The "Taint" Leakage Model

Ron Rivest

Crypto in the Clouds Workshop, MIT
Rump Session Talk

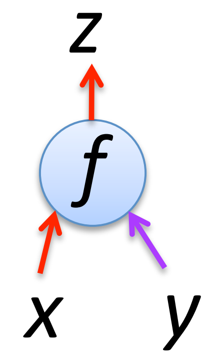
August 4, 2009

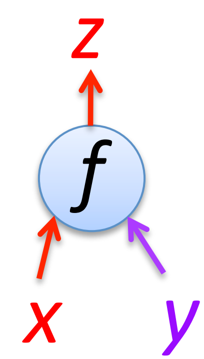
Taint

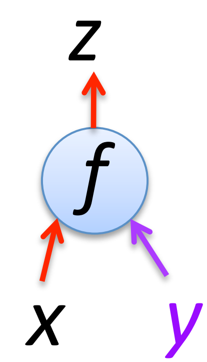
- Common term in software security
- Any external input is *tainted.*
- A computation with a tainted input produces *tainted output.*
- Think *tainted = "controllable" by adversary*
- *Untainted* values are private inputs, random values you generate, and functions of untainted values.
- E.g. what values in browser depend on user input?

Proposed "Taint Leakage Model"

- Only computations *with tainted inputs* leak information.
- Adversary learns output and *all* inputs (even untainted ones) of a computation with a tainted input.
- Define a valued as *spoiled* if it is untainted but input to a computation with a tainted input.
- Examples: tainted values in red,
  spoiled values in purple
  clean values in black (untainted and unspoiled)
  - *z = f(x,y)* No leakage; clean inputs gives clean outputs
  - *z = f(x,y)* *x* tainted so *z* tainted & *y* spoiled
  - z = f(x,y) x clean & y spoiled so z clean
- Leakable iff tainted or spoiled
- Adversary can learn all tainted and spoiled values.
- Leakage may be *unbounded* or *bounded.*



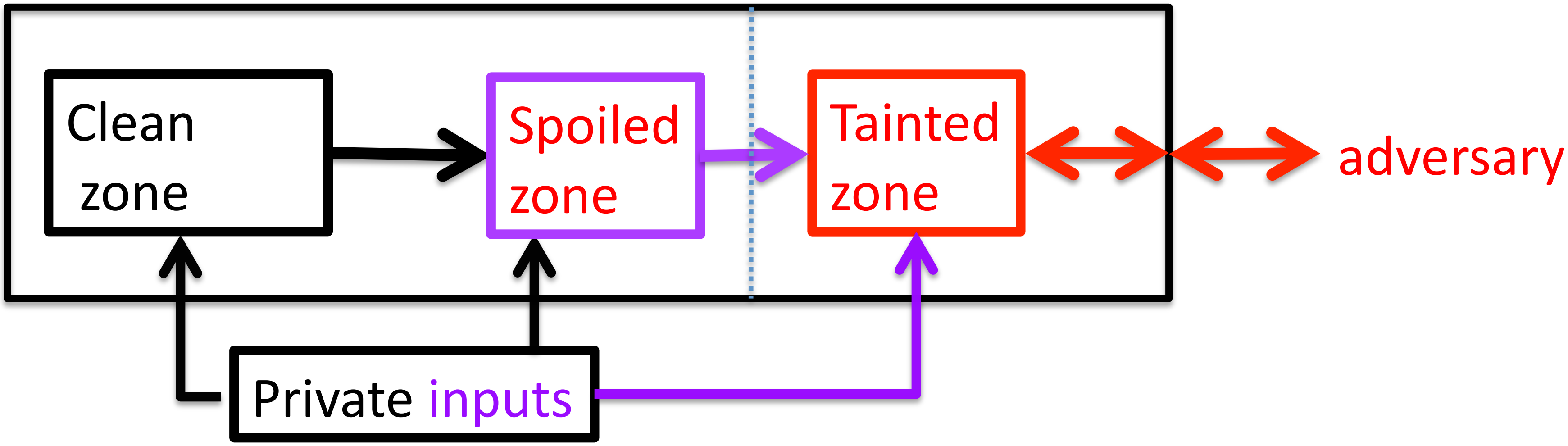




Motivating Sample

- What attacks motivate this model?
- Various forms of chosen-input attacks, such as timing attacks or differential attacks.
- $C = E_K(M)$
- Here K is spoiled, and thus leakable; this models timing attacks on K using adversary-controlled probes via control of M.

Model useful in building systems



Zones can be implemented separately

-- e.g. untainted on a TPM (or remote!)

-- clean zone may include a random source, and can do computations (e.g. keygen)

-- output could even be stored when independent of adversarial input (ref Dodis talk in this workshop)

Example

- *Encrypting (tainted) message M with key K :*
  - $C = E_K(M)$
    - *K is spoiled and thus leaks (since M is tainted)*
  - $C = (R, S)$ *where* $S = M$ xor Y *and* $Y = E_K(R))$
    - *K is not tainted or spoiled, thus protected*
    - *S is tainted (since M is tainted)*
    - *R is spoiled (since paired with tainted S) (but known anyway)*
    - *Y is spoiled (since M is tainted)*
- *Protect long-term keys by using random ephemeral working keys. (Can do similarly for signatures)*
- *Taint model more-or-less distinguishes between chosen-plaintext and known-plaintext attacks.*
- *Related to "on-line/off-line" primitives…*

Relation to other models

- Incomparable…
- Adversary is weaker with taint model than with computational leakage, since values not depending on adversarial input don't leak.
- Adversary is stronger than with bounded leakage models, since it is OK to leak *all* inputs and output of computation with tainted input.
- Taint model doesn't capture all attacks (e.g. power-analysis, memory remanence attacks, …)

Discussion

- Contribution here is probably mostly terminology; model presumably implicit (or explicit?) in prior work.
- Results in taint leakage model may be easy in some cases (e.g. using empheral keys). (ref Dodis talk in this workshop)
- Goals typically should be that leakage does at most temporary damage….
- *What can be done securely in this model?*

The End