。

flv2mp3 のプロセスIDを調べ

```
% ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
        (省略)
shudo     4733  0.0  3.0  18292 15564 pts/2    S+   Oct07   0:01 gdb ./flv2mp3
shudo     4744  0.0  1.2  19492  6172 pts/2    T    Oct07   0:00 /tmp/flv2mp3-wo

% less /proc/4744/maps
        (省略)
00900000-00a52000 r-xp 00000000 08:01 2031625    /lib/libc-2.6.90.so
00a52000-00a54000 r-xp 00152000 08:01 2031625    /lib/libc-2.6.90.so
00a54000-00a55000 rwxp 00154000 08:01 2031625    /lib/libc-2.6.90.so
        (省略)
08048000-08050000 r-xp 00000000 08:01 3866730    /tmp/flv2mp3-work/flv2mp3
08050000-08051000 rwxp 00007000 08:01 3866730    /tmp/flv2mp3-work/flv2mp3
```

0x804dfd0 番地の内容がどのファイル由来かを調べる

- libc ではなく、
  flv2mp3 自身の中に ioctl 関数がある？

flv2mp3 内の ioctl 関数

- flv2mp3 自身が ioctl を提供していることが判明。

```
% objdump -d flv2mp3 | less
        (省略)
0804dfd0 <ioctl>:
 804dfd0:       83 ec 1c                sub    $0x1c,%esp
 804dfd3:       8b 44 24 20             mov    0x20(%esp),%eax
 804dfd7:       89 5c 24 18             mov    %ebx,0x18(%esp)
```

メモリ上のものと
同じ内容

- Adobe 提供の libflashplayer.so 向けに、特殊な ioctl 関数を提供しているのだろう。
  - たしかに、ioctl の引数によって色々と条件分岐している。
  - 特定の引数に対して、特別な処理をしているのだろう。

flv2mp3 内の ioctl 関数

- flv2mp3 内の ioctl 関数は何をしているか？スタックポインタの値はなぜズレるのか？
- 処理を追う。

逆アセンブル

```
% objdump -d flv2mp3 | less
        (省略)
0804dfd0 <ioctl>:
        (省略)
 804e0d1:       b8 48 a0 04 08          mov    $0x804a048,%eax
 804e0d6:       89 da                   mov    %ebx,%edx
 804e0d8:       89 d4                   mov    %edx,%esp
 804e0da:       59                      pop    %ecx
 804e0db:       5a                      pop    %edx
 804e0dc:       6a 36                   push   $0x36
 804e0de:       52                      push   %edx
 804e0df:       ff e0                   jmp    *%eax
```

- 末尾で、EAX レジスタに格納されているアドレスにジャンプしている。どこに飛んでいる？

flv2mp3 内の ioctl 関数

- 末尾のジャンプ命令で、どこに飛ぶのか？

```
(gdb) b *0x804e0df
Breakpoint 3 at 0x804e0df: file intercept.c, line 96.
        (省略)
Breakpoint 3, 0x0804e0df in ioctl (fd=-1074706536, re
    at intercept.c:96
96      intercept.c: No such file or directory.
        in intercept.c
(gdb) info reg
eax            0x804a048        134520904
        (省略)
(gdb) disas 0x804a048

Dump of assembler code for function syscall@plt:
0x0804a048 <syscall@plt+0>:     jmp    *0x80502e0
0x0804a04e <syscall@plt+6>:     push   $0x178
0x0804a053 <syscall@plt+11>:    jmp    0x8049d48
```

ジャンプ命令にブレークポイントを設定

ジャンプ先である EAX レジスタの中身を調査

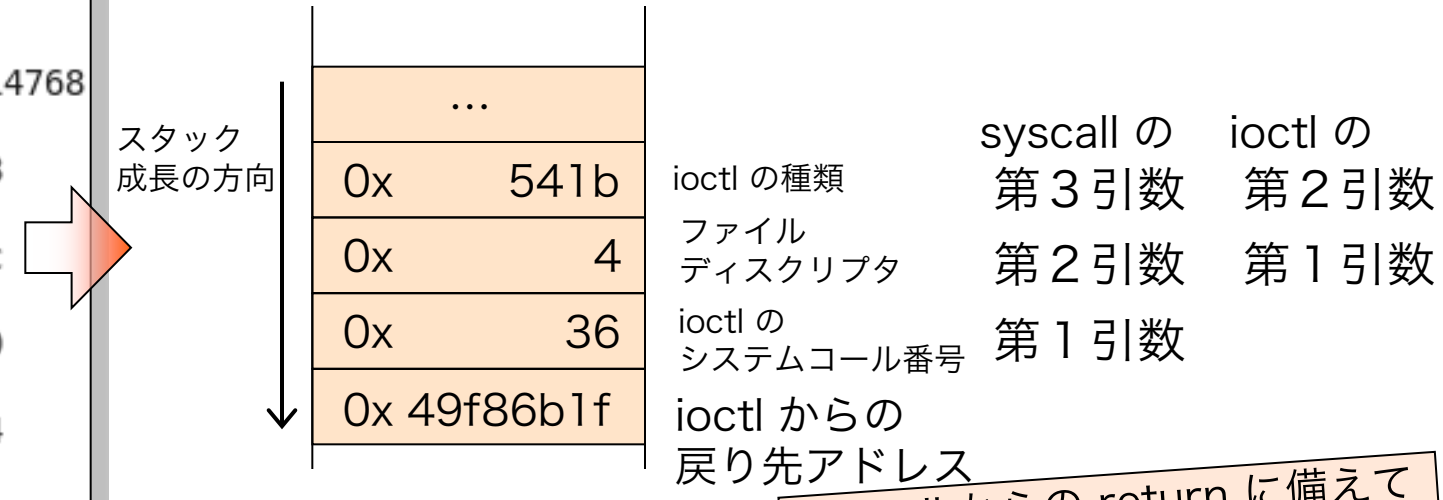
ジャンプ先を逆アセンブル

- どうも syscall 関数に**ジャンプ**するらしい。
  - 呼び出し (call) ではなくてジャンプ (jmp)。
  - つまり、syscall からの return を
    この ioctl からの return ということにしてしまう。

flv2mp3 内の ioctl 関数

- ioctl 関数から syscall 関数に直接ジャンプ。
  - syscall は、任意のシステムコールを呼び出す関数:
    `syscall(システムコール番号, システムコールの引数, …)`
- syscall へのジャンプ直前のスタックを調べる:

```
(gdb) info reg
        (省略)
esp            0xbff14768
        (省略)
(gdb) p/x *0xbff14768
$1 = 0x49f86b1f
(gdb) p/x *0xbff1476c
$2 = 0x36
(gdb) p/x *0xbff14770
$3 = 0x4
(gdb) p/x *0xbff14774
$4 = 0x541b
(gdb) p/x *0xbff14778
$5 = 0xbff147d4
```



syscall からの return に備えて
戻り先アドレスを積んである。

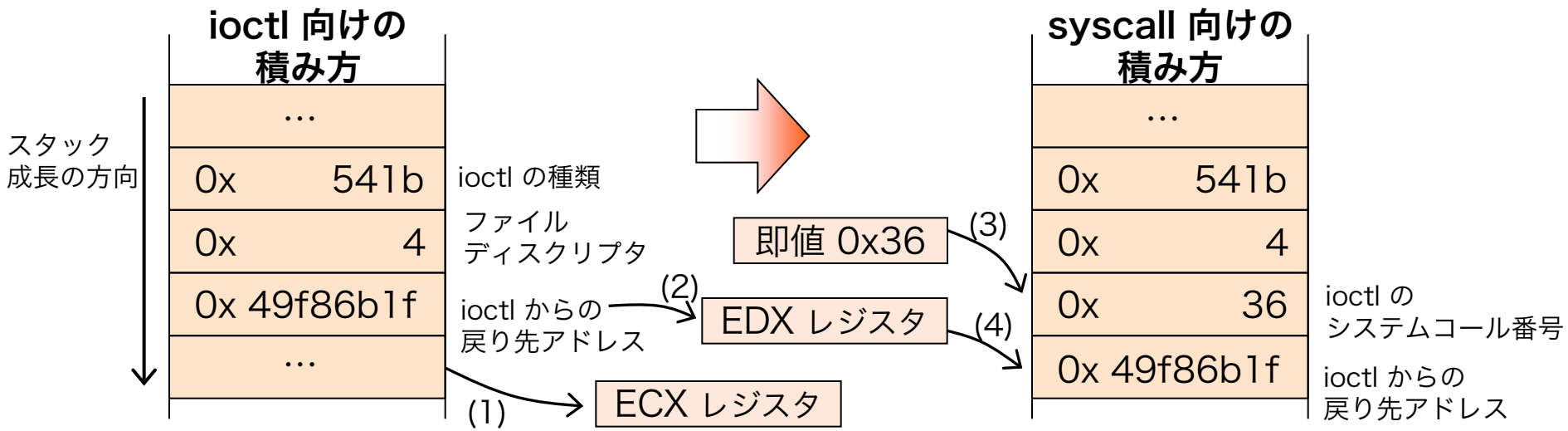
参考: ioctl の種類: `FIONREAD`

```
shudo@fleecy:/usr/include% fgrep -i 541b `find . -name '*.h'`
./asm/ioctls.h:#define FIONREAD 0x541B
```

flv2mp3 内の ioctl 関数

- ioctl 関数の末尾では、syscall 関数を呼ぶために、引数の積み直しをしている。

```
% objdump -d flv2mp3 | less
        (省略)
0804dfd0 <ioctl>:
        (省略)
 804e0da:       59                      (1)  pop    %ecx
 804e0db:       5a                      (2)  pop    %edx
 804e0dc:       6a 36                   (3)  push   $0x36
 804e0de:       52                      (4)  push   %edx
 804e0df:       ff e0                        jmp    *%eax
```



flv2mp3 内の ioctl 関数を呼び出した側にしてみると

- ioctl を呼んだつもりが syscall が実行される。
  - ここまでは、いい。

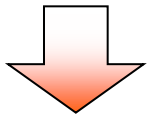
- 積んだおぼえのない引数が
  1 つ余分にスタックに積まれている！
  - ioctl のシステムコール番号 0x36

- 呼び出し側では、後始末のしようがない。
  - 注: 引数は、呼び出し側が積み、後始末する。
    後始末: スタックポインタの復旧

flv2mp3 内 ioctl 関数の修正

- そもそも、わざわざ引数を積み直して syscall を実行せずとも、libc 標準の ioctl を呼べばいいんじゃね？

- flv2mp3 内の「ioctl」から libc.so 内の「ioctl」を呼べるのか？
  - 同じ名前。
  - 呼べないかも。



- syscall 実行、という路線は変えない。

flv2mp3 内 ioctl 関数の修正

- syscall 実行後に、
  適切にスタックポインタを補正する。

(1) ioctl からの戻り先アドレスを ESI レジスタに保存する。
(2) ジャンプではなくて、syscall 関数を普通に呼び出す。
(3) syscall 向けに余分に積んだ引数の分、
スタックポインタを補正する。
(4) ioctl からの戻り先アドレスにジャンプする。

flv2mp3 の修正

- アセンブルして、書き込むべき機械語列を把握する。

```
% cat foo.s
popl %ecx
popl %esi
push $0x36
call *%eax
addl $4,%esp
jmp *%esi
% gcc -c foo.s
% objdump -d foo.o

foo.o:     file format elf32-i386

Disassembly of section .text:

00000000 <.text>:
   0:   59                      pop    %ecx
   1:   5e                      pop    %esi
   2:   6a 36                   push   $0x36
   4:   ff d0                   call   *%eax
   6:   83 c4 04                add    $0x4,%esp
   9:   ff e6                   jmp    *%esi
```

アセンブリコードを書き込んだファイルを用意

アセンブル

機械語列を覗く

11バイト

flv2mp3 の修正

- 変更したい個所の、ファイル中の位置を調べる。
  - % objdump –hd flv2mp3

```
Disassembly of section .text:

0804a520 <_start>:
 804a520:       31 ed                   xor    %ebp,%ebp
        (省略)
0804dfd0 <ioctl>:
 804dfd0:       83 ec 1c                sub    $0x1c,%esp
        (省略)
 804e0da:       59                      pop    %ecx
 804e0db:       5a                      pop    %edx
 804e0dc:       6a 36                   push   $0x36
 804e0de:       52                      push   %edx
 804e0df:       ff e0                   jmp    *%eax
 804e0e1:       b8 ff ff ff ff          mov    $0xffffffff,%eax
 804e0e6:       e9 57 ff ff ff          jmp    804e042 <ioctl+0x72>
 804e0eb:       90                      nop
 804e0ec:       8d 74 26 00             lea    0x0(%esi),%esi

0804e0f0 <open>:
```

ここから変更

22バイト書き込む余裕がある。

.text セクション中の位置:
0x804e0da – 0x804a520 =
0x3bba

```
flv2mp3:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
        (省略)
 11 .text         00003ed0  0804a520  0804a520  00002520  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
```

.text セクション自体の
ファイル中の位置: 0x2520

変更したい個所のファイル中の位置:
0x2520 + 0x3bba = 0x60da

flv2mp3 の修正

- お好みのバイナリエディタで。
  - 書き込む: 59 5e 6a 36 ff d0 83 c4 04 ff e6

```
        60D0: FF B8 48 A0 04 08 89 DA  89 D4 59 5A 6A 36 52 FF  ..H.......YZj6R
        60E0: E0 B8 FF FF FF FF E9 57  FF FF FF 90 8D 74 26 00  .......W.....t&
        60F0: FC 83 EC 1C B9 0B 00 00  00 89 5C 24 10 8B 5C 24  ..........\$..\
BEAV  I flv2mp3       File: flv2mp3       [RW]    CURSOR=000060DA, 1   HEX 8   0
```

- 残りの 11 バイト分は、0x90 (NOP) でも書き込んでおく。
  - これをやらないと、逆アセンブル時に困る:

```
804e0da:       59                      pop    %ecx
804e0db:       5e                      pop    %esi
804e0dc:       6a 36                   push   $0x36
804e0de:       ff d0                   call   *%eax
804e0e0:       83 c4 04                add    $0x4,%esp
804e0e3:       ff e6                   jmp    *%esi
804e0e5:       ff e9                   ljmp   *<internal disassembler error>
804e0e7:       57                      push   %edi
804e0e8:       ff                      (bad)
804e0e9:       ff                      (bad)
804e0ea:       ff 90 8d 74 26 00       call   *0x26748d(%eax)

0804e0f0 <open>:
```

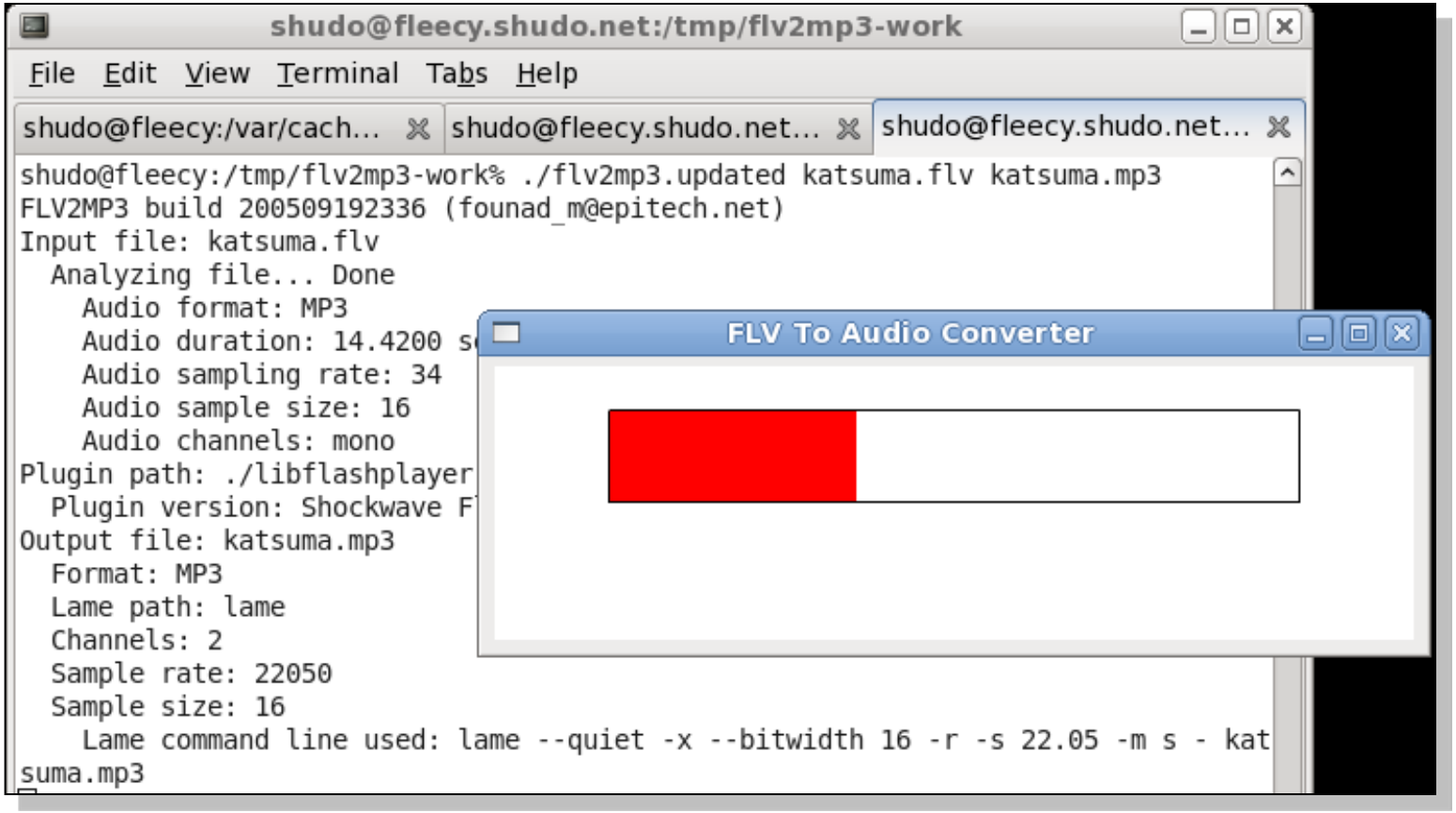
不正命令

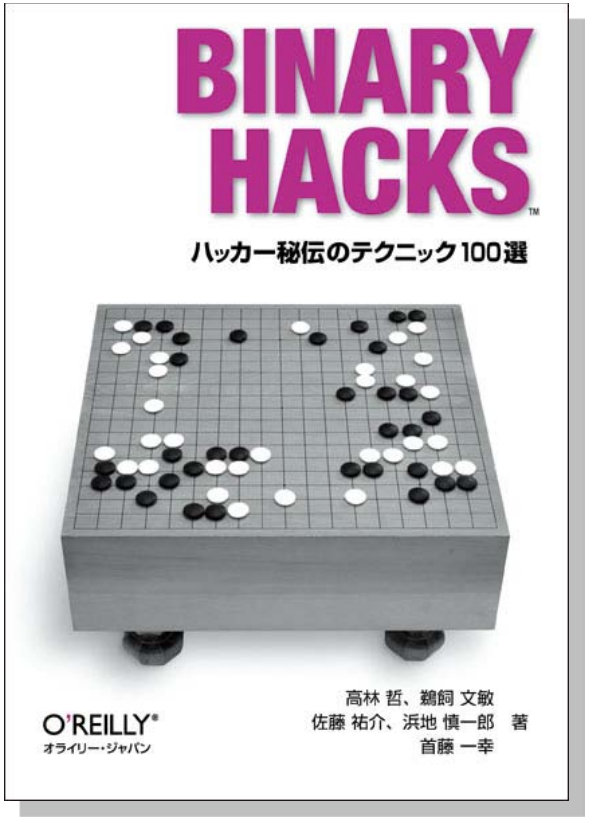
エラー

おかしなことに

ハッピーエンド

- 改造版 libX11.so なしでも
  flv2 mp3 が動いた！



いまどきの Binary Hacks

- ウェブ系でも大活躍！
  - GDB
  - GNU binutils の objdump
  - x86 アセンブリコード
    - 含 calling convention (呼び出し規約)
  - ELF オブジェクト
    - .text セクションとか PLT とか
  - バイナリエディタ
  - dynamic linker/loader のオプション
    - 環境変数 LD_ほげほげ
  - /proc ファイルシステム
    - メモリマップ
  - システムコール・ライブラリ関数
  - gcc の各種オプション
    - -c とか –S とか –E とか
  - 勘