Course material is not allowed during the exam. One A4 sheet (two pages) with handwritten notes allowed.

Try to keep your answers precise and short. You will not get extra points by giving very long answers or by writing down what you know instead of what is asked. Just answer the question that is asked. If your answer does not fit the space provided, it's too long (may not count all points).

Take 10 seconds to think about what you are going to write before writing it.

The points shown in the above table are not final and coefficients can/will be applied to harmonize the grades.
1. Race conditions. Check which of the following statements is true or not (overall question points between 0 and 5 pts. Each line counts: 0 if not answered, 1 if correct, -1 if incorrect):

<table>
<thead>
<tr>
<th>Statement</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Race conditions are only possible on file systems.</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Exploits that abuse race conditions cannot be made reliable.</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Race conditions always abuse timing effects.</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Computational complexity attacks must be used to succeed in race condition attacks.</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>TOCTOU attacks are not race conditions but a different kind of problem.</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

2. Rainbow tables.
   2.1. What is a rainbow table? Explain in one sentence how a rainbow table works.
   Example good answer:
   Indexing facility to reverse hashes. Rainbow tables contain a list of pairs of values: a start value and an end value. Start values are taken "randomly" and end values are computed from start values by recursively applying N times the hashing function and the hash to value function to it (N being the depth of the rainbow table). When someone tries to reverse a hash, he/she applies up to N times the hash function to its value and for each result, looks for it among the end values. If it is found, it means the requested hash is one of the successors of the corresponding start value.

   Great if the following is mentioned
   - Time memory trade off
   - Reversing hashes
   - brief explanation of how it works

   2.2. What makes it better when compared to simpler techniques? What are its drawbacks?

   Much more time-efficient than trying to bruteforce the hash function, much more space-efficient than keeping a dictionary of the hash function. Still takes a lot of space. Depth can be used to find a compromise between space- and time- efficiency.

   2.3. What is the most (or one of the most) common application of a rainbow table?

   Cracking passwords

   3.1. What is the main difference between command injection and SQL injection?

   SQL injections are used to get access to a database (typically from a server) while command injections are used to execute commands on the underlying operating system.

   3.2. What is common to both of them?
Both exploit the fact that what the developer assumed to be data could also be instructions. Interpretation of parameters as commands in the query.

3.3. Mention one counter measure that is common to both.

Input filtering, e.g., white listing...

3.4. Mention one countermeasure which is specific to SQL injections.

Possible answers:
- Prepared statements
- PLSQL

4. What does blind SQL injection make possible, give an example.

Blind SQL injections allow to exploit an SQL injection vulnerability while no debug information is available to the user.

For instance if "?user=dummy" gives a rightful page and "?user=dummy' and SUBSTRING(secret,1,1) = 'a" does not, first char of secret is 'a'

5. URL filtering in kernel land: what could possibly go wrong?
   Read the advisory in appendix I (you don't have to read everything or understand everything from it to be able to answer this but you will need to be able to extract essential information).
   5.1. Put a cross in the box in front of each of the Saltzer and Schroeder principles that have been breached in this design:

<table>
<thead>
<tr>
<th></th>
<th>Economy of mechanism</th>
<th>This is a feature that is not required and could be done in a simpler way</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fail-safe defaults</td>
<td>No problem here, it's not active by default</td>
</tr>
<tr>
<td>x</td>
<td>Complete mediation</td>
<td>Don't see a problem here, all http packets checked? Well HTTPS isn't (can't be seen from the kernel here, so incomplete mediation?)</td>
</tr>
<tr>
<td></td>
<td>Open design</td>
<td>Was not well known or documented, but open source so not really hidden</td>
</tr>
<tr>
<td>x</td>
<td>Separation of privilege</td>
<td>URL filtering component in the kernel so no filtering here Disputable as there is rather good access control permissions to use this</td>
</tr>
<tr>
<td>x</td>
<td>Least privilege</td>
<td>URL filtering with maximum privilege</td>
</tr>
<tr>
<td>x</td>
<td>Least common mechanism</td>
<td>Common mechanism with the kernel</td>
</tr>
<tr>
<td>X</td>
<td>Psychological acceptability</td>
<td>Gives an app all access to all URLs? Bad idea. Privacy/ethical problem</td>
</tr>
</tbody>
</table>
5.2. To which class of vulnerability does this vulnerability belong to?

Null pointer dereference

5.3. Describe the steps generally used in exploiting such a vulnerability.

Several steps:
- Map an address at page 0 from userspace,
- write there some data that will be used/executed by the kernel
- trigger the vulnerability by calling a syscall which will use that data.

5.4. Is this kind of vulnerability only exploitable in the kernel? Why?

Somewhat yes, in userspace null pointer dereference is possible if the page is mapped, but it's not clear how this could lead to an elevation of privilege.

5.5. Which countermeasures exist to prevent exploitability of such vulnerability?

Recent Linux kernels forbid access to memory below a given address (includes null) but do not prevent dereference of corrupted ptrs. PAX UDREF, SMAP/SMEP prevents careless dereference or execution of user space memory.

5.6. Do you think this vulnerability is exploitable in recent Linux kernels?

No: min map addr is default

6. Another kernel Bug.

The patch in appendix II is correcting a bug in the Linux kernel and was published in January 2015.

6.1. What is this patch doing?

Changing the type of a variable and the return type of a function from unsigned int to unsigned long.

6.2. Can you explain the root cause of the error.

Integer overflow

Used to get a random value for top of stack. Random value was left shifted depending on the size of the pages. On 32-bits architectures, addresses are coded on 32 bits and pages are smaller so shifting the random int would still leave enough random bits. However on 64-bits architectures PAGE_SHIFT is greater so most random bits would be shifted out, only leaving 32-12 = 20 bits of randomness.

6.3. What are the consequences of this bug.

Reduces the security of ASLR
6.4. Can you measure “how much” the security of the system was reduced because of this bug?

By using 64-bits values, all 22 bits of randomness are kept, which makes guessing the stack address 4 times harder.

**Appendix I**

Android Platform Security (Samsung kernel - URL filtering in kernel land)

URL filtering in kernel land: what could possibly go wrong?

- **Authors**: Roberto Paleari (@rpaleari) and Aristide Fattori (@joystick)
- **Notification date**: 20/10/2015
- **Release date**: 20/01/2016

While looking at the Samsung Android kernel, we stumbled into a custom module named secfilter. This module captured our attention right from its initialization routine, where some Netfilter hooks are promptly registered:

```c
static struct nf_queue_handler sec_url_queue_handler = {
    .name = SEC_MODULE_NAME,
    .outfn = sec_url_filter_slow
};
static struct nf_hook_ops sec_url_filter = {
    .hook = sec_url_filter_hook,
    .pf = PF_INET,
    .hooknum = NF_INET_LOCAL_OUT,
    .priority = NF_IP_PRI_FIRST
};
static struct nf_hook_ops sec_url_recv_filter = {
    .hook = sec_url_filter_recv_hook,
    .pf = PF_INET,
    .hooknum = NF_INET_LOCAL_IN,
    .priority = NF_IP_PRI_FIRST
};
...
if ((add_send_hook =nf_register_hook( &sec_url_filter)) <0) break;
if (nf_register_hook( &sec_url_recv_filter) <0) break;
...
```

These hooks are configured to intercept all incoming (NF_INET_LOCAL_IN) and outgoing (NF_INET_LOCAL_OUT) packets. After being intercepted, outgoing TCP packets that match certain criteria eventually reach an internal function named getURL() which (surprise!) checks whether the TCP data resembles an HTTP request, and parses the contained URL.

So why Samsung put an URL parser in the Android kernel? Well, this is part of a mechanism accessible from a user-space application. Possible use cases include parental control apps, which need to implement system-wide URL filtering.

**But how it works?**

Netfilter hooks are always present, but the URL filtering mechanism is enabled only after a proper ioctl() request issued to user-space device /dev/url/. On our test phones, this device is accessible members of the secnetfilter group:
In turn, access to the secnetfilter group is granted to applications via the com.sec.android.SAMSUNG_GET_URL permission, created with protection level signature. An App with this permission can then configure the device to enable different monitoring modes:

- **FILTER_MODE_ON_BLOCK**: standard blocking mode, URLs are checked on outgoing packets and eventually blocked.
- **FILTER_MODE_ON_RESPONSE**: alternative blocking mode, URLs are extracted from outgoing packets, but responses are blocked and replaced with a custom HTTP/404 message.
- **FILTER_MODE_ON_BLOCKREFER**: similar to **FILTER_MODE_ON_BLOCK**, but checks the Referer header rather than the requested URL.
- **FILTER_MODE_ON_RESPONSEREFER**: similar to **FILTER_MODE_ON_RESPONSE**, but checks the Referer header rather than the requested URL.

If any of the blocking modes is enabled, then each visited URL is put in a queue and blocked until the App unblocks it.

To wrap up, an App that wants to use the URL filtering feature should implement the following logic:

1. First, the App enables one of the blocking modes available, for example by writing the following binary data to /dev/url:
   ```
   <2-byte ver (0x00)><4-byte SET_FILTER_MODE cmd (0x01)><4-byte mode>
   ```

2. Then, the App polls on /dev/url until some data is available for reading. Read data has the following format:
   ```
   <4-byte header><4-byte blocked URL ID><4 irrelevant bytes><4-byte URL length><n-byte URL>
   ```

3. Finally, the App checks the URL, takes a decision according to some application-specific logics, and communicates the decision to the driver, writing the following bytes on /dev/url:
   ```
   <2-byte ver (0x00)><4-byte SET_USER_SELECT cmd (0x01)><4-byte URL ID><4-byte choice (0x64 for block, 0x65 for allow)>
   ```

The kernel module reads integers from the buffer through direct dereferences (e.g., `filterMode = *(int *)data`), so data must be in little-endian format. When the kernel module receives a SET USER_SELECT command, it checks if the received ID corresponds to a blocked request (i.e., to a blocked TCP flow) and acts on it, according to the App decision.

We currently identified only a single application which uses this permission, namely com.symantec.familysafety, thus the whole URL filtering mechanism is probably not so widespread, or maybe it is still at an early stage of its development.

**Security implications**

As discussed before, it is true that only applications signed by Samsung (and installed exclusively through the Samsung App store) can obtain the SAMSUNG_GET_URL permission, yet this kernel module should raise the concerns of more than one privacy-aware Samsung users. It can be certainly wrapped in a "child protection" coat, but it still is a module that allows an "authorized" App to monitor any visited URL.

Anyway, besides ethical and privacy-related considerations, this module is quite worrisome even from a merely technical perspective. Indeed, after the URL filtering mechanism has been activated by an "authorized" App, any user-space application can then trigger a NULL pointer dereference in the HTTP parsing code, and crash the system.

To trigger the bug, it is simply enough to issue an HTTP request with no URL, such as (see this file):

```
GET HTTP/1.1
Host: www.google.com
```

The actual destination of such a request is irrelevant, as all the outgoing HTTP traffic is inspected when URL filtering is enabled.

Technically speaking, Samsung releases the source code of its Android kernels through its "Open Source Release Center". For the NULL pointer dereference outlined above, the vulnerable code lies in drivers/secfilter/urlparser.c, function `getURL()`. In a nutshell, the module extracts the HTTP request path through the following call:

```
findStringByTag(node, &(node->url), dataStart, " HTTP/");
```

Where `dataStart` is a char string pointing right after the HTTP request method token. However, when `findStringByTag()` processes the malformed request above it finds a zero-length string for the HTTP path, and leaves `node->url` uninitialized. Few lines after, `node->url` is dereferenced while still NULL.

**Affected devices**

We confirm this issue affects the following device models. Other models and firmware versions are probably affected as well, but they were not tested.
• SM-N9005, build N9005XXUGBOB6 (Note 3)
• SM-G920F, build G920FXXU2COH2 (Galaxy S6)
Appendix II

Note that on x86_64:

- STACK_RND_MASK has value 0x3fffff (22 bits long)
- PAGE_SHIFT has value 12
- unsigned int is 32 bits
- unsigned long is 64 bits
- -1UL = 0xffffffffffffffff
- -1U = 0xffffffff

Diffstat

```
2 files changed, 6 insertions, 5 deletions
```

```diff
diff --git a/arch/x86/mm/mmap.c b/arch/x86/mm/mmap.c
--- a/arch/x86/mm/mmap.c
+++ b/arch/x86/mm/mmap.c
@@ -35,12 +35,12 @@ struct va_alignment __read_mostly va_align = {
   .flags = -1,
   }
-  static unsigned int stack_maxrandom_size(void)
+  static unsigned long stack_maxrandom_size(void)
  {
-    unsigned int max = 0;
+    unsigned long max = 0;
    if ((current->flags & PF_RANDOMIZE) &&
      !(current->personality & ADDR_NO_RANDOMIZE)) {
-      max = ((-1U) & STACK_RND_MASK) << PAGE_SHIFT;
+      max = ((-1UL) & STACK_RND_MASK) << PAGE_SHIFT;
      return max;
    }
```

```diff
diff --git a/fs/binfmt_elf.c b/fs/binfmt_elf.c
--- a/fs/binfmt_elf.c
+++ b/fs/binfmt_elf.c
@@ -645,11 +645,12 @@ out:
  static unsigned long randomize_stack_top(unsigned long stack_top)
  {
-    unsigned int random_variable = 0;
+    unsigned long random_variable = 0;
    if ((current->flags & PF_RANDOMIZE) &&
      ! (current->personality & ADDR_NO_RANDOMIZE)) {
      random_variable = get_random_int() & STACK_RND_MASK;
+      random_variable = (unsigned long) get_random_int();
      random_variable &= STACK_RND_MASK;
      random_variable <<= PAGE_SHIFT;
    }
```

```c
#endif CONFIG_STACK_GROWSUP
```