CS 161 Computer Security
Spring 2010 Paxson/Wagner

Discussion 2

February 2, 2010

1. TCB (Trusted Computing Base)

(a) Is trust a good thing? Why or why not?

(b) What is a trusted computing base?

(c) What can we do to reduce the size of the TCB?

(d) What components are included in the (physical analog of the) TCB for the following security goals:

i. Preventing break-ins to your apartment
ii. Locking up your bike
iii. Preventing people from riding BART for free
iv. Making sure no explosives are present on an airplane
v. Preventing all the money from being stolen from a bank vault

Answer

(a) It's great to trust a friend, but it's bad to have to trust a component in a system that has security goals. It means that the component can violate your security goals if it fails. This is the difference between something you *trust* and something that is *trustworthy*.

(b) It is the set of hardware and software on which we depend for correct enforcement of policy. If part of the TCB is incorrect, the system's security properties can no longer be guaranteed to be true. (Paraphrased from Pfleeger.)

(c) Privilege separation can help reduce the size of the TCB. You will end up with more components, but not all of them can violate your security goals if they break.

(d) (This list is not necessarily complete.)

i. the lock, the door, the walls, the windows, the roof, the floor, you, anyone who has a key
ii. the bike frame, the bike lock, the post you lock it to, the ground
iii. the ticket machines, the tickets, the turnstiles, the entrances, the employees
iv. the TSA employees, the security gates, the "one-way" exit gates, the fences surrounding the runway area (but NOT the airline employees, restaurant employees, others?)
v. the vault, the owner + the manager (together, but not separately, assuming one has the code and the other has the key)

2. Security Principles The following are the security principles we discussed in lecture:

A. Security is economics

B. Least privilege

C. Use failsafe defaults
D. Separation of responsibility
E. Defense in depth
F. Psychological acceptability
G. Human factors matter
H. Ensure complete mediation
I. Know your threat model
J. Detect if you can't prevent
K. Don't rely on security through obscurity
L. Design security in from the start

Identify the principle(s) relevant to each of the following scenarios:

(a) New cars often come with a valet key. This key is intended to be used by valet drivers who park your car for you. The key opens the door and turns on the ignition, but it does not open the trunk or the glove compartment.

(b) Many home owners leave a key to their house under the floor mat in front of their door.

(c) Convertible owners often leave the roof down when parking their car, allowing for easy access to whatever is inside.

(d) Warranties on cell phones do not cover accidental damage, which includes liquid damage. Unfortunately for cell phone companies, many consumers who accidentally damage their phones with liquid will wait for it to dry, then take it in to the store, claiming that it doesn't work, but they don't know why. To combat this threat, many companies have begun to include on the product a small sticker that turns red (and stays red) when it gets wet.

(e) Social security numbers, which we all know we are supposed to keep secret, are often easily obtainable or easily guessable.

(f) The TSA hires a lot of employees and purchases a lot of equipment in order to stop people from bringing explosives onto airplanes.

Answer (Note that there may be principles that apply other than those listed below.)

(a) Principle of least privilege. They do not need to access your trunk or your glove box, so you don't give them the access to do so.

(b) Unfortunately we often do rely on security through obscurity. The security of your home depends on the belief that most criminals don't know where your key is. With a modicum of effort, criminals could find your key and open the lock.

(c) Security is economics. Even if they left the top up, it would be easy for a criminal to cut through it. If the criminals did that, it would cost the owner the cost of the items in the car and the cost of a new roof!

(d) Detect if you can't prevent. People will try to scam cell phone manufacturers, and there is nothing the companies can do to stop this. But they can (and do) detect when people have voided their warranty via liquid damage.

(e) Design security in from the start. SSNs were not designed to be authenticators, so security was not designed in from the start. The number is based on geographic region, a sequential group number, and a sequential serial number. They have since been repurposed as authenticators.

(f) Security is economics. They spend a lot of money to protect airplanes, lives, and the warm/safe/fuzzy feeling that people want to have when they fly.

3. Adversaries

(a) When you book a flight on Southwest airlines, Southwest sends your ticket information to you via email. This email contains all the information you need to modify your itinerary (add, change, or cancel flights) and print your boarding pass. However, email is sent in the clear, meaning that anyone between your computer and the Southwest servers can read your messages and take your flight information. Moreover, for many of us, Google or Microsoft eventually gets to see your email as it sits in your inbox. Should we be concerned about this? Why not have Southwest send a physical envelope to your apartment where you would at least have evidence of tampering?

(b) Imagine you are a highly motivated attacker who wants to travel under someone else's name. How might you take advantage of Southwest's system?

Answer:

(a) Economics and threat model. For most people, this sort of manipulation is not of any concern. We are not worried about the threat of someone intercepting our messages; it just is not interesting to anyone out there. This is called a "threat model": the threats you are actually concerned with in your system. The other side of this is economics. We are more than happy to take the (minor) risk of someone intercepting our email because it is much more convenient than USPS.

(b) Print out two copies of a victim's boarding pass. Photoshop one to have your own name. DoS the victim so they cannot arrive at the airport. Go through TSA security with the modified boarding pass and your own id. Arrive at gate and use the unmodified boarding pass. As long as the victim does not arrive at the airport, you should be able to board easily.

4. Alternative Access Methods Suppose you have a Linux machine for your personal use. This includes storing sensitive information including your banking information, tax documents, calendar, etc. Naturally, you want to protect this information, so you make sure that it is readable and writable only by your user account. Assume your user account is password protected by an extremely strong password. Excluding software vulnerabilities in programs you may be running, is your information safe? If not, what security principle(s) apply, and what might you do to make your system more secure?

Hint: Consider physical access to the machine.

Answer: If the attacker gains physical access to your machine, they can take your hard disk and plug it into their own machine. Once that is done, they can read whatever data they want because they are in control of the system. Even simpler, they could just boot your machine off of an Ubuntu Live Disc, for example. This is an issue of complete mediation. You protected digital access to the machine, but you did not consider physical access.

Of course, you may not be worried about physical access. After all, we have locks on our doors for a reason. Don't forget about threat models: what threats are you actually concerned with?

You might solve this issue by encrypting the files on your machine.

5. **Reasoning About Code** For the following function, can you prove that the postcondition holds, i.e. that array dereference `my_stack->val[sp]` never goes out of bounds? If you can prove it, what are the necessary preconditions? If it cannot be proven, where does it fail?

```
#define MAXVAL 100

struct stack {
    int sp = 0;
    double *val;
}

/* Ensures my_stack->val[my_stack->sp] is never out of bounds. */
void push(double f, struct stack *my_stack){
  if(my_stack->sp < MAXVAL){
    my_stack->val[sp] = f;
    my_stack->sp++;
  } else{
    printf("error: stack full");
  }
}
```

Answer:

```
#define MAXVAL 100

struct stack {
    int sp = 0;
    double *val;
}

/* Ensures my_stack->val[my_stack->sp] is never out of bounds. */
//precondition: my_stack is a valid, non-NULL address && my_stack->sp >= 0
//my_stack->val is a non-NULL address && len(my_stack->val) >= MAXVAL
void push(double f, struct stack *my_stack){
  if(my_stack->sp < MAXVAL){
    //my_stack->sp >= 0 && my_stack->sp < MAXVAL &&
//len(my_stack->val) >= MAXVAL
    my_stack->val[my_stack->sp] = f;
    //my_stack->sp > 0  && len(my_stack->val) >= MAXVAL
    my_stack->sp++;
  } else{
    printf("error: stack full");
  }
}
```

At each line of the code is the invariant that is always true whenever that line of code is executed. We see that `my_stack` is only ever indexed by `sp` when the invariants show that $0 \leq$ `sp && sp` $<$ `MAXVAL`.

Because the precondition is that `len(my_stack->val)` $\geq$ `MAXVAL`, `sp` never indexes beyond the length of `my_stack`. We have shown the postcondition of the function holds if the precondition is met.