Mittuniversitetet
MID SWEDEN UNIVERSITY

Final exam

DT145G Computer Security

Daniel Bosk

Department of Information and Communication Systems,
Mid Sweden University, SE-851 70 Sundsvall
Email: daniel.bosk@miun.se
Phone: 010-142 8709

2015-06-04

Instructions

Carefully read the questions before you start answering them. Note the time limit of the exam and plan your answers accordingly. Only answer the question, do not write about subjects remotely related to the question.

Write your answers on separate sheets, not on the exam paper. Only write on one side of the sheets. Start each question on a new sheet. Do not forget to *motivate your answers.*

Make sure you write your answers clearly, if I cannot read an answer the answer will be awarded no points—even if the answer is correct. The questions are *not* sorted by difficulty.

Time 5 hours.

Aids Dictionary.

Maximum points 28

Questions 5

Preliminary grades

The following grading criteria applies: E $\geq$ 50%, D $\geq$ 60%, C $\geq$ 70%, B $\geq$ 80%, A $\geq$ 90%.

Questions

The questions are given below. They are not given in any particular order.

1. Explain the following terms:

(1p) (a) Confidentiality

(1p) (b) Integrity

(1p) (c) Availability

(1p) (d) Accountability

(1p) (e) Non-Repudiation

> **Suggested solution** See [**Gollmann2011cs**] and [**Anderson2008sea**] for definitions.

2. A user wishes to provide confidentiality to a file.

(3p) (a) She can accomplish this through mechanisms provided in the operating system. Explain how this works and what are the limits.

(3p) (b) She can also accomplish this through purely cryptographic mechanisms. Explain how this works and what are the limits.

> **Suggested solution** The first way she's securing her file is by using access control mechanisms in the operating system (OS).
>
> Assuming we have physical access to the computer, then we can just read the raw data from the disk. This can be accomplished by either booting our own OS on her computer, or by removing the disk.
>
> If we don't have physical access we can always try to bypass the access control mechanisms in other ways, e.g. by gaining privileges in the system or seeing if the OS has failed to protect reading from the raw disk (i.e. not using the file system).
>
> The main point here is that the operating system must be working correctly for its mechanisms to be effective. The *running* operating system will provide confidentiality by not allowing other users' requests to open the file.
>
> The most obvious way to have system independent security for this file is to encrypt it, i.e. using cryptographic mechanisms. This way no one can read it unless they have access to the key, and this is true no matter if you change the environment. (Of course, if the system is untrusted someone can get to the key that way, but that's outside the scope of this question.)

3. The University password composition policy is as follows[1]: The password must be at least eight (8) characters long, further, the first eight characters must contain two upper-case and two lower-case letters as well as two numbers.

(2p) (a) Estimate the upper bound of the entropy of the given password policy.

(3p) (b) Approximate the information gained by an attacker who learns this policy is in use.

> **Suggested solution** We assume uniform randomness, this yields the highest entropy. We know that users don't use uniform randomness, so the actual entropy will be lower and thus this will be the upper bound.
>
> The entropy per lower-case letter in the alphabet is approximately 5 bits, the same for the upper-case letters, 6 bits if they are mixed. It's approximately 3 bits for the numbers, it's

[1] It's slightly more advanced than this, but it's simplified here for reasons of convenience. Also, the effect might actually be the same anyway, depending on if users actually use the extension or not.

approximately 6 bits for all letters and numbers together ($64 = 2^6$). Hence, a password of length 8 would yield $6 \times 8 = 48$ bits of entropy. However, we know from [**Komanduri2011opa**] that this is more around 30 bits (basic8).

If the attacker learns the password composition policy, this reduces entropy by: 2 bits for forced lower-case (1 bit per letter, remove half of the possibilities), the same for forced upper-case, 6 bits for numbers (3 bits per letter, remove the majority of the possibilities). This means a total of $10 = 2 + 2 + 6$ bits reduction of entropy. This leaves us with a maximal entropy of $38 = 48 - 10$ bits. We know from [**Komanduri2011opa**] that this is closer to 34 bits of entropy (comprehensive8).

4. The apps in the Google Play store or Apple's AppStore are provided with some DRM, e.g. you cannot copy a paid-for app from one phone to another etc. There are also other things you are not allowed to do with these smartphones, e.g. installing whatever software you like, modifying the software in any way you like, etc. (However, many know of the terms "jail-breaking" or "rooting" their phones, which bypasses these mechanisms on the phones.)

(3p) (a) Explain how this type of protection system works, including the fundamental assumptions needed for it to fulfil its purpose.

(3p) (b) Explain how the above system can be broken and whether it could be fully solved or not.

Suggested solution The fundamental assumption is based on the fact that the operating system remains in control and that it cannot be modified. This is accomplished by the operating system being the only provided interface with which the user can interact with the device.

Essentially the user is allowed to use the operating system as an unprivileged user, whereas the system keeps superuser (or root) privileges for itself. Hence, as soon as the user gains these privileges, the system breaks down. This can be accomplished in many ways, e.g. exploiting bugs in privileged applications or the OS itself—and is thus called "rooting the device".

5. Look at the C code in Listing 1 on the next page.

(3p) (a) Identify all vulnerabilities in that code and motivate by stating how they can be exploited.

(3p) (b) Suggest improvements to remedy these vulnerabilities, you must motivate why they work.

Suggested solution There is a possibility for stack smashing in `get_some_input` and `main`. Check the boundaries of buffers, never let scanf(3) read more data than the buffer can hold.

There is an integer overflow in `make_full_name`. A size can never be negative, hence don't use signed integers. Use the `size_t` data type, which is defined as unsigned. This way we can only overflow using twice the data. Another way is to check if we expect an overflow to occur, e.g. if both sizes are larger than half of the maximum value.

```c
#include <stdio.h>

int
get_some_input( void )
{
  char buffer[128];

  printf( "Please enter the key: " );
  scanf( "%s", buffer );

  /* process input */

  return 0;
}

void
make_full_name( char *dst, int dstlen,
                const char *src, int srclen,
                int maxsize )
{
  if ( dstlen + srclen + 1 >= maxsize )
    return -1;

  strncat( dst, " ", 1 );
  return strncat( dst, src, srclen );
}

int
main( int argc, char **argv )
{
  char first[256];
  char last[256];

  printf( "Please enter your first name: " );
  scanf( "%s", first );
  printf( "Please enter you last name: " );
  scanf( "%s", last );

  make_full_name( first, strlen( first ),
                  last, strlen( last ), 256 );

  if ( get_some_input() < 0 )
    return -1;

  return 0;
}
```

Listing 1: Some vulnerable C code.