CSE 127 Lecture 6: Side-Channel Attacks

These notes were scribed by students from CSE 127 in Winter 2021. They have been lightly edited but may still contain errors.

In the previous lecture, we talked about the importance of isolation in building secure systems. The basic idea is that a secure system needs to protect secrets in hardware and software so that an attacker cannot access them across a trust boundary. Some common mechanisms for isolation include privilege separation, least privilege, and complete mediation. And, the assumption in building isolation in a system is that we know what our trust boundaries are and can control access to them.

However, isolation mechanisms in systems are still not perfect defenses. A machine may still have vulnerabilities. For example, an attack may find a vulnerability in an unprotected program, the kernel, a virtual machine, hypervisor, or the run-time system. The attackers can also find a hardware bug that lets them bypass isolation.

As computer scientists, we have learned to build computer systems on layers of abstraction. We build applications on top of the layer of the operating system. The operating system is built on top of the layer of the hardware. And, the hardware is built on top of the layer of physics. These abstraction layers let us specialize in one layer without having to worry about the other layers underneath. That is, we treat the layer below as a black box with well-defined behavior.

However, in this class, we have slowly discovered how the abstractions that we are using to build computer systems are imperfect. Since implementations of any code or machinery have artifacts and side effects, these artifacts and side effects can travel through different layers of abstractions. An attacker can observe these behaviors at different levels of abstraction to break the abstraction and violate the trust boundaries in a system we are building. Such source of information beyond the output specified by an abstraction is called a *Side Channel*. In this lecture, we will explore the history, examples, and countermeasures of side-channel attacks.

1 Overview and History of Side Channels

In 1945, at the end of World War II, the Soviet Union gave a great seal as a gift to a US ambassador. This seal was placed in the ambassador's residence, and it contained a passive listening device. This device would transmit audio when illuminated at a particular radio frequency. The device couldn't be found by looking for a signal because it was completely passive; it was powered by the radio frequency and only active while being illuminated at the signal from the Soviet. This was designed by Theremin, who was a brilliant electrical

engineer from the Soviet Union. The seal had been inside the ambassador's residence for 7 years until it was discovered in 1952 [**The˙Great˙Seal**]. This is an early case of advanced bugging.

During World War II, Bell Telephone discovered electromagnetic leakage in one-time pad teleprinters. In order to avoid it, they recommended keeping a 100-ft radius around the machine secured. However, this teleprinter leakage was rediscovered by the CIA in 1951. To avoid the leakage, the CIA recommended protecting the system with a shield and securing a 200-ft radius around it. In 1964, the US and NATO established a shielding rule called TEMPEST in order to protect their systems securely [**tempest**].

These side-channel attacks were the domain of the governments until 1985, when side-channel leakage was rediscovered in the open research world by Wim van Eck. He demonstrated side-channel image recovery from a CRT monitor to another monitor using off-the-shelf equipment [**Wim˙van˙Eck**].

This type of electromagnetic leakage can be demonstrated even for flat-panel displays. During the demonstration in 2004, Markus G. Kuhn could pick up the radiation coming from a screen connection cable and recover the original screen contents [**Kuhn**].

2 Examples of Side Channels

Side-channel attacks are using information gathered by observing a system. We will dive into different examples of side channels. One general family of side channels is the resources consumed by an operation. These could include timing, power consumption, and network traffic. Another general family of side channels is emission side channels, which are signals generated by performing some operation. These might include electromagnetic radiation, sound, and error messages.

2.1 Consumption Side Channels Example: Timing

We will now examine software side channels. Imagine if we implement a simple function that uses string comparison to compare the password entered by a user with the actual password. It would check character by character, and if there was a mismatch between characters between the inputted string and the actual string, we would return back. A malicious user who can run this program can measure the amount taken to return from our function. By knowing the amount of time taken to return from our function, the malicious user can deduce how many correct characters there were in the trial password they entered. Then they could use this information to learn the password character by character by timing the return time of our function.

2.2 Consumption Side Channels Example: Power Consumption and Network traffic

Power consumption is a measurable resource that can be recorded by an attacker with physical access to a device. Different operations may use different amounts of power in hardware. With this knowledge, we can interpret different patterns and use that to our

advantage. An additional example of a side-channel is analyzing network traffic. Network traffic side-channel attacks monitor how packets are communicated. By doing this, one can able to determine patterns from the traffic usage. Although this may not give direct access to the encrypted information, it provides ample amounts of data to the attacker. They can choose to continue with their current attack or look for something else based on the network packets transferred.

2.3 Emission Side Channel Example

Emission side channels are when an out-of-band signal is generated when performing an operation. This includes electromagnetic radiation. A nearby attacker would be able to measure different strengths of electromagnetic emanation generated by different amounts of voltage are running through a wire. Capacitors or fans producing noise are an examples of sounds that might provide side channel information for an acoustic attack. Error messages can sometimes be exploitable if they reveal secret-dependent information.

2.4 Tenex Password Bug

Alan Bell discovered that the Tenex computer system had a password checking bug. Tenex had an early virtual memory implementation that could be exploited. It initially compared characters one at a time, and users could set a flag to produce an interrupt if a specific memory page was accessed. With these tools, a user could test if the password-checking program would progress to a given character by checking if the interrupt happened. Now this allowed a malicious user to test a user's password character by character.

2.5 Timing Analysis of Keystrokes and Timing Attacks on SSH

This is an example of a more complex side channel attack. It was described in a paper by Song, Wagner, and Tian [**timing˙analysis˙ssh**]. SSH is a protocol that allows a user to establish an encrypted connection to a remote system. In typical usage, a user is typing commands into the remote shell, which are transmitted over the network. In order to keep the system responsive, keystrokes are sent as individual encrypted packets to the remote system.

The idea behind this attack is that an attacker can sit on the network connection and look at the network traffic to see when packets are sent from the user's computer to the remote system and measure the amount of time between packets. Naturally, the amount of time between different keystrokes depends on the actual keystrokes themselves for a given person. For example, typing the characters "as" will take a different amount of time than typing the characters "mq." Thus, it is possible to build a model of inter-character keystroke timings for a given person. This model would allow the attacker to predict the most likely pair of characters based on timings between the actual packets. The 2001 paper described using Viterbi decoding to produce the most likely sequence of keystrokes corresponding to a given set of timing meaurements, but we also discussed in class that it would probably be possible to use machine learning to improve the performance of this attack today.

This kind of side-channel attack is difficult to defend against. When designing the SSH system, the threat model is that an attacker is looking at the traffic between the user and the remote system. To prevent them from seeing the traffic, packets are encrypted. However, the timing of the packets, a side-channel, is outside the abstraction of the encryption. Possible protection strategies include adding noise to the data, adding a delay between packets to break the model, or even trying to add dummy traffic. However, none of these protections are 100% perfect, and adding noise and delay might degrade the experience for the end user.

This kind of attack can further be seen in Fabian Monrose's papers Hookt on fon-iks [**timing˙analysis˙voip**]. Ten years ago, codecs being used for encoding audio transmissions in video conferencing software like Skype were explicitly specialized for voice. However, the packet timings and packet length depended on the actual sound being transmitted. Various papers described being able to detect the language being spoken based on the packet timings and the packet lengths. Later papers described being able to complete a phonetic reconstruction of the language being spoken. From there, letter combinations could be figured out, and the audio could be read off as the actual words being spoken.

2.6 Power Analysis Attacks

Another example of a side-channel attack is a power analysis attack which can be used to break cryptography. Most of the work regarding side channels today has to do with breaking cryptography. The first paper on power analysis attacks was published in 1998 by Kocher, Jaffe, and Jun [**power˙analysis**].

The image below depicts a simple power analysis performed on a cryptographic circuit. The cryptographic circuit was implemented such that the power consumption was different based on if there was a 1 or 0 in the secret key. Analyzing the power consumption of the circuit allowed one to read off the secret key's bits.

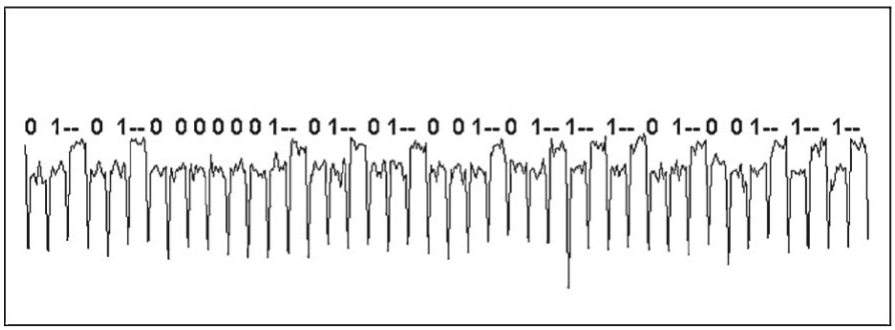

Figure 1: SPA leaks from an RSA implementation [**power˙analysis˙diagram**]

2.7 Acoustic Attacks

Acoustic attacks arise because of the fact that capacitors within computers make noise when discharging. A paper written by Genkin, Shamir, and Tromer describes this phenomenon and how it can be exploited [**acoutstic˙attacks**]. Data can be gathered by pointing a parabolic microphone at a vulnerable machine performing cryptographic computations.

The acoustic data gathered by the parabolic microphone can be displayed visually on a graph in which the actual bits can simply be read off, similar to the power analysis attack described above.

It is important to note that in most cases, acoustic attacks end up being fun for research purposes but not real threats. However, acoustic attacks highlight physical artifacts produced by software of a computer that is audible from feet away from the actual device.

2.8 Browser History Sniffing

Another example has to do with browser history sniffing. For web browsers, the default behavior is that unvisited links are colored blue while visited links are colored purple. However, the style attributes of text and links were available to scripts via the DOM. This allowed malicious websites to create an invisible iframe that would enumerate through a set of URLs and use the attributes of the links to determine whether the victim had visited the website or not. The victim's browser history is thus able to be assembled.

A number of malicious websites have exploited this as data such as this is valuable for ad targeting. This issue has been fixed in modern browsers preventing browsers from accessing the style and visited attribute of links. However, this particular threat of information leakage is still challenging to eliminate as there have still been attacks exploiting this vulnerability even with the countermeasures implemented by modern web browsers.

3 Active Side Channels

Active side channels are mainly fault attacks that can induce computational errors that may leak vulnerable information. One example of this is an attacker radiating a chip in order to flip a bit. Faults can be induced by many things – glitch power, voltage, the clock, varying temperature, and subjection to light and/or EM radiation (this can be done by pointing a laser at a machine or by sending radio waves).

3.1 Using Memory to Attack a Java VM

A clever example of a historical fault attack is a paper by Appel and Govindajhala that used memory errors to attack the Java Virtual Machine. They produced a Java heap overflow via glitching the address of a function pointer. To cause the memory errors that resulted in incorrect pointer addresses, they aimed an incandescent desktop lamp at the memory of a computer. When DRAM increases in temperature, the probability of a bit flip increases. The malicious program was written so that a bit flip would cause an overflow with 70 percent probability. The end result was a Java VM escape from mildly heating up a computer [**attack˙mem**].

3.2 Types of RAM

There are two main types of memory/storage: "Volatile" and "Persistent." Memory is considered volatile when the data can be retained only as long as the power is on. In

comparison, storage is considered persistent when data can be retained without power (like flash or magnetic disks).

3.2.1 SRAM

SRAM, or Static Random-Access Memory, is volatile memory that retains its bit values as long as power is on without a refresh. SRAM carries several advantages, such as being faster and lower density than DRAM. However, it does come with a higher cost. In addition, it exhibits a 'burn-in' phenomenon: on startup before being written to, bits are more likely to flip to a value that they "remembered". Essentially, if a bit was set to a value consistently over a period of time, the bit will be more likely to flip to this value when powered on. This might reveal secret values that were stored in memory (like cryptographic keys in an ATM). Because of the 'burn-in' effect, the bits will likely flip to their most remembered value, and the keys can then be read.

3.2.2 DRAM

DRAM, or Dynamic Random-Access Memory, is the most common RAM for normal modern computers. It is constructed from an array of capacitors, and requires periodic refreshing to retain stored data while the power is on. One benefit is that it contains a higher density of memory at a lower cost. There are a few examples of hardware attacks exploiting the physical properties of DRAM.

3.2.3 Cold Boot Attacks

This is one type of attack that exploits the decay rates of the capacitors in DRAM. In particular, from the paper, "once power is removed from the RAM, the data that it holds is removed within seconds or minutes, based on the temperature of the RAM. The colder the RAM chips are, the longer they maintain their data. By spraying the RAM with compressed air, the RAM can reach temperatures that allow for data to remain for hours. This allows an attacker to remove the RAM from a stolen computer, quickly freeze it, and then extract the keys that remain in memory" [**cold˙boot**]. The demonstration attack read cryptographic keys out of a DRAM chip that an attacker had physical access to.

3.2.4 Rowhammer Attacks

Another type of attack that can be done on a DRAM is glitch attacks. Cells in memory are generally groups in rows, and at the time of refreshing, all cells are refreshed together. For modern DRAM chips with high densities, it has been observed that the opening or closing of a row consistently over the refresh interval can cause errors in adjacent rows. This can be exploited by an attacker running a malicious process on the same machine as the victim [**rowhammer˙attacks**].

4 Covert Channels

If Side Channels are artifacts of implementation, then Covert Channels use these artifacts on purpose. An example of this can be described in the following situation: If a malicious program produces a side channel, it can send encoded information across a trust boundary. When the information is received by the other side, it can then be decoded and used. Example covert channels include variations of time, memory usage, and even cache attacks. In general, it can be very difficult to protect against covert channels if they are within your threat model.

4.1 Radio Covert Channels

An example of an academic demonstration of a covert channel is a Radio Covert Channel generated by memory writes. Essentially, when the CPU writes to memory, the memory bus emits radio waves. This can be used as a covert channel to cross an air gap: software can be written on the air gapped machine to perform memory operations to transfer bits, and this signal can be detected by a cell phone, which decodes the message from the airgapped computer [**radio˙covert**].

5 Cache Attacks

As discussed in the previous lecture, operating systems provide isolation between processes on the same computer, and a hypervisor provides isolation between virtual machines running on the same computer. However, this separation is at the software level and not at the hardware level. The programs are running using the same CPU, the same CPU memory caches, and the same physical RAM, even though the OS and hypervisor provide an abstraction of process and VM memory to the software.

Accessing main memory in RAM is relatively slow, so hardware CPU caches were implemented to optimize memory accesses by storing frequently used data closer to the CPU. Caches are much smaller than main memory; therefore, data is constantly evicted to make room for new data. Caches are a shared memory resource, so unlike page tables where each process has its own, caches are shared amongst processes.

Many cloud service providers, such as Amazon's AWS, allow different clients to use their computers for web servers. Many of these web servers run on the same physical hardware and are only separated based on the abstraction of virtual machines through a Hypervisor. Because these VMs are on the same hardware, they share the same cache memory. An attacker VM can use this shared cache memory to observe side-channel information that might reveal information about victim VM's data. Below are three techniques an attacker can use to infer details on a victim's private data through cache side-channel attacks.

5.1 Evict and time

Cache implementations use eviction policies like "Least Recently Used" (LRU) at a hardware level to manage the cache while maximizing temporal and spatial locality automatically.

An attacker can take advantage of this mechanism by making a series of consecutive reads from addresses out of a victim's address range. This puts the cache in a reproducible state where the victim will start with a string of cache misses that allows the attacker to deterministically figure out what blocks of data the victim requests during run time. From there, the attacker can tweak the victim's parameters and/or evict specific blocks from the cache to narrow down the branching patterns and eventually learn the secret.

5.2 Prime and Probe

This technique focuses on measuring the run time of the attacker process instead of the victim process. To learn the victim's secret, the attacker needs first to establish a baseline run time of their process in the case where the cache contains data relevant to the attacker's process. After running the victim, blocks have a chance to be evicted from the cache depending on the control flow of the process. Any slowdowns in the attacker's process mean that the victim needed to overwrite a block that the attacker cached. Just as before, the attacker can then employ other tactics to learn the secrets eventually.

5.3 Flush and Reload

The attacker can intentionally flush out the cache, meaning that the attacker would run into a string of cache misses the next time the process is run from the same initial state. However, if the victim loads a block into the cache that the attacker shares, the attacker will experience a cache hit that will speed up execution time.

6 Mitigating Side Channels

Side channels can hard to eliminate entirely, but there are several best practices that can be used to mitigate common side channels. These include constant-time programming techniques, eliminating secret-dependent executions or branches, and hiding or blinding inputs.

6.1 Constant Time Programming

The idea behind constant-time programming is simple: eliminate timing variations by writing a program that always takes the same amount of time to run. However, this is much harder than it might seem at first glance, for a number of reasons.

First, even if your code itself is written to perform exactly the same number of operations no matter what input it is provided, the *operations themselves* may take different amounts of time for different inputs. This is because some CPU instructions take different amounts of time depending on operands. Here is an example:

```
void foo(double x) {
    double z, y = 1.0;
    for (uint32_t i = 0; i < 100000000; i++) {
        z = y*x;
```

```
    }
}
```

The function call foo(1) would run faster than foo(1.0e-323). Since there are special microcode instructions dealing with special cases of subnormal floating-point values, executing foo(1.0e-323) would end up going into microcode instead of purely in hardware. The for loop amplifies the timing difference. This is an unusual kind of CPU behavior that would enable attackers to observe time differences in execution that could be exploited in attacks. In order to prevent cache attacks or time attacks with constant time programming, a defender will need to avoid all variable-time instructions like sub-normal floating point values.

6.1.1 Modular exponentiation

```
m=1
for i = 0 ... len(d):
    if d[i] = 1:
        m = c * m mod N
    m = square(m) mod N
return m
```

This is an implementation of modular exponentiation using the square and multiply algorithm. The execution time for this code is dependent on the number of bits in d[] that are 1. For example, if d[] stores a secret key, the code's execution would leak some information regarding its values. This is a classic example of an algorithm that is vulnerable to a side-channel attack but is often run on cryptographic secrets. (We will talk more about this when we get to cryptography.)

6.1.2 If-statements on secrets are unsafe

At the code structure level, an if-statement on a secret is unsafe because the execution time depends on the value of secret and can leak information to attackers. In the code structure below, if the secret value is true, it would execute 4 instructions (i.e. s0, s1, s2, s3). Otherwise, it executes run 2 instructions (i.e. s0, s3).

```
s0;
if (secret) {
    s1;
    s2;
}
s3;
```

We might consider padding the else branch to allow either branch to take the same amount of time:

```
if (secret) {
    s1;
```

```
    s2;
} else {
    s1';     // where s1' takes the same amount of time as s1
    s2;
}
```

However, an attack could potentially still perform a cache attack to figure out which instructions were loaded from the cache. With this information, an attack would be able to know which branch was taken. Similarly, an attacker can also observe the success or failure of branch prediction to extract information.

To avoid this, one can use arithmetic techniques to mitigate if-else statements. Consider the following example where code block A can be replaced with code block B:

```
// A - original
if (secret) {
    x = a;
} else {
    x = b;
}

// B - replaced with the following
x = secret * a + (1-secret) * x
x = (1-secret) * b + secret * x
```

Overall, writing constant-time code is complicated. Although there are tools to help, most code is still written by hand. It can be slower, larger, and more complex. Thus, people generally recommend not doing your cryptographic implementations. Thankfully, as students, we shouldn't be too worried about side-channel attacks unless we work for the government.