OLLSCOIL NA hÉIREANN
THE NATIONAL UNIVERSITY OF IRELAND, CORK

COLÁISTE NA hOLLSCOILE, CORCAIGH
UNIVERSITY COLLEGE, CORK

Summer Examination 2013

CS4615: Computer Systems Security

Professor I. Gent,
Professor B. O'Sullivan,
Dr. S.N. Foley

Answer *all* questions

1.5 Hours

A *Note to extern. This module is 5 ECTS credits. The exam paper is graded out of 80 marks with 20 marks for Continuous Assessment (per Marks and Standards).*

Questions are either: straightforward regurgitation of material; a reasonably familiar problem that requires application of knowledge, or intended to stretch the student with more challenging/unfamiliar problems. The intention is that a student who can regurgitate material can pass; a student who not only 'knows' the material but can apply it in straightforward ways can achieve a second class honours student. A first class honours fits the two previous categories and can apply the knowledge in more challenging ways to trickier and unfamiliar problems. $\overline{\textbf{A}}$

1. a) Give an example of an *Attack Tree* and outline its use. (*6 marks*)

A *Any sensible attack tree illustrating AND/OR decomposition of threats.* $\overline{\textbf{A}}$

b) Write a Java security policy rule that permits Bob's applet read and write access to files in `file:/usr/home/alice/data`. Note any assumptions you make. (*6 marks*)

A *If we assume that* `Bob` *is an alias for Bob's public key in (Alice's) keystore then*

```
grant signedby "Bob" {
permission java.io.FilePermission("/usr/home/alice/data/*","read,write")
}
```

Alternatively, we could specify a rule referencing a (signed) code source for Bob's applet (with or without signing). The exact Java syntax of the rule is not important, rather its ingredients: granting a (signed) code source a permission defined in terms of a target and action. $\overline{\textbf{A}}$

c) Using a simple example of a policy rule, illustrate how the *Java Authentication and Authorization Service* (JAAS) extends Java's code-centric security policy model. (*6 marks*)

A *Java's code-centric policy model defines security in terms of the permissions that (mobile) code has on objects: for example, the* `grant` *entry in the previous example.*

```
grant signedby "Bob" codebase "http://www.bob.com/code.jar"
{permission java.io.FilePermission("/usr/home/alice/data/*","read,write")
}
```

JAAS extends this by defining security in terms of the permissions that subjects have when executing (mobile) code to access an object. For example, if Clare is a Kerberos authenticated subject then

```
grant signedby "Bob" codebase "http://www.bob.com/code.jar"
Principal javax.security.auth.kerberos.KerberosPrincipal Clare@ucc.ie
{permission java.io.FilePermission("/usr/home/alice/data/*","read,write")
}
```

$\overline{\textbf{A}}$

d) Discuss security compliance audit. Use the PCI-DSS Requirement 3.4 *"Render Primary Account Number unreadable anywhere it is stored [...]"* to illustrate your answer. (*6 marks*)

A *Answer should demonstrate understanding of security controls (such as 3.4), that they have corresponding testing procedures that test control efficacy (eg grep PANs) and that audit is the regular activity of checking the presence of the controls, the procedure tests and the efficacy reports; this information is used in determining the Merchant levels in PCI-DSS and this determines their terms and conditions.* $\overline{\textbf{A}}$

e) If security state matrix $M[s, o]$ defines the access that subject s has on object o then define the Bell and LaPadula axioms for multilevel security. (*6 marks*)

A *No write down: $w \in M[s, o] \Rightarrow level(s) \leq level(o)$; No read up $r \in M[s, o] \Rightarrow level(o) \leq level(s)$, where security classifications are ordered according to $\leq$, level() gives the current security classification of a subject/object and Tranquility requires that these levels may not change with state transition.* $\overline{\textbf{A}}$

(*Total 30 marks*)

2. A programmer implements *Terry's Game* in C on Unix. Player high-scores are stored in the file `/etc/highscores`. The game is SUID root and is executable by all, the highscores file is owned by root and is readable and writable only by its owner.

   a) Why is the game SUID root? What are the dangers of running the game as SUID root? Using a sample Domain Definition Table, explain why Type Enforcement (used in SELinux) can provide better protection. (*15 marks*)

   A *(Terry's Game is an application used in lectures to demonstrate secure Java programming concepts.) Making it SUID ensures that the scores file can be updated when the game is run, but not otherwise. Ensures that players cannot cheat by directly manipulating the scores file. Looking for straightforward description of SUID (root), principle of least privilege and the potential threats (eg buffer overflow attack by player).*

   We have DDT[games,scores]=RW and all high-scores related files are type scores and executing a games program causes process to enter domain `games`. *The DDT does not permit access to* `scores` *files from any other domain. In this case, if a program fails in the* `games` *domain then its behavior is confined to that domain and it cannot access any other resources. This is unlike a setuid-root program.* $\overline{\textbf{A}}$

   b) The SUID root program described Part (a) above may take as parameter a path to a scores file (to override default `/etc/highscores` path). The program has behaviour:

```
  void main (int argc, char* argv[]){    // the terryG program
        // argv[0] gives path to scores file
     ...// Step 0. play Terry's game and on completion,
     ...// Step 1. open scores file to obtain this player's last score;
     ...// Step 2. create/open temporary file stmp in same directory as scores;
     ...// Step 3. open scores file, copy contents to stmp and current score;
     ...// Step 4. close files and rename stmp as score file;
}
```

   Explain how a player can carry out a race/TOCTOU attack on this program. (*10 marks*)

   A *Steps 1,2,3,4 of the program are vulnerable to a race TOCTOU attack as follows. We assume that the attacker has the ability to track the execution steps of the* `terryG` *game.*

   The race attack follows a methodology similar to that used on an early version of `/bin/passwd` *and is as follows. Suppose an attacker wants to modify root's* `.rhosts` *file. In his own directory he creates a directory* `xxx` *containing* `.rhosts` *with the desired entries. In* `xxx` *he creates a symbolic link file* `link` *to* `xxx/.rhosts` *and invokes* `terryG xxx/link`. *After Step 1 and before Step 2, the attacker changes* `link` *to point to root* `/.rhosts`*; Step 2 results in* `stmp` *created/opened in* `/`. *After Step 2 and before Step 3, the attacker changes* `link` *to point back to* `xxx/.rhosts`*; Step 3 results in the contents of* `.rhosts` *to be copied to* `/stmp`. *After Step 3 and before Step 4, the attacker changes* `link` *to point to root directory* `/.rhosts`*; Step 4 renames* `/stmp` *as* `/.rhosts` *(link) and the attack is complete. A similiar attack strategy could be used to modify the* `/etc/highscores` *file.* $\overline{\textbf{A}}$

(*Total 25 marks*)

3. a) In order to defend against Denial of Service, the server hosting Terry's Game runs a modified three-way handshake: a connection request results in a response containing SYN cookie $h(ip_s, ip_d)$, given source IP ip_s, destination IP ip_d and one-way hash function $h()$.

Explain how a SYN cookie mechanism operates, outline its advantages and note any vulnerabilities in the design of the mechanism above. (*15 marks*)

A *The cookie replaces the conventional serial number used in the three way handshake. It acts as a cryptographic checksum for the subsequent ACK from the source. In step 2 of the handshake, the server responds with* $SYN(cookie); ACK(serialnumberfromsource + 1)$ *and forgets the state associated with the request (does not add it to the half open connection table). In step 3, the requester effectively re-presents the half open connection state along with* $ACK(cookie + 1)$ *and the server uses the cookie to confirm the integrity of the information.*

The advantage of the cookie mechanism is that it makes the server-side stateless during the three way handshake (no half-open connection table to maintain). Therefore its not possible to carry out a SYN flood attack on the table/server. However, it does not capture the entire state of an entry in the half-open connection table and cannot support TCP options which may lead to performance issues.

The proposed implementation has a number of problems. Firstly, it does not provide a checksum on the minimal information required for the connection and cannot distinguish cookies for different source/destination ports. Secondly, the cookie can be easily forged since its not based on a secret. As a result, the attacker can, for example, forge a cookie for someone else's IP address and carry out an IP spoofing attack. The cookie should be at least, $h_K(ip_s, ip_d, port_s, port_d, t)$ *where K is a server secret and t is a counter value which helps prevent forgery and means that the cookie value also provides a sequence number.* $\overline{\textbf{A}}$

b) Suppose that Terry's Game is hosted at IP 192.168.1.1 on Port 666 and a firewall is used to provide host access control. The firewall is configured with policy:

index	SrcIP	DstIP	SrcPort	DstPort	Action
1	192.168.2.*	192.168.1.1	*	666	drop
2	192.168.*.*	192.168.1.1	*	666	drop
3	192.*.*.*	192.168.1.1	*	666	allow
4	192.168.2.20	192.168.1.1	*	666	allow
5	*.*.*.*	192.168.1.*	*	666	drop

Identify any anomalies in the policy and discuss how they might be eliminated. (*10 marks*)

A *Rule 1: redundant to 2 (but not 5); Rule 3: generalized with 1, 2 Rule 2: Rule 4: redundant to 3, shadowed by 1 and 2;*

While redundant rules can be eliminated, shadowing and generalization require administrator decision. For example, perhaps the intended policy was:

index	SrcIP	DstIP	SrcPort	DstPort	Action
3	192.168.2.*	192.168.1.1	*	666	allow
5	*.*.*.*	*	*	*	drop

$\overline{\textbf{A}}$

(*Total 25 marks*)