Jared DeMott
Crucial Security, Inc.
Black Hat 2008

Source Code Auditing

Special thanks to ISE for smoking me with this test once upon an interview

Source Code Auditing

- Why?
  - To make software better, or to hack software
- How?
  - With automated tools and/or manually
- Who?
  - Hackers, researchers, security consultants, software QA engineers, etc.
- Where?
  - Do you find out about this … right here. ☺ And in a great book called: "The Art of Software Security Assessment", by Dowd, McDonald, and Schuh
- When?
  - People have been doing this with varying degrees of success for a long time

Only God is Perfect

- Anything designed by humans has a likelihood to contain errors
- The amount, magnitude, and repercussion of said errors varies greatly by problem
- If a bridge fails, people die.
- If a little .bat file you wrote doesn't work well, you might not care too much
- If mission critical server software fails, and hackers steal credit card info from your clients, your company looses money

Bug Hunting

- When you're bug hunting there's three general methodologies
  - Reverse Engineering
  - Source Code Auditing
  - Fuzzing
- We're talking about code auditing here, which makes one big assumption … you have access to the source code. ☺
  - Which means that QA engineers or 3rd party contractors do this a bit more than hackers. Hackers focus more on RE and fuzzing, which the exception of open source software.

Automated or not?

- Free <-> Expensive tools can be used to audit source code
  - This can be done to varying degrees and at varying stages of the development lifecycle
    - For example, some checks are standard at compile time. Both GCC and Visual Studio will warn against the use of dangerous API calls such as *strcpy()*
    - Other tools can work against the completed code base without compiling it
  - RATS and Fortify are examples of automated tools

Language Differences

- Auditors often specialize in particular languages
- Auditing C, C++, C#, python, perl, PHP, Java, Ruby, Visual Basic, etc…
  - They're each quite different
  - Strong knowledge of how to code in that language is required
  - Strong knowledge of typical bug classes are required
    - For that language and in general

In Practice

- From my personal experience it seems that you really only find two types of software auditing contractors out there if you're looking
  - Those that specialize in Web Application code
    - PHP, Java, .ASP, etc.
  - And those that specialize in traditional application code
    - C, C++
  - Either would probably tackle a perl or python job as well

Today's learning

- Today we're just going to focus on one small area of all of that which was just mentioned
  - We're going to do a manual audit of some intentionally buggy and tricky C code
  - It's tricky in the sense that areas that may seem like bugs (use of dangerous API) might be ok in this case
    - yet other subtle bugs that seem ok might be dangerous
  - This is realistic, because much of the low-hanging fruit has already been plucked from modern production quality C programs

A fun little game

- I'll point to each section of code
- I need someone to raise a hand (or both) and tell me why this section of code is or isn't vulnerable
- Using Google, wiki, or testing on your Linux box right now are all fair! ☺
  - There are 8 total bugs in this code
  - Please note that the problems in this sample may not be present in all platforms/compilers.

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
      WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
      syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
      if(strlen(src) < BUFLEN - 1)
            while(*src)
                  *dst++ = *src++;
      *dst = '\xoo';
}
int main(int argc, char *argv[]){
      char buf1[16];
      char buf2[16];
      char buf3[BUFLEN];
      char *buf4;
      char *buf5;
      char buf6[16];
      char *buf7;
      int i, len;

      if(argc != 12)
            exit(0);
      strncpy(buf1, argv[1],
      sizeof(buf1));
      len = atoi(argv[2]);
      if (len < 16)
            memcpy(buf2, argv[3], len);
      else {
            char *buf = malloc(len + 20);
            if(buf){
                  snprintf(buf, len+20,
      "String too long: %s", argv[3]);
                  mylog(LOG_ERR, buf);
            }
      }
      mycpy(buf3, argv[4]);
      strncat(buf3, argv[5],
      sizeof(buf3)-1);
      if(fork()){
            execl("/bin/ls", "/bin/ls",
      argv[6], 0);
      }
      // filter metacharacters
      char *p;
      if(p = strchr(argv[7], '&'))
            *p = 0;
      if(p = strchr(argv[7], '`'))
            *p = 0;
      if(p = strchr(argv[7], ';'))
            *p = 0;
      if(p = strchr(argv[7], '|'))
            *p = 0;
      if(strlen(argv[7]) < 1024){
            buf4 = malloc(20 +
      strlen(argv[7]));
            sprintf(buf4, "/bin/cat %s",
      argv[7]);
            system(buf4);
      }

      buf5 = malloc(strlen(argv[8]) +
      strlen(argv[9]) + 2);
      strcpy(buf5, argv[8]);
      strcat(buf5, argv[9]);
      memcpy(buf6, argv[10],
      strlen(argv[10]));
      buf7 = malloc(4 * DWORDSIZE);
      for(i=0; i<4; i++){
            memcpy(buf7 + 4 * i,
      argv[11] + 4 * i, DWORDSIZE);
      }

      printf("\nGot %s, (%d) %s, %s,
      %s, %s, %s, %s\n", buf1, len,
      buf2,buf3, buf4, buf5, buf6,
      buf7);
}
```

Yup that's bug number 1

strncpy(buf1, argv[1], sizeof(buf1));

- No null termination, if string is larger than 16
  - This is ugly, since *strncat()* is the opposite – it's the culprit of many off-by-ones since it'll go one byte beyond the boundary.
- Strncpy() works by copying the specified number of bytes (from src -> dst), or up until the first null, since it's a string based operation
- This could cause various errors depending on how buf1 is later used

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
    WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
    syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
    if(strlen(src) < BUFLEN - 1)
        while(*src)
            *dst++ = *src++;
    *dst = '\xoo';
}
int main(int argc, char *argv[]){
    char buf1[16];
    char buf2[16];
    char buf3[BUFLEN];
    char *buf4;
    char *buf5;
    char buf6[16];
    char *buf7;
    int i, len;

    if(argc != 12)
        exit(0);
    strncpy(buf1, argv[1],
    sizeof(buf1));
    len = atoi(argv[2]);
    if (len < 16)
        memcpy(buf2, argv[3], len);
    else {
        char *buf = malloc(len + 20);
        if(buf){
            snprintf(buf, len+20,
    "String too long: %s", argv[3]);
            mylog(LOG_ERR, buf);
        }
    }
    mycpy(buf3, argv[4]);
    strncat(buf3, argv[5],
    sizeof(buf3)-1);
    if(fork()){
        execl("/bin/ls", "/bin/ls",
    argv[6], 0);
    }
    // filter metacharacters
    char *p;
    if(p = strchr(argv[7], '&'))
        *p = 0;
    if(p = strchr(argv[7], '`'))
        *p = 0;
    if(p = strchr(argv[7], ';'))
        *p = 0;
    if(p = strchr(argv[7], '|'))
        *p = 0;
    if(strlen(argv[7]) < 1024){
        buf4 = malloc(20 +
    strlen(argv[7]));
        sprintf(buf4, "/bin/cat %s",
    argv[7]);
        system(buf4);
    }

    buf5 = malloc(strlen(argv[8]) +
    strlen(argv[9]) + 2);
    strcpy(buf5, argv[8]);
    strcat(buf5, argv[9]);
    memcpy(buf6, argv[10],
    strlen(argv[10]));
    buf7 = malloc(4 * DWORDSIZE);
    for(i=0; i<4; i++){
        memcpy(buf7 + 4 * i,
    argv[11] + 4 * i, DWORDSIZE);
    }

    printf("\nGot %s, (%d) %s, %s,
    %s, %s, %s, %s\n", buf1, len,
    buf2,buf3, buf4, buf5, buf6,
    buf7);
}
```

That's bug number 2

```
if (len < 16)
        memcpy(buf2, argv[3], len);
```

- Why you ask?
  - Try putting in a len of -4
  - Why would this cause a problem?
    - That's right, because in the 'if' statement we perform a signed comparison since *len* is defined as type "int", but according to the memcpy prototype: void *memcpy(void **s1*, const void **s2*, size_t *n*);
    - We note that the size is "size_t". Well, what's that?
      - That's an "unsigned int" on most all systems, so -4 = 4294967292 in decimal

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
    WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
    syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
    if(strlen(src) < BUFLEN - 1)
        while(*src)
            *dst++ = *src++;
    *dst = '\xoo';
}
int main(int argc, char *argv[]){
    char buf1[16];
    char buf2[16];
    char buf3[BUFLEN];
    char *buf4;
    char *buf5;
    char buf6[16];
    char *buf7;
    int i, len;

    if(argc != 12)
        exit(0);
    strncpy(buf1, argv[1],
    sizeof(buf1));
    len = atoi(argv[2]);
    if (len < 16)
        memcpy(buf2, argv[3], len);
    else {
        char *buf = malloc(len + 20);
        if(buf){
            snprintf(buf, len+20,
    "String too long: %s", argv[3]);
            mylog(LOG_ERR, buf);
        }
    }
    mycpy(buf3, argv[4]);
    strncat(buf3, argv[5],
    sizeof(buf3)-1);
    if(fork()){
        execl("/bin/ls", "/bin/ls",
    argv[6], 0);
    }
    // filter metacharacters
    char *p;
    if(p = strchr(argv[7], '&'))
        *p = 0;
    if(p = strchr(argv[7], '`'))
        *p = 0;
    if(p = strchr(argv[7], ';'))
        *p = 0;
    if(p = strchr(argv[7], '|'))
        *p = 0;
    if(strlen(argv[7]) < 1024){
        buf4 = malloc(20 +
    strlen(argv[7]));
        sprintf(buf4, "/bin/cat %s",
    argv[7]);
        system(buf4);
    }

    buf5 = malloc(strlen(argv[8]) +
    strlen(argv[9]) + 2);
    strcpy(buf5, argv[8]);
    strcat(buf5, argv[9]);
    memcpy(buf6, argv[10],
    strlen(argv[10]));
    buf7 = malloc(4 * DWORDSIZE);
    for(i=0; i<4; i++){
        memcpy(buf7 + 4 * i,
    argv[11] + 4 * i, DWORDSIZE);
    }

    printf("\nGot %s, (%d) %s, %s,
    %s, %s, %s, %s\n", buf1, len,
    buf2,buf3, buf4, buf5, buf6,
    buf7);
}
```

Looking close … and no bug here

```
len = atoi(argv[2]);                    //Could be +/- 2147483647
if (len < 16)
        memcpy(buf2, argv[3], len);     //but if negative go here
    else {                              //else if positive go here
        char *buf = malloc(len + 20);
        if(buf){
            snprintf(buf, len+20, "String too long: %s", argv[3]);
```

- You might be tempted to see an overflow here, but not so…
  - And as a real treat, we even check to see if malloc failed

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
      WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
      syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
      if(strlen(src) < BUFLEN - 1)
            while(*src)
                  *dst++ = *src++;
      *dst = '\xoo';
}
int main(int argc, char *argv[]){
      char buf1[16];
      char buf2[16];
      char buf3[BUFLEN];
      char *buf4;
      char *buf5;
      char buf6[16];
      char *buf7;
      int i, len;

      if(argc != 12)
            exit(0);
      strncpy(buf1, argv[1],
      sizeof(buf1));
      len = atoi(argv[2]);
      if (len < 16)
            memcpy(buf2, argv[3], len);
      else {
            char *buf = malloc(len + 20);
            if(buf){
                  snprintf(buf, len+20,
      "String too long: %s", argv[3]);
                  mylog(LOG_ERR, buf);
            }
      }
      mycpy(buf3, argv[4]);
      strncat(buf3, argv[5],
      sizeof(buf3)-1);
      if(fork()){
            execl("/bin/ls", "/bin/ls",
      argv[6], 0);
      }
      // filter metacharacters
      char *p;
      if(p = strchr(argv[7], '&'))
            *p = 0;
      if(p = strchr(argv[7], '`'))
            *p = 0;
      if(p = strchr(argv[7], ';'))
            *p = 0;
      if(p = strchr(argv[7], '|'))
            *p = 0;
      if(strlen(argv[7]) < 1024){
            buf4 = malloc(20 +
      strlen(argv[7]));
            sprintf(buf4, "/bin/cat %s",
      argv[7]);
            system(buf4);
      }

      buf5 = malloc(strlen(argv[8]) +
      strlen(argv[9]) + 2);
      strcpy(buf5, argv[8]);
      strcat(buf5, argv[9]);
      memcpy(buf6, argv[10],
      strlen(argv[10]));
      buf7 = malloc(4 * DWORDSIZE);
      for(i=0; i<4; i++){
            memcpy(buf7 + 4 * i,
      argv[11] + 4 * i, DWORDSIZE);
      }

      printf("\nGot %s, (%d) %s, %s,
      %s, %s, %s, %s\n", buf1, len,
      buf2,buf3, buf4, buf5, buf6,
      buf7);
}
```

That's bug number 3

```
void mylog(int kind, char *msg){
      syslog(LOG_USER | kind, msg);
}
mylog(LOG_ERR, buf);
```

- Strait up format string buf
  - Should be syslog(LOG_USER | kind, "%s", msg)
  - Enter something like %n%n%n%n to trigger the bug

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
    WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
    syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
    if(strlen(src) < BUFLEN - 1)
        while(*src)
            *dst++ = *src++;
    *dst = '\xoo';
}
int main(int argc, char *argv[]){
    char buf1[16];
    char buf2[16];
    char buf3[BUFLEN];
    char *buf4;
    char *buf5;
    char buf6[16];
    char *buf7;
    int i, len;

    if(argc != 12)
        exit(0);
    strncpy(buf1, argv[1],
    sizeof(buf1));
    len = atoi(argv[2]);
    if (len < 16)
        memcpy(buf2, argv[3], len);
    else {
        char *buf = malloc(len + 20);
        if(buf){
            snprintf(buf, len+20,
    "String too long: %s", argv[3]);
            mylog(LOG_ERR, buf);
        }
    }
    mycpy(buf3, argv[4]);
    strncat(buf3, argv[5],
    sizeof(buf3)-1);
    if(fork()){
        execl("/bin/ls", "/bin/ls",
    argv[6], 0);
    }
    // filter metacharacters
    char *p;
    if(p = strchr(argv[7], '&'))
        *p = 0;
    if(p = strchr(argv[7], '`'))
        *p = 0;
    if(p = strchr(argv[7], ';'))
        *p = 0;
    if(p = strchr(argv[7], '|'))
        *p = 0;
    if(strlen(argv[7]) < 1024){
        buf4 = malloc(20 +
    strlen(argv[7]));
        sprintf(buf4, "/bin/cat %s",
    argv[7]);
        system(buf4);
    }

    buf5 = malloc(strlen(argv[8]) +
    strlen(argv[9]) + 2);
    strcpy(buf5, argv[8]);
    strcat(buf5, argv[9]);
    memcpy(buf6, argv[10],
    strlen(argv[10]));
    buf7 = malloc(4 * DWORDSIZE);
    for(i=0; i<4; i++){
        memcpy(buf7 + 4 * i,
    argv[11] + 4 * i, DWORDSIZE);
    }

    printf("\nGot %s, (%d) %s, %s,
    %s, %s, %s, %s\n", buf1, len,
    buf2,buf3, buf4, buf5, buf6,
    buf7);
}
```

Nope, we're ok here

```
#define BUFLEN 16
void mycpy(char *dst, char *src){
      if(strlen(src) < BUFLEN - 1)
            while(*src)
                  *dst++ = *src++;
      *dst = '\xoo';
}
mycpy(buf3, argv[4]);
```

- I don't blame you for thinking twice on this one
  - Homegrown copies are always suspect
  - Particularly since BUFLEN is globally defined

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
      WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
      syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
      if(strlen(src) < BUFLEN - 1)
            while(*src)
                  *dst++ = *src++;
      *dst = '\xoo';
}
int main(int argc, char *argv[]){
      char buf1[16];
      char buf2[16];
      char buf3[BUFLEN];
      char *buf4;
      char *buf5;
      char buf6[16];
      char *buf7;
      int i, len;

      if(argc != 12)
            exit(0);
      strncpy(buf1, argv[1],
      sizeof(buf1));
      len = atoi(argv[2]);
      if (len < 16)
            memcpy(buf2, argv[3], len);
      else {
            char *buf = malloc(len + 20);
            if(buf){
                  snprintf(buf, len+20,
      "String too long: %s", argv[3]);
                  mylog(LOG_ERR, buf);
            }
      }
      mycpy(buf3, argv[4]);
      strncat(buf3, argv[5],
      sizeof(buf3)-1);
      if(fork()){
            execl("/bin/ls", "/bin/ls",
      argv[6], 0);
      }
      // filter metacharacters
      char *p;
      if(p = strchr(argv[7], '&'))
            *p = 0;
      if(p = strchr(argv[7], '`'))
            *p = 0;
      if(p = strchr(argv[7], ';'))
            *p = 0;
      if(p = strchr(argv[7], '|'))
            *p = 0;
      if(strlen(argv[7]) < 1024){
            buf4 = malloc(20 +
      strlen(argv[7]));
            sprintf(buf4, "/bin/cat %s",
      argv[7]);
            system(buf4);
      }

      buf5 = malloc(strlen(argv[8]) +
      strlen(argv[9]) + 2);
      strcpy(buf5, argv[8]);
      strcat(buf5, argv[9]);
      memcpy(buf6, argv[10],
      strlen(argv[10]));
      buf7 = malloc(4 * DWORDSIZE);
      for(i=0; i<4; i++){
            memcpy(buf7 + 4 * i,
      argv[11] + 4 * i, DWORDSIZE);
      }

      printf("\nGot %s, (%d) %s, %s,
      %s, %s, %s, %s\n", buf1, len,
      buf2,buf3, buf4, buf5, buf6,
      buf7);
}
```

Yikes! Here's bug 4

mycpy(buf3, argv[4]); //was ok ☺
strncat(buf3, argv[5], sizeof(buf3)-1); //bad

- You've got to lookout for strncat()
  - Seems ok, sorta like strncpy() would be ok here
  - But strcats begin where the last string left off
    - i.e begin at the location of the terminating NULL byte
  - So the mycpy was ok by itself, but a strncat to the same buffer right after will cause a buffer overflow with the right sized inputs

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
    WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
    syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
    if(strlen(src) < BUFLEN - 1)
        while(*src)
            *dst++ = *src++;
    *dst = '\xoo';
}
int main(int argc, char *argv[]){
    char buf1[16];
    char buf2[16];
    char buf3[BUFLEN];
    char *buf4;
    char *buf5;
    char buf6[16];
    char *buf7;
    int i, len;

    if(argc != 12)
        exit(0);
    strncpy(buf1, argv[1],
    sizeof(buf1));
    len = atoi(argv[2]);
    if (len < 16)
        memcpy(buf2, argv[3], len);
    else {
        char *buf = malloc(len + 20);
        if(buf){
            snprintf(buf, len+20,
    "String too long: %s", argv[3]);
            mylog(LOG_ERR, buf);
        }
    }
    mycpy(buf3, argv[4]);
    strncat(buf3, argv[5],
    sizeof(buf3)-1);
    if(fork()){
        execl("/bin/ls", "/bin/ls",
    argv[6], 0);
    }
    // filter metacharacters
    char *p;
    if(p = strchr(argv[7], '&'))
        *p = 0;
    if(p = strchr(argv[7], '`'))
        *p = 0;
    if(p = strchr(argv[7], ';'))
        *p = 0;
    if(p = strchr(argv[7], '|'))
        *p = 0;
    if(strlen(argv[7]) < 1024){
        buf4 = malloc(20 +
    strlen(argv[7]));
        sprintf(buf4, "/bin/cat %s",
    argv[7]);
        system(buf4);
    }

    buf5 = malloc(strlen(argv[8]) +
    strlen(argv[9]) + 2);
    strcpy(buf5, argv[8]);
    strcat(buf5, argv[9]);
    memcpy(buf6, argv[10],
    strlen(argv[10]));
    buf7 = malloc(4 * DWORDSIZE);
    for(i=0; i<4; i++){
        memcpy(buf7 + 4 * i,
    argv[11] + 4 * i, DWORDSIZE);
    }

    printf("\nGot %s, (%d) %s, %s,
    %s, %s, %s, %s\n", buf1, len,
    buf2,buf3, buf4, buf5, buf6,
    buf7);
}
```

Tricky, but it's ok

execl("/bin/ls", "/bin/ls", argv[6], 0);

- I know what you might have been thinking, because I was too … when you see code getting launched from inside a program, which includes user input for parameters, you're thinking command injection
  - But because argv[6] is used directly in an execl() typical shell escape tricks such as "; cat /etc/passwd" won't work

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
    WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
    syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
    if(strlen(src) < BUFLEN - 1)
        while(*src)
            *dst++ = *src++;
    *dst = '\xoo';
}
int main(int argc, char *argv[]){
    char buf1[16];
    char buf2[16];
    char buf3[BUFLEN];
    char *buf4;
    char *buf5;
    char buf6[16];
    char *buf7;
    int i, len;

if(argc != 12)
    exit(0);
strncpy(buf1, argv[1],
sizeof(buf1));
len = atoi(argv[2]);
if (len < 16)
    memcpy(buf2, argv[3], len);
else {
    char *buf = malloc(len + 20);
    if(buf){
        snprintf(buf, len+20,
"String too long: %s", argv[3]);
        mylog(LOG_ERR, buf);
    }
}
mycpy(buf3, argv[4]);
strncat(buf3, argv[5],
sizeof(buf3)-1);
if(fork()){
    execl("/bin/ls", "/bin/ls",
argv[6], 0);
}
// filter metacharacters
char *p;
if(p = strchr(argv[7], '&'))
    *p = 0;
if(p = strchr(argv[7], '`'))
    *p = 0;
if(p = strchr(argv[7], ';'))
    *p = 0;
if(p = strchr(argv[7], '|'))
    *p = 0;
if(strlen(argv[7]) < 1024){
    buf4 = malloc(20 +
strlen(argv[7]));
    sprintf(buf4, "/bin/cat %s",
argv[7]);
    system(buf4);
}

buf5 = malloc(strlen(argv[8]) +
strlen(argv[9]) + 2);
strcpy(buf5, argv[8]);
strcat(buf5, argv[9]);
memcpy(buf6, argv[10],
strlen(argv[10]));
buf7 = malloc(4 * DWORDSIZE);
for(i=0; i<4; i++){
    memcpy(buf7 + 4 * i,
argv[11] + 4 * i, DWORDSIZE);
}

printf("\nGot %s, (%d) %s, %s,
%s, %s, %s, %s\n", buf1, len,
buf2,buf3, buf4, buf5, buf6,
buf7);
}
```

Ok here

```
if(strlen(argv[7]) < 1024){
        buf4 = malloc(20 + strlen(argv[7]));
        sprintf(buf4, "/bin/cat %s", argv[7]);
}
```

- Again, assuming malloc doesn't fail and return 0, we'll receive a big enough buffer.
- Strlen can't be negative and can't be larger than 1024

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
      WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
      syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
      if(strlen(src) < BUFLEN - 1)
            while(*src)
                  *dst++ = *src++;
      *dst = '\xoo';
}
int main(int argc, char *argv[]){
      char buf1[16];
      char buf2[16];
      char buf3[BUFLEN];
      char *buf4;
      char *buf5;
      char buf6[16];
      char *buf7;
      int i, len;

      if(argc != 12)
            exit(0);
      strncpy(buf1, argv[1],
      sizeof(buf1));
      len = atoi(argv[2]);
      if (len < 16)
            memcpy(buf2, argv[3], len);
      else {
            char *buf = malloc(len + 20);
            if(buf){
                  snprintf(buf, len+20,
      "String too long: %s", argv[3]);
                  mylog(LOG_ERR, buf);
            }
      }
      mycpy(buf3, argv[4]);
      strncat(buf3, argv[5],
      sizeof(buf3)-1);
      if(fork()){
            execl("/bin/ls", "/bin/ls",
      argv[6], 0);
      }
      // filter metacharacters
      char *p;
      if(p = strchr(argv[7], '&'))
            *p = 0;
      if(p = strchr(argv[7], '`'))
            *p = 0;
      if(p = strchr(argv[7], ';'))
            *p = 0;
      if(p = strchr(argv[7], '|'))
            *p = 0;
      if(strlen(argv[7]) < 1024){
            buf4 = malloc(20 +
      strlen(argv[7]));
            sprintf(buf4, "/bin/cat %s",
      argv[7]);
            system(buf4);
      }

      buf5 = malloc(strlen(argv[8]) +
      strlen(argv[9]) + 2);
      strcpy(buf5, argv[8]);
      strcat(buf5, argv[9]);
      memcpy(buf6, argv[10],
      strlen(argv[10]));
      buf7 = malloc(4 * DWORDSIZE);
      for(i=0; i<4; i++){
            memcpy(buf7 + 4 * i,
      argv[11] + 4 * i, DWORDSIZE);
      }

      printf("\nGot %s, (%d) %s, %s,
      %s, %s, %s, %s\n", buf1, len,
      buf2,buf3, buf4, buf5, buf6,
      buf7);
}
```

Ah! There you are bug 5

```
char *p;
        if(p = strchr(argv[7], '&'))
                *p = 0;
        if(p = strchr(argv[7], '`'))
                *p = 0;
        if(p = strchr(argv[7], ';'))
                *p = 0;
        if(p = strchr(argv[7], '|'))
                *p = 0;
        if(strlen(argv[7]) < 1024){
                buf4 = malloc(20 + strlen(argv[7]));
                sprintf(buf4, "/bin/cat %s", argv[7]);
                system(buf4);
        }
```

- Why you ask? Similar to the execl(), except with system we now have a problem. System executes in a shell of it's own and therefore shell escape characters will work. We filtered for some, but black-listing is never as good as white-listing
- Something like " mypasswdfile > /etc/passwd" proves the point

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
    WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
    syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
    if(strlen(src) < BUFLEN - 1)
        while(*src)
            *dst++ = *src++;
    *dst = '\xoo';
}
int main(int argc, char *argv[]){
    char buf1[16];
    char buf2[16];
    char buf3[BUFLEN];
    char *buf4;
    char *buf5;
    char buf6[16];
    char *buf7;
    int i, len;

if(argc != 12)
    exit(0);
strncpy(buf1, argv[1],
sizeof(buf1));
len = atoi(argv[2]);
if (len < 16)
    memcpy(buf2, argv[3], len);
else {
    char *buf = malloc(len + 20);
    if(buf){
        snprintf(buf, len+20,
"String too long: %s", argv[3]);
        mylog(LOG_ERR, buf);
    }
}
mycpy(buf3, argv[4]);
strncat(buf3, argv[5],
sizeof(buf3)-1);
if(fork()){
    execl("/bin/ls", "/bin/ls",
argv[6], 0);
}
// filter metacharacters
char *p;
if(p = strchr(argv[7], '&'))
    *p = 0;
if(p = strchr(argv[7], '`'))
    *p = 0;
if(p = strchr(argv[7], ';'))
    *p = 0;
if(p = strchr(argv[7], '|'))
    *p = 0;
if(strlen(argv[7]) < 1024){
    buf4 = malloc(20 +
strlen(argv[7]));
    sprintf(buf4, "/bin/cat %s",
argv[7]);
    system(buf4);
}

buf5 = malloc(strlen(argv[8]) +
strlen(argv[9]) + 2);
strcpy(buf5, argv[8]);
strcat(buf5, argv[9]);
memcpy(buf6, argv[10],
strlen(argv[10]));
buf7 = malloc(4 * DWORDSIZE);
for(i=0; i<4; i++){
    memcpy(buf7 + 4 * i,
argv[11] + 4 * i, DWORDSIZE);
}

printf("\nGot %s, (%d) %s, %s,
%s, %s, %s, %s\n", buf1, len,
buf2,buf3, buf4, buf5, buf6,
buf7);
}
```

Um… are we ok here? Lets see next

```
buf5 = malloc(strlen(argv[8]) + strlen(argv[9]) + 2);
strcpy(buf5, argv[8]);
```

- Buf5 was dynamically allocated via user input
  - Often fishy
  - Also, again, we don't check to see if malloc() returned a valid pointer

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
    WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
    syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
    if(strlen(src) < BUFLEN - 1)
        while(*src)
            *dst++ = *src++;
    *dst = '\xoo';
}
int main(int argc, char *argv[]){
    char buf1[16];
    char buf2[16];
    char buf3[BUFLEN];
    char *buf4;
    char *buf5;
    char buf6[16];
    char *buf7;
    int i, len;

if(argc != 12)
    exit(0);
strncpy(buf1, argv[1],
sizeof(buf1));
len = atoi(argv[2]);
if (len < 16)
    memcpy(buf2, argv[3], len);
else {
    char *buf = malloc(len + 20);
    if(buf){
        snprintf(buf, len+20,
"String too long: %s", argv[3]);
        mylog(LOG_ERR, buf);
    }
}
mycpy(buf3, argv[4]);
strncat(buf3, argv[5],
sizeof(buf3)-1);
if(fork()){
    execl("/bin/ls", "/bin/ls",
argv[6], 0);
}
// filter metacharacters
char *p;
if(p = strchr(argv[7], '&'))
    *p = 0;
if(p = strchr(argv[7], '`'))
    *p = 0;
if(p = strchr(argv[7], ';'))
    *p = 0;
if(p = strchr(argv[7], '|'))
    *p = 0;
if(strlen(argv[7]) < 1024){
    buf4 = malloc(20 +
strlen(argv[7]));
    sprintf(buf4, "/bin/cat %s",
argv[7]);
    system(buf4);
}

buf5 = malloc(strlen(argv[8]) +
strlen(argv[9]) + 2);
strcpy(buf5, argv[8]);
strcat(buf5, argv[9]);
memcpy(buf6, argv[10],
strlen(argv[10]));
buf7 = malloc(4 * DWORDSIZE);
for(i=0; i<4; i++){
    memcpy(buf7 + 4 * i,
argv[11] + 4 * i, DWORDSIZE);
}

printf("\nGot %s, (%d) %s, %s,
%s, %s, %s, %s\n", buf1, len,
buf2,buf3, buf4, buf5, buf6,
buf7);
}
```

BUG 6! An Integer overflow

```
buf5 = malloc(strlen(argv[8]) + strlen(argv[9]) + 2);
    strcpy(buf5, argv[8]);
    strcat(buf5, argv[9]);
```

- Could have exploited the previous copy as well
- Granted this bug would be a bit tricky to deliver and exploit
  - In this case we're not getting the size of argv8/9 from a input number, but rather by calculating the strlen.
  - For example, if strings are of size \x7fffffff + \x7fffffff +2 = 0
    - No memory gets allocated and we have the worlds hugest buffer overflow, with either of the copy operations

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
    WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
    syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
    if(strlen(src) < BUFLEN - 1)
        while(*src)
            *dst++ = *src++;
    *dst = '\xoo';
}
int main(int argc, char *argv[]){
    char buf1[16];
    char buf2[16];
    char buf3[BUFLEN];
    char *buf4;
    char *buf5;
    char buf6[16];
    char *buf7;
    int i, len;

if(argc != 12)
    exit(0);
strncpy(buf1, argv[1],
sizeof(buf1));
len = atoi(argv[2]);
if (len < 16)
    memcpy(buf2, argv[3], len);
else {
    char *buf = malloc(len + 20);
    if(buf){
        snprintf(buf, len+20,
"String too long: %s", argv[3]);
        mylog(LOG_ERR, buf);
    }
}
mycpy(buf3, argv[4]);
strncat(buf3, argv[5],
sizeof(buf3)-1);
if(fork()){
    execl("/bin/ls", "/bin/ls",
argv[6], 0);
}
// filter metacharacters
char *p;
if(p = strchr(argv[7], '&'))
    *p = 0;
if(p = strchr(argv[7], '`'))
    *p = 0;
if(p = strchr(argv[7], ';'))
    *p = 0;
if(p = strchr(argv[7], '|'))
    *p = 0;
if(strlen(argv[7]) < 1024){
    buf4 = malloc(20 +
strlen(argv[7]));
    sprintf(buf4, "/bin/cat %s",
argv[7]);
    system(buf4);
}

buf5 = malloc(strlen(argv[8]) +
strlen(argv[9]) + 2);
strcpy(buf5, argv[8]);
strcat(buf5, argv[9]);
memcpy(buf6, argv[10],
strlen(argv[10]));
buf7 = malloc(4 * DWORDSIZE);
for(i=0; i<4; i++){
    memcpy(buf7 + 4 * i,
argv[11] + 4 * i, DWORDSIZE);
}

printf("\nGot %s, (%d) %s, %s,
%s, %s, %s, %s\n", buf1, len,
buf2,buf3, buf4, buf5, buf6,
buf7);
}
```

Ok, this one was easy. Bug 7

```
char buf6[16];
memcpy(buf6, argv[10], strlen(argv[10]));
```

- Strait up overflow here
  - Any string bigger than 16 will begin clobbering bytes on the stacked, like the stored EBP and EIP

```
#include <syslog.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define BUFLEN 16
#define WORDSIZE 2
#define DWORDSIZE
    WORDSIZE+WORDSIZE
void mylog(int kind, char *msg){
    syslog(LOG_USER | kind, msg);
}
void mycpy(char *dst, char *src){
    if(strlen(src) < BUFLEN - 1)
        while(*src)
            *dst++ = *src++;
    *dst = '\xoo';
}
int main(int argc, char *argv[]){
    char buf1[16];
    char buf2[16];
    char buf3[BUFLEN];
    char *buf4;
    char *buf5;
    char buf6[16];
    char *buf7;
    int i, len;

if(argc != 12)
    exit(0);
strncpy(buf1, argv[1],
sizeof(buf1));
len = atoi(argv[2]);
if (len < 16)
    memcpy(buf2, argv[3], len);
else {
    char *buf = malloc(len + 20);
    if(buf){
        snprintf(buf, len+20,
"String too long: %s", argv[3]);
        mylog(LOG_ERR, buf);
    }
}
mycpy(buf3, argv[4]);
strncat(buf3, argv[5],
sizeof(buf3)-1);
if(fork()){
    execl("/bin/ls", "/bin/ls",
argv[6], 0);
}
// filter metacharacters
char *p;
if(p = strchr(argv[7], '&'))
    *p = 0;
if(p = strchr(argv[7], '`'))
    *p = 0;
if(p = strchr(argv[7], ';'))
    *p = 0;
if(p = strchr(argv[7], '|'))
    *p = 0;
if(strlen(argv[7]) < 1024){
    buf4 = malloc(20 +
strlen(argv[7]));
    sprintf(buf4, "/bin/cat %s",
argv[7]);
    system(buf4);
}

buf5 = malloc(strlen(argv[8]) +
strlen(argv[9]) + 2);
strcpy(buf5, argv[8]);
strcat(buf5, argv[9]);
memcpy(buf6, argv[10],
strlen(argv[10]));
buf7 = malloc(4 * DWORDSIZE);
for(i=0; i<4; i++){
    memcpy(buf7 + 4 * i,
argv[11] + 4 * i, DWORDSIZE);
}

printf("\nGot %s, (%d) %s, %s,
%s, %s, %s, %s\n", buf1, len,
buf2,buf3, buf4, buf5, buf6,
buf7);
}
```

The best for last

```
#define WORDSIZE 2
#define DWORDSIZE WORDSIZE+WORDSIZE

buf7 = malloc(4 * DWORDSIZE);
      for(i=0; i<4; i++){
            memcpy(buf7 + 4 * i,    argv[11] + 4 * i,
   DWORDSIZE);
      }
```

- I love this one, so I'll give you a hint…
  - The problem lies in how MACROs work

Bug 8: The MAC daddy

- So let's calculate the memory allocated

```
#define WORDSIZE 2
#define DWORDSIZE WORDSIZE+WORDSIZE
buf7 = malloc(4 * DWORDSIZE);
```

  - 4*2+2=10
- Now let's calculate the memory used

```
for(i=0; i<4; i++){
    memcpy(buf7 + 4 * i,    argv[11] + 4 * i, DWORDSIZE);
}
```

  - DWORDSIZE = 2 + 2 = 4
  - 4+4+4+4=16

Summary

- So, who thinks they're cut out to audit source code all day for a living?
  - Fear not, it really becomes much more fun as you get better at it, and the pay is quite nice if you are good at it. ☺
  - I love the security field:
    - Always changing
    - Always in need of new talent
    - Getting harder all the time, as system security gets better
  - Rather you're a hacker, or a network security engineer trying to keep bad guys out, security is relevant, often required (HIPPA), and fascinating