Simon Foley,
Department of Computer Science, UCC

CS4615 Computer Systems Security
Mid-Term test 2014

STUDENT NAME	
STUDENT NUMBER	

A multilevel secure system has only one printer that is used to print jobs at all security levels. It is in a secured area and printouts are carefully labelled. A multilevel secure print queue manager accepts requests from subjects at any security level. Its operations are:

- `lpr <filename>`. Assign job number and add file to print queue. Returns `job#` to requester.
- `lprm <job#>`. Remove specified print job. Returns `success` or `failure`.

1. Provide suitable algorithms that describe the behaviour of the above operations taking care to ensure that multilevel security is preserved. For the sake of simplicity you should assume that all print-job data is stored in a simple dictionary lookup table, whereby the `job#` uniquely identifies the file to be printed, along with any other information; it is not necessary to consider printer controls/scheduling. (*3 marks*)

A `JOB lpr(FILE f, USR s);` *A user process* `s` *invokes* `JOB lpr(f,s)` *to add file* `f` *to the print queue; the assigned job number is returned. Note that the security level of the print job is determined by the level of the requesting user* `slev(s)` *and not the level of the file.*

```
JOB lpr(FILE f, USR s){
  j:= new job number;
  if (file f exists
     and flev(f)<=slev(s)) then {
     add dictionary entry j -> (f, slev(s));
    return(j);
  }else
    return(null)
}/*lpr*/
```

`Boolean lprm(JOB j, USR s)`. *A user process* `s` *invokes* `lprm(j,s)` *to remove job* `j` *from the print queue. Value* `True` *is returned if the deletion was successful, otherwise* `False` *is returned.*

```
Boolean lprm(JOB j, USR s){
  if job j mapping to (f,l) in dictionary
     and l=slev(s) then {
    remove job j from dictionary;
```

```
    return(true);
  }else
    return(false);
}/*lprm*/
```

A

2. Suppose that the value for the next `job#` identifier is determined as a one-way hash of the previous `job#` value plus one. Give an example of a covert channel in the queue manager that permits a Trojan Horse signal one bit of high-level data to a low-level subject and suggest how it should be eliminated. *(3 marks)*

A *Let* $job\# = h(previousjob\# + 1)$ *(or, alternatively,* $job\# = h(previousjob\#) + 1$*; either interpretation makes no difference to the answer).*

- *Suppose an (unclassified) malicious user submits a print job on Monday, which is assigned job number denoted* $jobMon\#$*.*
- *Trojan Horse (running at secret) accesses secret data on Tuesday and wants to transmit a single bit of (secret) data.*
  - *If the value is 0 the Trojan Horse does nothing,*
  - *if the value is 1 the Trojan Horse submits a (secret) print job and is assigned job number* $jobTue\# = h(jobMon\# + 1)$*.*
- *Malicious user submits another (unclassified) printjob on Wednesday and is assigned a job number denoted* $jobWed\#$
  - *If the value of* $jobWed\#$ *is* $h(jobMon\# + 1)$ *then the user deduces that no job was submitted Tuesday (by the Trojan) and therefore the secret value is 0.*
  - *If the value of* $jobWed\#$ *is* $h(h(jobMon\# + 1)+)$ *then the user deduces that a job was submitted Tuesday (by the Trojan) and therefore the secret value is 1.*

There's two possible ways to eliminate the Trojan Horse.

- *Use a secure random number generator to generate the job number: a user cannot infer what future (or past) job numbers might be based on the currently assigned job number.*
- *Have a different job number counter for each security classification. For example, at top-secret have a counter that stores the job id of the previous top-secret print job, and at secret have a different counter that stores the job id of the previous secret print job, etc. In this case all a low-user can see is the job number of the last print job that was requested at low and cannot infer anything about the values of the high-level job counters.*

A

STUDENT NAME	
STUDENT NUMBER	

3. Suppose that the print queue manager is hosted on a dedicated server that accepts print requests over the network. It is discovered that requests over a certain size result in a buffer overflow in the manager software. Assuming that you have a suitable request that can exploit this vulnerability, explain how it would be used to compromise the system. (*2 marks*)

A *The print queue manager accepts requests over the network from users and we assume that its software was designed in such as way that it upheld the BLP axioms (as in the answer to Question 1). The user does not have any other access to the system that hosts the print queue manager. We assume that the attack string contains instructions for executing some user-selected command: these instructions are executed as a consequence of the buffer overflow. An example of the user-command might be to get the print queue manager to copy a top-secret job and enter it as an unclassified job in its print queue table/data-structure.*

We assume that the attack string can be provided as part of the user request that is sent to the print queue manager. When the print queue manager receives the attack string/user request over the network, it causes the print queue to execute the user-supplied command and behave in a way that breaks its expected behavior (violating the BLP axioms) and therefore breaks the security of the print queue manager system. **A**

4. Suppose that you have available a high-assurance multi-level secure system. A separate copy of the print queue manager is run for each security classification. Will this have any impact on the effectiveness of the buffer overflow exploit? Explain your answer. (*2 marks*)

A *In Question (1), the queue manager manages a queue of print jobs at different security levels. This means that it must be trusted to enforce MLS properly, and this is why we needed to show that it was compliant to the Bell LaPadula model for MLS.*

In this question (a variation of which was discussed in a Tutorial) the suggestion is that we have a separate print queue manager for each security classification. This means that each print queue manager just manages jobs at its security level. For example, the unclassified print queue manager manages a single-level queue of unclassified print jobs; the secret print queue manager manages a single-level queue of secret print jobs, and so forth.

If I had an existing MLS system then each print queue manager would just be a conventional subject/process (running at its corresponding security classification) and with its own print queue (an object) at the same level. The MLS system will enforce security and ensure, for example, that a unclassified user cannot read/submit a secret file to the unclassified print subject, or that a secret user cannot submit a secret file to a an unclassified print queue. In effect each print queue manager is put in its own sandbox where the MLS axioms are applied on all access.

If the print queue manager had a buffer overflow vulnerability then the damage is confined to the 'sandbox' of its process: all accesses (by the attacking code) are subject to the BLP axioms and therefore the buffer overflow is not effective (in that we cannot use it to bypass MLS).

$\overline{\mathbf{A}}$