COM 402 exercises 2023, session 10: Trusted Hardware and Trusted Computing

Exercise 10.1

In the following scenarios explain whether you need isolation, attestation, or both. Explain what would happen if these properties were not met.

1. A city government uses an HSM to sign updates to the population census.
2. An IoT sensor signs a message authenticating itself to the central server that collects measurements.
3. We use a TPM to ensure secure booting of a server located in an isolated room in the basement of a bank.
4. Intel SGX is used to perform biometric access control on a backend server in the cloud.

Exercise 10.2

In question 10.1(1) what would be a better security practice to avoid fraudulent transactions: giving the mayor of the city the right to execute the signature in the HSM, or have the HSM require the credentials of two census civil servants to permit the operation (justify your answer).

Exercise 10.3

Alice runs a certificate authority, and needs a secure machine to store the CA keys and sign certificate requests. What are the challenges, advantages or disadvantages if Alice runs their signing infrastructure:

1. General purpose CPU
2. Enclave running on commercial processor
3. Dedicated hardware security module (HSM)

Exercise 10.4

- Recall that in a secure boot supported by a TPM, TPM_Startup(ST_CLEAR) is run to reset the PCRs. What would go wrong if TPM_Startup(ST_CLEAR) could be called at any time after boot?

Exercise 10.5

- During attestation of software, what could go wrong if the following steps of the protocol do not happen:
  1. The challenger does not send a Challenge (also known as Nonce)
  2. Application and challenger perform a key exchange to establish a secure channel.

Exercise 10.6

- When using trusted hardware, should we care about covert channels[1]?

[1]`https://fas.org/irp/nsa/rainbow/tg030.htm#2.1`

Exercise 10.7

- In Dynamic Data Authentication, would encrypting the PIN verification solve the YesCard problem? If yes, justify. If no, propose a solution.

Exercise 10.8

- Is there any timing side-channel in the following code? If a timing side-channel exists, (1) explain how to exploit it. (2) change the code to prevent the attack (Hint: try to make time measurements useless for the adversary).

Example A:

```
Bool CheckPin(string check, string pin){
    for (int i = 0; i < 4; i++)
        if (check[i] != passcode[i])
            return false;
    return true;
}
```

Example B:

```
// exp(sec, x) computes x**{sec} mod n
const BigNumber n = [A big number];
BigNumber exp(BigNumber sec, BigNumber x){
    BigNumber out = 1;
    for ( int i = 0; i < sec.size(); i++){
        if (sec[i] == 1)
            out = (out*x) %n;
            out = (out * out)%n;
    }
    return out;
}
```

Example C:

```
// This function computes the dot product of two vectors
// The contents of the vectors are confidential
int dotproduct(int *veca, int *vecb, int len) {
  int acc = 0;
  for(int i = 0; i < len; i++)
    acc += (veca[i] + vecb[i]);

  return sum;
}
```

Solutions to the Exercises

Solution 10.1

1. Needs isolation. If an adversary can extract the secret key from the HSM, this adversary can sign any census update. Attestation is not absolutely needed. The city hall owns the trusted hardware on its premises, and therefore it need not prove that the trusted hardware is there (first attestation property). Regarding software run on the hardware, a signature (on the census, or anything else) can be verified externally given the public key. If the HSM performs any other operation, the signature would not pass the verification. (One could also argue that attestation is needed to ensure that the HSM signs once, and only once, a transaction and does not leak the secret key.)
2. Needs isolation. An adversary that extracts the secret key from the IoT sensor can impersonate this sensor, effectively tampering with the measurements. Same considerations regarding attestation as in (1).
3. Does not need isolation. The device is already in an isolated room. Needs attestation of hardware and code, as it is remote, we need to ensure the device is there when we are not in the room.
4. One needs attestation. Since the application is run remotely on an untrusted environment (the Cloud), it is important that SGX ensures the client has run code that performs the correct access control. One could argue that, since no one has physical access to the cloud where SGX is installed, there is no need for isolation. On the other hand, it could be argued that the Cloud is adversarial, and SGX needs to be protected to avoid tampering attacks. Since biometrics are stored in the clear, we are not concerned with the cloud trying to breach their confidentiality (It already has read access the templates).

Solution 10.2

The second option, two civil servants, is a better option. This configuration respects the separation of privilege principle. To avoid that a single action (the mayor is a malicious actor) can break the system, we require more than one actor to execute the operation, namely the census update.

Solution 10.3

Alice needs to store her private signing keys where it cannot be accessed (read/write) by her employees.

1. A general purpose CPU has no protection against major attack vectors: against a untrusted OS or against physical access to the system (including system RAM). However, if the physical environment can be secured otherwise (physical access controls), a general purpose CPU environment provides very high performance.
2. An enclave on a commercial processor (ARM TrustZone or Intel SGX/TDX) will provide defence against untrusted OS, and in some cases even against physical access to the system. Depending on the enclave technology, there may be a residual atttack surface from physical attack (including system RAM swapping/rollback). However, these environments often come with greater performance overheads, or resource limitations (SGX limits enclaves to 128MB physical memory).
3. Dedicated HSMs are built with an threat model including attackers with physical access, and contain intrusion protection mechanisms, and run their own software stack. However, HSMs come with some significant disadvantages. HSMs often have proprietary designs, and are expensive. The proprietary designs prevent external auditing and checks, and allows bugs to remain hidden. Performance and power cost might be higher than previous options. HSMs might be vulnerable to physical side-channels if no physical isolation is implemented. Development of software for HSMs can be challenging due to proprietary designs. Replication for fault tolerance can be challenging.

Solution 10.4

A malicious OS could reset the PCRs post-boot and then set them to an invalid OS hash. PCRs would then look as if a valid OS loaded.

Solution 10.5

1. The signature can be replayed! Once the app has authenticated once, if the adversary captures the signature, they can reply it later even if the app has not loaded properly.
2. The user / OS can reboot the machine after attestation and run arbitrary software pretending to be the application that performed the attestation. The key exchange ensures that only the application can communicate with the challenger.

Solution 10.6

1. In general, no if the trusted hardware is isolated and implements attestation. In that case, the adversary cannot measure emissions from the hardware or run code on the hardware to establish covert channels.

Solution 10.7

No, the problem is authentication, not confidentiality. A solution is to authenticate the PIN transmission, e.g., have the card sign the OK response.

Solution 10.8

Example A:
In the CheckPin, the function spends time linear to the size of the largest prefix of check that matches the passcode. The attacker can use timing to infer the length of the correct prefix. The attacker starts with a partially correct passcode (initially empty). In each step, it adds a digit to the end of the current password. The one that consumes more time in the function is the correct digit. Fix: Use a constant time function.

```
Bool CheckPin(string check, string pin){
  Bool ok = true;
  for (int i = 0; i < 4; i++){
    Ok = Ok & (check[i] == passcode[i])
  }
  return ok;
}
```

Example B:
The exp(sec, x) computes x^{sec} mod n. Multiplying out*xis is time consuming. The system only performs this action in the i^{th} step if the secret key's i^{th}bit is one. Based on timing difference, the adversary can guess secret key's i^{th} bit. Afterward, the attacker can repeat this process for other bits. Fix: make the function constant time. The computation time should be independent of the secret key.

```
BigNumber exp(BigNumber sec, BigNumber x){
  BigNumber out = 1;
  for ( int i = 0; i < sec.size(); i++){
    tmp = (out*x) %n;
  if (sec[i] == 1)
    memcpy(out,tmp);
  else
    memcpy(out,out);
  out = (out * out)%n;
  }
  return out;
}
```

Example C:
This function has a constant number of operations irrespective of the contents of vectors A and B. The programmer will expect this program to be constant time. In some modern machines, surprisingly, even this program might not be constant time. Two optimizations, zero-idiom and identity-idiom, introduce variance in the multiplication operation. If either of the multiplication operands is 0 or 1, the processor can skip the multiplication — the outcome of the multiplication will be 0 or the other operand respectively. Hence, multiplying vectors containing 0 or 1 runs faster than vectors without.

Unfortunately, there is no fast and easy solution. A programmer can implement constant-time multiplication based on integer operations, but there is no guarantee that the integer operations will be constant time either. To properly fix this side channel, future system designs need to include a hardware-software contract allowing programmers to only use constant-time operations.

Besides having a constant time, there are other possible, but worse, countermeasures:

Hiding: The system adds noise to the sensitive operations to hide the signal (valuable data).

```
Bool RandomizedCheckPin(string check, string pin){
    Bool ok = true;
    for ( int i = 0; i < rand(); i++)
        check[i] == passcode[i];     // waste time
    for ( int i = 0; i < 4; i++)
        If (check[i] != passcode[i])
            Ok = false;
    return ok;
}
```

In this example, it performs a random number of comparison prior to the real check. Hence, the timing of the function will not be correlated with the length of the correct prefix. This remediation is ineffective if the adversary can measure the emissions or current consumption of the processor, since they will be able to watch and distinguish the two loops.

Masking: The system does not operate directly on the sensitive data.

```
Bool HidingCheckPin(string check, string pin){
    // There is no correlation between the input of the hash
    // and the necessary time to compute the hash.
    check_hash = Hash(check);
    pin_hash = Hash(pin);
    return check_hash == pin_hash;
}
```

In this example, the check is performed on the hashes of the inputs (check and pin) rather than the inputs directly. But, pick a a good hash function!