CS5431 – Computer Security Practicum Spring 2017

Lecture 9: Side Channels

April 28, 2017 *Instructor: Eleanor Birrell*

1 Revisiting Threat Models

Earlier in this course, we talked about the critical role that threat models play in systems security: the first step towards defending a system is defining the capabilities of the adversary against whom you wish to secure the system. Characterizing these capabilities includes specifying the interface through which an adversary might interact with the system, for example interacting with a UI or monitoring network traffic. If the threat model is accurate and the system is secured against that threat model, then the system will be secure. However, it turns out that anticipating all possible ways an adversary might interact with your system is difficult; communication models that are not captured by the threat model are known as *side channels*.

2 Side Channel Attacks

Over the last twenty years, researchers and malicious actors have discovered a wide range of side channels—that is, communication channels that lie outside the defenders' threat model—and have discovered varied and innovative techniques to leverage available side channels in order to recover secret information.

Differential Power Analysis The earliest known example of side channel attacks in practice dates back twenty years, when researchers observed that chips are built out of individual transistors, which are voltage-controlled switches. When charge is applied to a transistor's gate, current flows across the transistor substrate and delivers change to other transistor gates or to wires. This current consumes power. The power consumption of a chip is the result of the aggregate activity of all the transistors on that chip, and power consumption could (potentially) be observed by an adversary, thus introducing a side channel.

Power consumption can be exploited by a class of side channel attacks known as Differential Power Analysis (DPA); successful DPA attacks have been demonstrated against a wide range of encryption algorithms, including AES.

Recall that AES works as follows: to initialize, the secret key is used to define a set of round keys (the first n bytes are simply the secret key, the remaining bytes are determined by the specified key schedule), and the plaintext is represented as a 4×4 matrix of bytes. The algorithm then goes through a series of rounds, in each of which it (1) xors the 128 bit round key with the current cipher state, (2) it substitutes each byte using the S-box, a lookup table defined by the AES standard, (3) the bytes of

each row of the current cipher state are rotated, and (4) the bytes of each column of the current cipher state are mixed. (In the final round, the mix column step is eliminated and the cipher state is instead xor'd with the final round key).

It turns out that the power consumption at various points of this computation depends on the value of the secret key; one such point is immediately after the S-box computation in the first round (traces in which the least significant bit of any particular byte is zero are statistically different from traces in which the least significant bit of that byte is 1). To learn the secret key, an adversary begins by measuring the power consumption for a large set of encryptions of known plaintexts. The adversary then proceeds to determine the secret key byte by byte: for each byte, the adversary considers all 256 possible values. For each possible value, it goes through all the observed traces and separates them into two sets: those that result in the least significant bit of the first S-box being 0 and those that result in 1. It then compares the two sets: if they are statistically different, then the value of that byte of the key was guessed correctly (incorrect guesses result in sets of traces that are not statistically different).

Timing Attacks Recall that earlier in the semester we discussed the differences between cryptography in theory and cryptography in practice, and we looked specifically at a range of optimizations and attacks on RSA. In particular, we discussed how square-and-multiply is used to perform efficient modular exponentiation, but this requires branching on the bits of the secret key. This results in a timing channel. As in the case of DPA attacks, the secret key can be recovered by guessing the bits of the key iteratively and comparing the observed timing behavior to the expected behavior if that guess is correct. Timing attacks have also proven effective against other implementations of RSA, including OpenSSL's sliding-window implementation.

Cache Attacks The key observation behind cache attacks is that a computer's cache is a shared resource; all programs running on the same machine share the same cache and therefore the cache usage of one program can affect the cache behavior observed by another program, resulting in a side channel. Cache attacks, which leverage this side channel, have been demonstrated against a variety of algorithms including AES and RSA.

One type of cache attacks that is successful against AES is a Prime+Probe attack. Prime+Probe attacks on AES take advantage of the fact that AES implementations make use of stored lookup tables. Intuitively, you can think of the S-box as a lookup table; in practice, most AES implementations make use of a set of four lookup tables $T_0, \ldots, T_3$ that allow the four steps in an AES round to be merged into a single operation. This table (or tables) is stored in memory, and which values in that table are accessed depends on the plaintext message and the encryption key. An adversary that can determine which table values are accessed (for a known plaintext) can infer the key.

Prime+Probe depend of the fact that most modern machines use set-associate caches; the location where a memory value is cached is partially dependent on the memory address of that value. A set-associated cache is divided into W ways, each of which has S sets; each memory location is mapped to a particular cache set, but can be stored in any way depending on availability and eviction algorithm. Prime+Probe attacks therefore proceed as follows: the attacker begins by allocating an array A of size $S \cdot W \cdot B$ at a location congruent to the start of the lookup table mod $S \cdot B$ (here B is the cache line size). The adversary begins by reading a value from each location in array A; this has the effect of completely filling (priming) the cache. The adversary then triggers an encryption of a known plaintext message m. Finally, for each table T_i and for each cache set s, the adversary computes the total time required to access the W entries of A corresponding to the W ways. If the table entry was accessed by the encryption computation then the array value corresponding to that location will have been flushed from cache resulting in a slower measurement. Using these measurements, the adversary can determine the original key value.

3 Remote Side Channel Attacks

Side-channel attacks are typically associated with physical access. Certainly certain types of side-channel attacks (DPA, electromagnetic analysis, acoustic analysis, cold boot attacks) do inherently require physical access to the machine. And it does make sense that timing and cache attacks—which depend on fine-grained timing measurements—might be impractical over a network. Two lines of work have attempted to investigate the feasibility of side channel attacks in cases where the adversary does not have physical access to the target machine.

Remote Timing Attacks Remote timing attacks are conducted like standard timing attacks, except that timing measurements are taken over a network. In recent work, researchers explored the feasibility of such attacks across a local network. They demonstrated successful attacks on the RSA implementation in OpenSSL running on various applications, even though there were several routers and a network backbone between the attacker and the target. Their results indicate that, in general, networks with variance under 1 ms are likely vulnerable to timing attacks.

Cloud Side Channels Another case in which side channel attacks might be feasible despite a lack of physical access is when applications or services are run in the cloud. Cloud providers like Amazon's EC2 provide infrastructure as a service (IaaS); they provide VMs running on top of hardware and the consumer runs their own OS-platform-application on top of the provided infrastructure. Although it is possible to pay for dedicated machines, many users do not. This introduces the possibility that

an attacker might co-locate an adversarial instance on the same physical machine and use that proximity to perform a side-channel attack, e.g., a cache attack.

To successfully perform a cache attack on an EC2 instance, the attacker needs to (1) co-locate an attacker instance with a target instance and (2) detect co-residency. At a high level, an attacker can force co-location by brute force (spinning up lots of attacker instances), although attackers can improve the probability of successful co-location by forcing creation of more target instances (e.g., through increasing load on the target service) and/or by exploiting known features of the cloud cartography (e.g., setting user-defined parameters to maximize the chance of co-location with a target instance). A recent study found the chances of successfully forcing co-residency on EC2 (given 10 victim instances and 30 attacker instances) was about 60%. Co-residency can then be detected by observing cache load measurements while interacting with the system (e.g., making lots of HTTP GET requests). One experiment found that cache loads were significantly factor for non-co-resident instances under these circumstances. Of course, this assumes that the attackers interactions significantly affect the load on the victim system, a circumstance that is likely difficult to achieve for a high-profile target.

4 Defenses

Forced Equivalence The theory behind forced equivalence is to eliminate all differences (in power consumption,timing, cache patterns, etc) that depend on sensitive values. Unfortunately, this in inherently inefficient; all computations require the maximum possible resources. This can be relaxed somewhat by quantizing—requiring all computations to take some multiple of a fixed quantum—but it still has a significant performance overhead. Moreover, most such techniques are imperfect. They generally just reduce the signal to noise ration. Smaller variation between key values therefore renders successful side-channel attacks more difficult (and often requires an attacker to acquire more samples) but it does not preclude the possibility of successful side-channel attacks by a motivated and resourceful attacker.

Blinding The most popular current defense against side-channel attacks is blinding. Blinding is a general approach that involves randomly mauling the input (plaintext or ciphertext) prior to performing the cryptographic operation (encryption, decryption, or signing) and then canceling out the introduced randomness after the operations is complete. For example, RSA can be blinded by taking the ciphertext c, choosing a random value r, computing $c' = r^e \cdot c \bmod n$, decrypting the mauled ciphertext $m' = (c')^d \bmod n$, and then extracting the true plaintext message $m = m' \cdot r^{-1}$.

Blinding has the effect of preventing the adversary from knowing the input to the cryptographic operation (in the above example, it is now the random string c' instead of the known ciphertext c); since most attacks require knowing this input, blinding

is an effective defense against most known side-channel attacks.

LR Crypto A final approach to mitigating side-channel attacks is a new branch of cryptographic theory known as leakage-resilient cryptography. The motivating idea behind LR crypto is that current cryptographic protocols are inherently vulnerable to side-channel attacks because security proofs break if the adversary learns any information about the secret key. LR cryptographic protocols, which have been defined and constructed by cryptographic researchers, have two key features: (1) they remain secure even when the adversary learns partial information about the secret key, and (2) the secret key can be refreshed without changing the public key. As long as the rate at which the adversary can learn information about the secret key is bounded, the adversary will be provably unable to learn the secret key using any form of side-channel attack.

LR crypto currently performs significantly slower than standard cryptographic protocols and is not deployed in practice, but it offers an interesting approach toward future defense against side-channel attacks.