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

October 21, 2022

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 [3].

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

```
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:

- There is some secret $x \in \mathcal{X}$ randomly generated (for example a database containing some confidential informations)
- Iteratively and adaptively,
  1. The adversary provides some application together with some public input $l \in \mathcal{L}$
  2. The application (taking x as a secret parameter) 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 π_i (e.g. initially, π_0 would be the uniform distribution on $\mathcal{X}$). In our context, an application corresponds to a family of

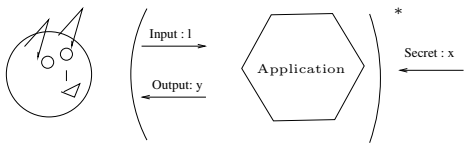

Figure 2: The target scenario

probabilistic channels $(\mathcal{C}_l)_{l\in\mathcal{L}}$, such that for each $x \in \mathcal{X}$ and $l \in \mathcal{L}$, it returns a $y \in \mathcal{Y}$ according to some distribution $\mathcal{P}_{\mathcal{C}_l}(Y = y \mid X = x)$. In the considered scenario, the adversary interacts iteratively (Figure 3) with the database containing the secret information x until his knowledge π_k achieves some desired vulnerability level $\mathcal{V}(\pi_k)$.

```
π←π0;                                              // (1)
while V(π) ≤ ε do                                  // (2)
    l0←argmax_{l∈L}  V[π▷Cl];                      // (3)
    Provide and Execute App with input l0;
    Get the output y0 returned by App;
    Update π according to  y0                      // (4)
```

where

- (1) π_0 is the initial probabilistic information about the secret $x[1]$ (called the prior)
- (2) ϵ 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 ; $\pi\triangleright\mathcal{C}_{l_0}$ is the hyper-distribution corresponding to executing App with prior π 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: $\pi\leftarrow\mathcal{P}(X \mid Y = y_0, L = l_0)$

Figure 3: Attacker's strategy

In [2], the authors explicitely track the querier's belief about secret data, represented as a probability distribution, and deny any query that could increase knowledge above a given threshold. Their security model is quite restrictive, resulting in denying even a very popular query corresponding to a password checker. In [5] we extended this work in several directions: we developped an algorithm able to quantify the information leaked by the application about the secret in a more realistic security model and taking into account a more powerfull adversary, able to get side-channel informations about the execution

of the application (for example, the branchings taken during an execution).

In this internship, we plan to build upon this work in several directions:

- to extend the algorithm from [5];
- to implement this algorithm via abstract interpretation, for example by extending the probabilistic polyhedra model introduced in [2];
- apply the prototype tool in order to measure the vulnerability of different benchmarks from literature;
- implement the scenario from Figure 3 in order to synthesis an adaptive attack [4].

References

[1] Mário S. Alvim, Konstantinos Chatzikokolakis, Carroll Morgan, Catuscia Palamidessi, Geoffrey Smith, and Annabelle McIver. An Axiomatization of Information Flow Measures. *Theoretical Computer Science*, 777:32–54, 2019.

[2] P Mardziel, S Magill, M Hicks, and M Srivatsa. Dynamic enforcement of knowledge-based security policies using probabilistic abstract interpretation. *Journal of Computer Security ACM SIGSOFT*, 21(4):463–532, 2013.

[3] Quoc-Sang Phan, Lucas Bang, Corina S Pasareanu, Pasquale Malacaria, and Tevfik Bultan. Synthesis of adaptive side-channel attacks. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 328–342. IEEE, 2017.

[4] Seemanta Saha, William Eiers, Ismet Burak Kadron, Lucas Bang, and Tevfik Bultan. Incremental attack synthesis. *ACM SIGSOFT Software Engineering Notes*, 44(4):16–16, 2019.

[5] Valentin Viollet. Weighted side channel analysis. master cybersecurity, master of science in informatics at grenoble, master mathematiques and applications, september 2022.