Leakage in presence of an active and adaptive adversary

Cristian Ene, Laurent Mounier
e-mail: Cristian.Ene@univ-grenoble-alpes.fr
Laurent.Mounier@univ-grenoble-alpes.fr
January 14, 2021

Measuring the information leakage of a system is very important for security. From side-channels to biases in random number generators, quantifying how much information a system leaks about its secret inputs is crucial for preventing adversaries from exploiting it; this has been the focus of intensive research efforts in the areas of privacy and of quantitative information flow (QIF). For example, both programs in Figure 1 are leaking some additional information about the secret if one can measure the execution time or if one can observe the instruction cache. Moreover, by interacting iteratively with the application, the adversary is able to improve his knowledge [4].

```c
void compare(int l, int s){
    if (s<l) {
        write_log('too large'); // 1 sec.
    } else {
        some_computation(); // 2 sec.
    }
}
```

```c
int pwdCheck(char *l, char* pwd){
    unsigned i;
    for (i=0; i<B_Size; i++)
        if (l[i]!=pwd[i])
            return 0;
    return 1;
}
```

Figure 1: Leaking programs

Hence the overall scenario (Figure 2) is the following one:

- Some secret \( x \in \mathcal{X} \) is generated and provided to the application
- Iteratively and adaptively,
  1. The adversary provides some public input \( l \in \mathcal{L} \) to the application
  2. The application does some computation and outputs some \( y \in \mathcal{Y} \)

The adversary’s knowledge about the the secret \( x \in \mathcal{X} \) at some moment \( i \) is called the prior probability \( \pi_i \) (e.g. initially, \( \pi_0 \) would be the uniform distribution on \( \mathcal{X} \)). In our context, the application corresponds to a family of probabilistic channels \( (C_l)_{l \in \mathcal{L}} \), such that for each \( x \in \mathcal{X} \) and \( l \in \mathcal{L} \), it returns a \( y \in \mathcal{Y} \) according to some distribution \( P_{C_l}(Y=y \mid X=x) \). In the considered
scenario, the adversary interacts iteratively (Figure 3) with the application until his knowledge $\pi_k$ achieves some desired vulnerability level $\mathcal{V}(\pi_k)$.

\begin{align*}
\pi \leftarrow \pi_0; & \quad \text{// (1)} \\
\text{while } \mathcal{V}(\pi) \leq \epsilon \text{ do } & \quad \text{// (2)} \\
& \quad l_0 \leftarrow \text{argmax}_{l \in \mathcal{L}} \mathcal{V}[\pi^\top \mathcal{C}_l]; & \quad \text{// (3)} \\
& \quad \text{Execute App with input } l_0; \\
& \quad \text{Get the output } y_0 \text{ returned by App;} \\
& \quad \text{Update } \pi \text{ according to } y_0 & \quad \text{// (4)}
\end{align*}

where

- (1) $\pi_0$ is the initial probabilistic information about the secret $x$ (called the prior)
- (2) $\epsilon$ is the intended level of knowledge (modelled by some measure $\mathcal{V}$) about the secret
- (3) find the “best” input $l_0$ that optimises the leakage; $\pi^\top \mathcal{C}_l$ is the hyper-distribution corresponding to executing App with prior $\pi$ and input $l_0$, i.e. the distribution of posteriors $\mathcal{P}(X \mid Y = y_0, L = l_0)$, each with probability $\mathcal{P}(Y = y_0 \mid L = l_0)$
- (4) use the Bayes law to update the belief: $\pi \leftarrow \mathcal{P}(X \mid Y = y_0, L = l_0)$

Several issues can be investigated in this internship:

- What are the best choices for the measures $\mathcal{V}$ and $\mathcal{V}[\pi^\top \mathcal{C}]$ ?
- How to compute/approximate for each input $l$, the associated probabilistic channel $\mathcal{C}_l$ (do we have the source code or not for App)?
- How to compute/approximate $\text{argmax}_{l \in \mathcal{L}} \mathcal{V}[\pi^\top \mathcal{C}_l]$? , given that in most of the cases the sets of inputs $\mathcal{L}$, secrets $X$ and observables $\mathcal{Y}$ can be very large, and that in the most realistic scenario, the output $Y$ will rather be a continuous random variable [2].
- Information-theoretic vs. probabilistic polynomial-time adversary
The topic of this internship can be oriented in various directions:

- refining the scenario from Figure 3 in a grey-box case, where the attacker has the binary code of the application
- apply machine learning (ML) methods [3, 5] in order to get the necessary scalability (whenever the sets of inputs $L$ or secrets $X$ are very large or if the output $Y$ is a continuous random variable) for the grey- or black-box measurements
- implementing the scenario from Figure 3 in order to synthesis an adaptive attack [6] or to measure the vulnerability for a concrete application.

References


