אבטחת מידע: תאוריה בראי המציאות

מבחן מועד א׳ סמסטר א׳ תשע״ב
20 בפברואר 2012
מרצה: ערן טרומר

פתרון המבחן

Question 1

a. Amazon's VMM can read the memory of Achilles, including his decrypted data. The TPM offers no protection since it is (or could be) virtualized by Amazon's VMM. (TPM measurements are only meaningful if the full real boot sequence is protected). Another consideration is architectural side channels.
b. The spy can reboot the machine into malicious code of its choosing, reset the TPM by shorting its reset pin (now the PCRs are zero), extend the PCRs to the "correct" values of the Windows version Achilles installed, and then unseal the disk decryption key.

Question 2

a. Yes. Give the program the capability to access the image file, and nothing else. In particular, the program must not be able to talk to other processes and obtain additional capabilities from those; hence, such talking must be controlled by capabilities as well. (Most answers omitted this last part.)
b. We can try to create a new user account (subject) for running the program, and give this user access only to the image file. But this is difficult to enforce with Discretionary Access Control based on ACLs. For example, there could be some files in the system that are accessible by any user account according to their ACL; or other human users on the system may change ACLs on files they own to permit access to the new user. (With Mandatory Access Control, we could enforce the requisite permissions.)

Question 3

a. The Manifest should include:
    i. A declaration of a new READ_CONTACTS permission.
    ii. A component definition - a "Content Provider" or a Service
        - The component should require the READ_CONTACTS permission in order to IPC with it (if component is provider, it should be its "read permission").

    Note that the application does not require any special permission, because it is assumed to have the appropriate UID/GID to access the contacts database file.
b. This suggestion violates the principle of "least privilege": it means the application will have contact access when used just to access the SMS database, and vice versa; thus, a vulnerability will expose more information than necessary. Also, it would expose a broader attack surface because it offers more functionality.

A better suggestion would be to implement a second, independent application which only offers access to the SMS database.

Question 4

a. Start with the known initial state of S, and observe what values are swapped by the swap line in order to learn how it is modified. We then know the full content of S at the end of schedule.
b. Let cache_line by the size of a cache line in bytes, and let cache_sets be the number of cache sets. Accesses to memory addresses that are the same modulo cache_line*cache_sets are indistinguishable by the given side channels. Thus, it suffices to use a sparse representation of S, where values are stored cache_line*cache_sets bytes apart. The code is the same as the one in the question, except we malloc(256*cache_line*cache_sets) and every index into S is multiplied by cache_line*cache_sets.
   Alternative solution 1: always access all values of S. This is inefficient.
   Alternative solution 2: randomize the layout of S. This is has hard to do securely and efficiently, and it's hard to analyze its security.

Question 5

a. Generate a key pair with public key k_{pub}, and choose some new domain name *domain*. Run MD5COLLIDE(k_{pub},*domain*) to generate *z* which looks like a new domain name (if *z* doesn't look like a proper domain name, or is already registered, retry; the success probability is high enough). Register *domain* and send the request (k_{pub},*domain*) to get a certificate (k_{pub},*domain*,*sig*). Then (k_{pub},*z*,*sig*) is also a valid certificate.
b. Generate a key pair (k_{pri},ki_{pub}). Use MD5COLLIDE(k_{pub},"update.microsoft.com") to generate a new domain name *z* as before, and register the domain *z*. Send request pair (k_{pub},*z*) to get a certificate (k_{pub},*z*,*sig*). Then (k_{pub},"update.microsoft.com",*sig*) is a valid certificate for update.microsoft.com, and we know the corresponding private key, so we can impersonate Windows Update in SSL/TLS authentication.

Question 6

a. No. Bob can change the clock on the laptop (using either software or hardware), or clone the hard disk and run it under a virtual machine with a faster virtual clock. Software reverse-engineering can also be done pretty quickly.
   - The intention was that Bob has no Internet access, but since this was not stated explicitly, solutions that relied on network access were accepted.
   - TPM functionality is not sufficient – recall that TPMs are not resistant to hardware attacks, and Bob has the laptop in his hands.
b. Yes. The USB device can include an internal clock and a microcontroller that releases the message only after a day has elapsed. These can be protected by advanced hardware protection mechanisms, as discussed in class, which will take more than a day to break.
   - Some students said the USB device only stores data ("mass storage device"), but there was no reason to assume this.
c. No. As discussed in class, It is very difficult to create a hardware device that will withstand reverse-engineering and keep a secret for a whole year at the hands of an expert who can perform invasive attacks like FIBs. (Alice can't create a new device

within a day, unless she reuses existing designs, in which case Bob can practice on those, making his attack easier.)

d. Yes. Two solutions:
    i. Use fully-homomorphic encryption. Alice generates a key pair sends her encryption key and the encryption of her book name. Bob locally encrypts his book names and then homomorphically computes the Boolean circuit that compares Alice's book to his books and answers 0/1. This results in an encryption of 0 or of 1. Bob sends this encrypted result back to Alice, who decrypts it.
    This solution has very low communication, but is computationally expensive.
    ii. Bob hashes all of his book names, and sends the hashes to Alice. Alice hashes her book name, and checks if his hash value is among those that Bob sent. (For provable security, use a keyed hash function, where Alice chooses the key).
    This solution is very cheap computationally, but the communication is linear in the number of books in Bob's list; the saving is just because a sufficient hash value (say, 64 bits) is smaller than a typical book name.
        - In this solution, it's crucial that Bob sends his hashes to Alice. We cannot ask Alice to send the hash digest of her book name to Bob – see below.
    - Note that the question stresses the communication limitations, so we expected the first answer (but accepted the second too).

e. No. For any protocol that answers correctly, Bob can find out which book Alice is asking about, out of any list *L* (e.g., the Library of Congress catalog): Bob receives Alice's query, and then for each book *B* in *L*, builds a fake book list containing just *B*, performs the computation prescribed in the protocol, and observes the resulting answer.
    - Note that the homomorphic encryption solution doesn't work, since Bob can't decrypt the final result to get a yes/no answer. (FHE doesn't give any mechanism for this, and moreover, if this is possible then the encryption is insecure under the Semantic Security definition shown in class.)

Question 7

a. To get the key, query (`name=”%s”`, `weight=KEYADDR+1`). This causes `lessweight=KEYADDR`, and printf thinks that `lessweight` (which is next in the stack after the printf format string pointer) is its next parameter, so it prints the string that lessweight pints to

    - The answer `name=”%s%s%s...”` without specifying `weight` is incorrect for several reasons, the first of which is that it will try to print pretty-random memory addresses (those pointed to by the values on the stack), and is very likely to always crash by accessing some invalid address; so the attacker learns nothing.

    - Answers that put the address in name are also incorrect; think about where `%s` will look for the address in the stack.

- Answers that assume knowledge of stack or code addresses are incorrect, since it is stated that “the stack and code addresses are well-randomized using ASLR”. If the answer explictly stated that 32-bit x86 ASLR is weak; it got partial credit: ASLR is indeed weak in common 32-bit x86 implementations, but better implementations are certainly possible, and even the 16-bit entropy of Linux PaX mean the attacker will crash the server about 2^15 times before success (which is likely to be noticed and stopped).

b. Query (`name="QQQ...QQQ%n"`, *addr*+1) where *addr* is the address to be written, and the character `Q` appears *n* times, where *n* is the value to be written in the word starting at *addr*.

   - As before, you cannot put the address inside name.
   - Practically, not all 32-bit values can be written, since the length of number of `Q`'s would be too long. But writing arbitrary values wasn't required in the question, and is not needed for the subsequent attacks.

c. Use a variant of the the Differential Fault Analysis attack against unknown ciphers. Choose some legitimate query *Q*. For each *i* from *keylength* down to 1: request a signature for *Q* and call it S_i, then write the value 0 to address `KEYADDR`+*i-1*.
Now recover the key bytes one by one. Assuming knowledge of the first *i* key bytes $K_0 \ldots K_{i-1}$, we can find byte K_i by trying all 256 hypotheses for K_i : sign by yourself the query Q using the key $K_0 \ldots K_{i-1} K_i 0 \ldots 0$ and check if this matches the stored value S_{i+1}. (We assume deterministic signatures. If there are multiple matching hypotheses, we can do backtracking; in typical cases this is very unlikely.)

   - Another answer was to attack each key byte (or dword) at a time, making 2^8 (or 2^32) queries to see which value yields the correct signature. This generates many queries to the web service, making the attack slower and more likely to be caught, so this answer got partial credit.
   - There was a common mistake of assuming %n writes a single byte rather than a 32-bit dword (no points were subtracted for this).

d. Assuming the RSA signing uses the Chinese Remainder Theorem implementation, we can conduct a fault attack as shown in class. The secret key at `KEYADDR` will contain the primes *P* and *Q*. First get an RSA signature *S* on some message. Then corrupt *P* but not *Q*, and ask for another signature *S*' on the same message. Then S is congruent to S' modulo Q but not modulo P, hence $\gcd(N, S-S') = Q$.