Side Channels

April 1, 2014

1 Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow

1.1 Summary

This paper investigates side channels in real world web applications. They first present an abstract model of web applications and their interactions. They then use side channels to learn private information about users on popular website. Specifically, they observe packet sizes of communications between users and web applications, which reveals secret information as a side effect. For example, by measuring transmissions to Google Health, the authors could determine which symptoms a user looked up. The authors then analyze the effectiveness of mitigation techniques and claim that solutions need to be application specific.

1.2 Questions

Mike: Do you think this is actually a side channel? I claim that it is not since you are observing the packet size, which is dependent on the code's functionality.

Mike: Do you buy their argument that mitigation techniques must be application specific? I think it is probably true, but I don't believe their argument.

2 Language-Based Control and Mitigation of Timing Channels

2.1 Summary

This paper focuses on mitigating leaks from external timing channels. External timing channels occur when adversaries learn information from a program's interaction with the outside world. Internal timing channels happen when timing affects the results within a system, which reveals information to attackers.

The authors mitigate external timing channels in the programming language. They use a standard information flow type system, but they add notions of a machine environment. This machine environment is invisible at the language

level, but affects timing. They assume hardware is divided into different labels and explicitly add hardware read and write labels for each statement. These read and write labels don't have anything to do with the semantics of the statements. The read label indicates the upper bound on the hidden machine state that the statement can read from. On the other hand, the write label is the lower bound on the machine state that the statement could write to.

To ensure that timing channels are closed, they basically enforce two invariants. 1) Implicit flows cannot happen (Program counter is less than the write label). 2) Properly keep track of the influence on the read label. To prove that their language satisfies non-interference, they make various assumptions about the hardware. They then argue that real machines, like Intel CPUs in no-fill mode, actually do satisfy these assumptions.

Eliminating timing leaks can be too restrictive so the authors added a mitigate statement to the language. These statements allow programmers to enable predictive mitigation around wrap expressions. Predictive mitigation works by estimating the expression's execution time. Then when the expression is evaluated, its runtime is padded out to the estimated time. If execution takes longer than the estimated time, the padding time is doubled from then on.

2.2 Questions

Mike: Can you just keep adding no-ops so that each branch takes the same number of statements, but the semantics are preserved? No, this is not sufficient because there is no language to talk about lower, machine level semantics. Timing could depend on more than just the number of statements executed.

Kris: I doubt whether this would work in real world systems like a medical database.
Mike: Maybe, maybe not, but they do have a simulation that this actually works on.

Andrew: What happens when your language allows you to get timing information? Does this no longer work?
Kris: Probably not.
Mike: Probably not, but let me think about it.

Mike: Why is Property 7 a reasonable property?