CSE543/Fall 2007 - Midterm

Thursday, November 1, 2007 — Professor Trent Jaeger

Please read the instructions and questions carefully. You will be graded for clarity and correctness. You have 75 minutes to complete this exam, so focus on those questions whose subject matter you know well. Write legibly and check your answers before handing it in.

Short Answer (Answer 12 of 14) - some will be one or two words – no more than 3 sentences

1. (*3pts*) What is the difference between *protection* and *security*?

   answer: A system that provides security ensures the protection of its data (i.e., enforcement of its security goals) even when a user may run code that has malicious intent. Systems that provide protection enforce the specified policy only if the user runs trusted code.

2. (*3pts*) Define *protection system*.

   answer: A protection system consists of a *protection state* describing the current access policy, a *reference monitor* to enforce the protection state, and *administrative operations* to modify the protection state.

3. (*3pts*) How can you configure the Windows access control model to ensure that a particular subject only has access to one file?

   answer: Restricted context is easiest. Add a negative ACE at the beginning of all other object ACLs, except the one being permitted. Grant rights for that one.

4. (*3pts*) What are the *guarantees* that a UNIX sandbox (e.g., Janus) must provide in order to ensure that the process cannot escape the limited permissions defined?

   answer: Basically, reference monitor guarantees. It must provide a *tamperproof* implementation that enforces a mandatory access control policy (also tamperproof) that *mediates* all security sensitive operations, and the policy must ensure that the process does not gain unauthorized access (i.e., *verifiably* enforce security goals).

5. (*3pts*) What is the *confused deputy problem*?

   answer: That a multi-client server may be spoofed into granting one client unauthorized rights to another client's objects because it must have the permissions for all the clients to run in an ACL system.

6. (*3pts*) What mechanisms does Multics use to protect the *integrity* of objects?

   answer: Protection rings provide the mediation points for enforcing integrity. The access and call bracket policies describe the integrity policy of a Multics system. A tamperproof kernel enforces the policy.

7. (*3pts*) Define the two fundamental properties of the Bell-LaPadula model (i.e., multilevel security model).

   answer: Simple-security property – no read up, and $\star$-security property – no write down.

8. (*3pts*) What does the *extend* operation of the Trusted Platform Module do (be precise)?

   answer: It extends a hash value in a particular TPM register (PCR) by taking the current register value and hashing it concatenated with the input value to the extend operation.

9. (*3pts*) Why is it more secure to implement a *reference monitor* inside the kernel (as in LSM), rather than to use *system call interposition* (as in Janus)?

   answer: Time-of-check-to-time-of-use attacks. In interposition, the mapping between names and actual objects is computed separately from the kernel, so it may be possible for a concurrent process to change the mapping (e.g., changing a local file to be a link to `/etc/shadow`). That is, the label-file mapping is not tamperproof in Janus.

10. (*3pts*) What are *buffer overflow*, *heap overflow*, and *integer overflow* vulnerabilities?

    answer: A buffer overflow occurs on the stack by overwriting the return address. A heap overflow occurs on the heap by overwriting a key pointer, such as a function pointer. An integer overflow occurs on signed integers, when the maximum value is reached, computer integer operations differ from true integer operations.

11. (*3pts*) When a cryptographic construction provides message *non-repudiation*, what can the receiver prove?

    answer: The receiver can prove that the message originated from a single principal, the holder of the associated private key. This can even be proven to third parties.

12. (*3pts*) What impact does the *birthday paradox* have on the security of a 90-bit hash function?

    answer: Due to the *birthday paradox*, the probability of finding a collision in a 90 bit hash is only one in 2^{45}.

13. (*3pts*) How does the use of a Kerberos *authenticator* replace function of the last two messages in the Needham-Schroeder symmetric key protocol?

    answer: An authenticator aims to prove that Alice's message with the corresponding ticket (and the new session key) has been freshly created by Alice by using a *timestamp*. This replaces the need for a challenge-response provided in the last two messages. (Bob still proves ownership of the corresponding TGS-Bob key through use of the session key, but this is done later – not required in the answer).

14. (*3pts*) Identify the conditions when you would want to add an HMAC to a secure communication.

    answer: HMAC is used to justify the *integrity* and *authenticity* of a message under a *shared key*. Should a shared key be available, already distributed, and should there be integrity requirements on the message, you almost always want to HMAC. A rare exception is when the message has a well-known format, so that the receiver could check based on the content of the message – Kerberos is an example.

Long Answer - no more than 3 paragraphs

15. (*7pts*) Specify how domain transitions occur in UNIX, SELinux, and Multics. Just outline the mechanisms – no specific rules are required. Indicate the security advantages of SELinux and Multics over UNIX domain transitions in your description.

    answer: UNIX transitions domains via `setuid`. The UID of the process changes to that of the owner of the file.

    SELinux defines rules to limit when domain transitions are permitted and what the destination domain will be. These rules constrain who can cause a transition (not every invocation gains privilege) and limits the privileges based on the caller (different callers get different privileges).

    Multics defines domain transitions via call brackets that state when low privileged processes may transition to higher and vice versa. Multics also defines gatekeepers to ensure that higher privileged code cannot be compromised by low integrity inputs. Multics defines multiple ring levels (not just `root`) and protects inputs.

16. (*7pts*) What are the components that ensure integrity in a Clark-Wilson integrity system and a Biba integrity system (i.e., there are two different sets to be specified)? How do these components ensure integrity in each system?

    answer: Using Clark-Wilson, we need to add Integrity Verification Procedures to test integrity at the outset, and verified Transformation Procedures to handle the high integrity data. TPs must be assured to operate on high integrity data correctly and protect themselves from low integrity inputs.

    In Biba, we only add integrity guards to protect normal processes from low integrity inputs, and we can proceed as long as no low integrity inputs reach the high integrity processes, defined by the policy.

17. (*7pts*) Why is it necessary to prevent forgery of capabilities in capability systems to meet reference monitor guarantees? Specify the conditions under which it is necessary to *weaken* (reduce) the permissions available to a capability.

    answer: If a capability can be forged, then a process can create its own permissions to any object that it can name. This would circumvent the tamperproofing of system policies, and nothing would be verifiable.

    We must weaken a capability when a high secrecy subject fetches a capability from a low secrecy memory. Because this capability may have been created by a low secrecy subject, it may have write permission to low secrecy objects. Since this would violate the *-property, such capabilities must be weakened to remove the write permission.

18. (*7pts*) What purpose does the *ticket* in a Kerberos message serve? Why can't Mallory spoof Alice to Bob by generating an authenticator claiming to be from Alice $\{Alice, timestamp\}K_{session}$ and replaying a ticket for which she knows the session key from a previous session with Bob $\{Bob, Mallory, timestamp, lifetime, K_{session}\}K_{Bob-TGS}$?

    answer: A ticket is used to securely provide the session key to the server. It includes the names of the client and server for which the session key applies, a freshness timestamp, the ticket's lifetime, and the session key. Because the ticket is encrypted in a key known only to Bob and the TGS, and only the format is well-defined, the ticket securely conveys the key.

    If Mallory substitutes her ticket, Bob will be able to see that the ticket says that it is for a communication from Mallory to Bob, so he will know it is not from Alice. Plus, the timestamp may have expired, but that is not guaranteed.

Word Problems - take your time and answer clearly and completely.

19. (*10pts*) Suppose that Alice uses a capability system that uses cryptography to protect its capabilities from forgery. She has an object *obj* and *rw* (read and write) rights to that object.

The kernel builds capabilities for its users, and authorizes access by verifying the integrity and authenticity of the capabilities (generated by kernel only) and the checking the rights included in the capability correspond to those requested. The kernel has a symmetric key K and a private key pair K^- and K^+.

NOTE: Users must be able to see the target of the capability in order to use it in this system. That is, the capability data is not secret (unlike the notes).

(a) (2pts) Using the symmetric key cryptography, design a capability to give to Alice for *obj* with the rights specified above.

(b) (2pts) Using the public key cryptography, design a capability to give to Alice for *obj* with the rights specified above (equivalent function as (a)).

(c) (2pts) Suppose the kernel wants to track the distribution of capabilities in a manner that is secret to the users. For example, the kernel adds Alice's name to capabilities created for her, so that it can track how they are distributed. Using the symmetric key cryptography, design a capability to give to Alice for *obj* (as in (a)) that includes her name, in secret.

(d) (2pts) Using public key cryptography, generate a capability that meets the requirements of (c).

(e) (2pts) Suppose that the kernel has a mechanism to collect all capabilities to a specific object (i.e., one object at a time) and revoke those that no longer satisfy the current policy. The kernel stores a revocation time with each object that indicates the last time such a revocation was performed. Write a capability, using symmetric key cryptography, that the holder can prove is fresh since the last revocation for that object.

answer:

Suppose base capability is $C = \{obj, rw\}$.

(a) $C + HMAC(K, C)$
(b) $C + S(K^-, C)$
(c) $C + E(K, A) + HMAC(K, C + A)$
(d) $C + E(K^+, A) + S(K^-, C + A)$
(e) $C + timestamp + E(K, A) + HMAC(K, C + A + timestamp)$

20. (*10pts*) Answer the questions below regarding the security-typed, key generation code for Diffie-Hellman.

```
int {secret} key; /* global */

int{public} DHgen(int n, addr dest)
{
    int{public} p = 3, g = 4;
    int{secret} x = n;
    int{public} y, y';
    int z;

    y = ???; // (see part a)

    send(y, dest);
    recv(y', dest);

    z = ???; // (see part a)

    if (z <= 1)
      return -1;

    key = z;
    return 0;
}
```

(a) (2pts) Write the Diffie-Hellman equations to compute y and z.

(b) (2pts) Suppose n is 5 and y' is 2. What is your shared DH key?

(c) (2pts) From a security-typed language perspective, what is the problem with sending y to the destination in this code via `send(y, dest)`?

(d) (2pts) What is the inferred security type (label) of z?

(e) (2pts) If we add a declassifier for y, will this security-typed program compile? Explain briefly.

answer: (a) $y = g^x mod p$ and $z = y'^x mod p$
(b) 2
(c) y is computed from x which is secret. y's public label will be in error, so a declassifier is needed. Note that the problem of computing x from g^x is hard, but this needs to be made known to the labeling.
(d) secret
(e) No. The implicit flow from the conditional ($z \leq 1$) is not allowed either.

21. (*10pts*) Suppose that you are a Multics administrator (again). You need to configure a password program to enable users to modify their passwords. To change password data, we execute a *password executable file* from a *user's shell process* to modify a password file or shadow file (only the latter holds the actual password hashes – as in a UNIX system). A summary of the key files is below.

    - Password executable file
    - Password file: has user entries
    - Shadow file: has password hashes
    - User's shell process

    Suppose there are two secrecy levels: (1) system secret and (2) public. The former is more secret than the latter and used to protect data that is secret to the system.

    Answer the following questions.

    (a) (2pts) Select secrecy levels for the files and processes above.

    (b) (2pts) Select legal execution rings for the password process and user shell. Note: Kernel runs in ring 0 and administrators run in ring 2. Assume highest ring is 7.

    (c) (2pts) Select access brackets for the files above that protect the integrity of the files appropriately.

    (d) (2pts) Select the call bracket for the password executable to protect the integrity of the code appropriately.

(e) (2pts) If the password process runs at system secret, which files can it write?

answer:

(a) Only the shadow file has secrets

- Password executable file: public
- Password file: system secret in this case (not in UNIX)
- Shadow file: system secret
- User shell process: ambiguous so either is accepted

(b) Must be higher than the admin shell. Script must be higher than the web server.

- Password process: 1, 2, or 3
- User shell > password process and admin

(c) Suppose password process runs at 3

- Password executable file: passwd can read, admin can write – (2, 3)
- Password file: passwd can read and write, but no higher can read or write – (3, 3)
- Shadow file: passwd can read and write, but no higher can read or write – (3, 3)

(d)

- Password Executable: Anyone can execute – (3, 7)

(e) Shadow and password file.