292 - Fall 2021
Quantitative Information Flow and Side Channels

Instructor: Tevfik Bultan

Lecture 3

Slides for this lecture are based on the following papers:

Boris Köpf, David A. Basin. An information-theoretic model for adaptive side-channel attacks. ACM Conference on Computer and Communications Security 2007: 286-296

Boris Köpf, David A. Basin. Automatically deriving information-theoretic bounds for adaptive side-channel attacks. Journal of Computer Security 19(1): 1-31 (2011)

Analyzing leakage

- In Lecture 2, we discussed information leakage and showed that information leakage can be modeled using entropy
- We saw that the information leaked can be characterized as the initial uncertainty about the secret minus the remaining uncertainty about the secret
- We saw that different entropy formulations can be used to quantify the amount of uncertainty about the secret

Analyzing leakage vs. analyzing attacks

- The leakage measurements we discussed in Lecture 2 answer only the following question:
  - How much information is leaked after a single interaction with the program (i.e. executing the program once)
  - An attacker can execute the program multiple times!
- Moreover, we simplified the problem and did not take into account that attacker may control part of the input
  - There may be a public input that influences that program's behavior which can be controlled by the attacker

Modeling and analyzing attacks

- In order to model and analyze the attacks we need to take into account that
  - Attacker can run the program multiple times
  - Attacker can provide public inputs to the program

Adaptive attacks

- Note that the first time the attacker interacts with the program, the attacker has no information about the secret
  - First, attacker needs to provide a public input to the program without prior knowledge about the secret
- However, the next time the attacker interacts with the program, they may already have some information about the secret based on their previous interactions with the program
  - So, attacker should try to pick a public input that is consistent with what they know about the secret

Adaptive attacks

- In order to develop an efficient attack, the attacker has to “adapt” its interactions with the program
  - based on the information they gain by their prior interactions with the program

- This type of attacks are called “adaptive attacks”

An example: password checker

```
check(password,guess) {
    if (password == guess) permit();
    else error();
}
```

- Inputs to a password checker are the password and the guess provided by the user
- Password is the secret input
    - Attacker does not know the password
- Guess is the public input
    - Attacker controls the guess

An example: password checker

Consider a password checker with no side channels.

However, the password checker still leaks information since its main channel output tells the user if the guess matches the password or not.

An attacker can develop a brute-force attack to discover the password

Note that, an efficient brute-force attack should be an adaptive attack

An example: password checker

How should an efficient brute-force attack adapt?

- The public input provided by the attacker is the guess
- First guess could be a random value chosen from the domain of the password
  - such as a random string
- However, the next input selected by the attacker should not be a random value
  - for an efficient brute-force attack, the attacker should try a guess that they did not try before!
  - this is the adaptive strategy that makes the brute-force attack work

Modeling programs with public input

S is the secret input to the program.

P is the public input to the program. We will assume that this input is under the control of the attacker

O is the public output of the program.

f is going be our model for the program and it is a function from values of S and P to values of O, we will assume that f is deterministic

Modeling programs with public input

- Given a program

$$f : \mathcal{S}, \mathcal{P} \rightarrow \mathcal{O}$$

- So for the password example

  Secret (S) is the password

  Public input (P) is the guess

Partitioning the secret domain

- Given a program

$$f : \mathcal{S}, \mathcal{P} \rightarrow \mathcal{O}$$

- The values we observe as the output of the program define an equivalence relation for the secret S

$$s \sim s' \text{ iff } f(s, p) = f(s', p)$$

- So, by observing output of the program, we partition the secret values to equivalence classes

Partitioning the secret domain using public input

- Let π = {B_1,B_2, …, B_r} denote a partition of the secret domain
  - where B_is (blocks/cells of the partition) are mutually exclusive
  - and their union is equal to the secret domain

- Note that, every equivalence relation on the secret domain corresponds to a partition of the secret domain
  - where the values that are equivalent are placed in the same block/cell of the partition

Partitioning the secret domain using public input

- Each public input p defines an equivalence relation/partition of the secret domain
- Let π_p denote the partition of the secret domain induced by the public input p and the program f

- Let $\mathbf{\Pi} = \{\pi_p | \ p \in \mathcal{P}\}$ denote the set of all partitions induced by the public inputs ($\mathcal{P}$ is the domain of public inputs)

Attack steps

Based on this model, two basic attack steps can be characterized as follows:

- Attacker chooses a partition from $\mathbf{\Pi}$
  - i.e., attacker chooses an input
- Attacker executes the program on the chosen public input
  - the output of the program reveals which partition block/cell the secret belongs to

Partition refinement

Once the attacker figures out which partition cell the secret belongs to, now the attacker has to “adapt” its strategy to pick the next public input

Note that, based on their knowledge of the secret, the attacker can refine the partitions.

For example, if the attacker knows that the secret is in the set $A \subseteq S$, then the attacker can use a refined partition which partitions the set A

Partition restriction

Given a partition π of S and a set A $\subseteq$ S

Let A $\cap$ π denote a partition of A which is obtained by taking the intersection of each cell/block of π with A

This is called a restriction of partition π to A

Partition refinement

A partition π_1 is a refinement of partition π_2 if each block/cell of π_1 is a subset of some block/cell of π_2

We denote π_1 is a refinement of partition π_2 as

$$\pi_1 \sqsubseteq \pi_2$$

Given two partitions π_1 and π_2 the partition $\pi_1 \cap \pi_2$ is their refinement obtained by taking intersection of all their blocks/cells

$\pi_1 \cap \pi_2 \sqsubseteq \pi_1$ and $\pi_1 \cap \pi_2 \sqsubseteq \pi_2$

Attack strategy

An attack strategy can be thought of as an attack tree

- Each node corresponds to a subset of the secret domain
- Root node corresponds to the full secret domain
- Each node of the tree corresponds to a partition/public input selection
  - Children of each node n in the tree correspond to the blocks/cells of a partition obtained by the restriction of a partition in $\mathbf{\Pi}$ to the subset of the secret domain that correspond to the node n

Attack Strategy

Given the set of partitions $\mathbf{\Pi}$, an attack strategy is a triple (T, v^*, L) where

- $T = (V, E)$ is a tree with $v^* \in V$ is the root
- $L : V \rightarrow 2^S$ is a vertex labeling with the following properties
  - $L(v^*) = S$
  - For every $v \in V$ there is a $\pi_p \in \mathbf{\Pi}$ where

$$L(v) \cap \pi_p = \{L(w) \mid (v,w) \in E\}$$

An example

Consider the secret domain {1, 2, 3, 4} and consider the set of partitions:

Π = {

{{1},{2,3,4}},

{{1,2},{3,4}}

{{1,2,3},{4}}

}

We can show different attack strategies as attack trees

UCSB

Example 1

```
f(S,P) {
    if (P == 1)
        switch (S) {
            case 1: sleep(0);
            case 2: case 3: case 4: sleep(1);
            }
    if (P == 2)
        switch (S) {
            case 1: case 2: sleep(0);
            case 3: case 4: sleep(1);
        }
    if (P == 2)
        switch (S) {
            case 1: case 2: case 3: sleep(0);
            case 4: sleep(1);
        }
}
```

Example 2

```
f(S,P) {
    if (P == 1)
        switch (S) {
            case 1: case 2: sleep(0);
            case 3: case 4: sleep(1);
        }
    if (P == 2)
        switch (S) {
            case 1: case 3: sleep(0);
            case 2: case 4: sleep(1);
        }
}
```

Example 3

```
f(S,P) {
    if (P == 1)
        switch (S) {
            case 1: case 2: case 3: sleep(0);
            case 4: sleep(1);
        }
    if (P == 2)
        switch (S) {
            case 1: case 3: sleep(0);
            case 2: case 4: sleep(1);
        }
}
```

Adaptive vs. non-adaptive attack strategies

An attack strategy is called non-adaptive:

- if the attacker does not have access to the output of the program
  - i.e., the attacker does not know which block/cell the secret belongs to after each step

An adaptive attack strategy does have access to the output of the program and can tell which block/cell the secret belongs to after each step and can use this information in selecting the next partition

Evaluating attack strategies

Given this model, we can use different entropy measures to evaluate the effectiveness of different attack strategies

Let V_a denote the choices the attacker makes in selecting the partition, i.e., the attack strategy of the attacker

Then $H(S \mid V_a)$ denotes the remaining uncertainty about the secret given the attack strategy V_a

We can use this measure to evaluate attacks

We can also use different entropy definitions like Shannon entropy, min-entropy, guessing entropy

Optimal attacks

Let $\Phi_H(n) = H(S \mid V_a)$ where a is an optimal attack of length n

Hence, $\Phi_H(n)$ denotes the remaining uncertainty after maximum amount of leakage for the given program against an optimal attack of length n

Again, different entropy definitions can be used.

Optimal attacks

$\Phi_H(n)$ can be computed in

$O(n \times |P|^e \times |S| \times \log|S|)$ where $e = r^n$ and r is the maximum number of cells in a given partition

Approximating optimal attacks

$\mathbf{\Phi}_H(n)$ can be approximated using a greedy heuristic

And approximation of $\mathbf{\Phi}^{greedy}_H(n)$ can be computed in

$O(n \times r \times |P| \times |S|^2)$ where r is the maximum number of cells in a given partition