Side-Channel Attacks: A Best Practice Guide

This guide has been written to be used by academics, professors, and students with the intention of educating and informing them on the basics of side-channel attacks, along with the various mitigation strategies for each category of attack. The purpose of the guide is to provide a free source of information on the topic of side-channel attacks, as well as providing a professional repository of sources that can be freely referenced.

In this guide we will explain the basics of a side-channel attack in general, and then go into specifics about five different subcategories of attacks, each being highly detailed and having accompanying papers referenced with them. All attack types will have various mitigation strategies and the effectiveness of said strategies.

The author of this guide is Tom Harris, a MSc student at the University of South Wales who is greatly interested in the topic of side-channel attacks. This guide was written as during his time in university there was not as many easy-to-read resources on the subject of side-channel attacks and a guide that allows easy reading and understanding with a significant number of professional references included allows those who may be in the same position in future would have a greater understanding of the topic.

What this guide includes:

What are Side-Channel Attacks?

A side-channel attack is a security exploit that an attacker can take advantage of in an attempt to steal secrets (or cryptographic keys) from either the system, or CPU itself. This guide will focus on CPU-based attacks and areas of the CPU that are targeted.

There are many concepts within a CPU that can be targeted to perform a side-channel attack, some attacks for example target the noise a CPU produces (part of the **Acoustic Cryptanalysis** attack family), while others look at the resources assigned to a particular processes within the CPU (called the **Resource Allocation** family). All attack families target some concept and generally use a standard resource or output of a CPU and then use this as the vector for retrieving what should be sensitive information. Furthermore, while a large degree of this guide is research and theoretical based, there will be links and resources available hosted and distributed through other formats that consist of a more practical approach, one of which is a YouTube video which demonstrates the capabilities of Spectre (a side-channel attack that uses the speculative execution process within the CPU). Finally, an interactive and educational activity through the form of a TryHackMe room (an online hosted service that allows paying users to create ‘rooms’ which can be used to learn tools and techniques of computer and cyber security), in which participants will read over various resources, look at source code, and watch videos to answer various questions on Spectre (as this has the simplest implementation of a Side-channel attack).

All attacks that are mentioned in this guide have also had some form of mitigation included, these may be relevant to manufacturers or to standard users although all are relevant to researchers and/or other academic staff.

Families of Attacks

This guide uses the term *family* to describe any collection of attacks that function in a similar fashion or target the same element of a device/CPU. While there are many families (and subfamilies) of side-channel attacks, this guide will go over a select few as to not over-saturate the guide with less than adequate detail on every attack and family. Within each family of attack, one or two case studies (peer reviewed papers/proof of concept) will be used to provide an understanding, as well as recommending various mitigation strategies.

Acoustic Cryptanalysis Attack

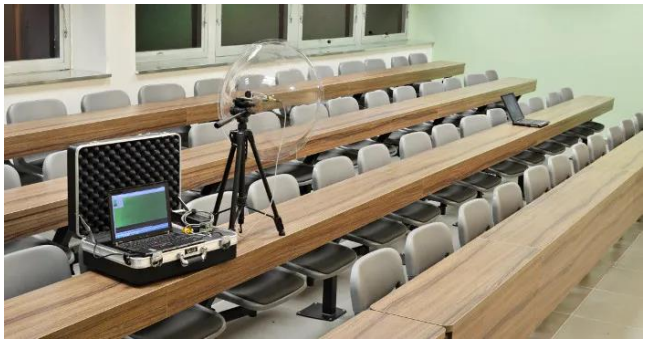
Acoustic cryptanalysis, meaning ‘solving cryptographic sequences using sound or relating it to sound’, is a family of side-channel attacks that utilises sound when attempting to break a cryptographic sequence. Some attacks within this family will target a device that uses cryptography (such as a CPU) with concentrated, low frequency vibrations in an attempt to disrupt the internal clock, allowing data leakage to occur. Other, more common, attacks use the noise produced by a device such as a CPU whilst it performs particular operations to calculate the operation being performed, leading to a discovery of the cryptographic sequence being used (such as RSA, AES, Diffie-Hellman Key Exchange).

Attack in Practice

For this guide, the main form of acoustic cryptanalysis included will be those that analyse the sound produced, which is best defined by Shamir and Tromer (2007) as a way to ‘exploit acoustic emanations from modern computers, wherein the power circuitry creates vibrations that are modulated by CPU activity’. To understand this attack, Shamir and Tromer (2007) state that ‘RSA signature/decryption sounds different on different secret keys’ which shows that even within a cryptographic method, different secret keys can be differentiated by different sounds.

Mitigation Strategies

Mitigation techniques for this family of attack are generally physical or hardware related, such as the addition of a physical access control to lower the sound produced by the CPU or as Shamir and Tromer (2007) state ‘careful circuit design and use of high-quality components’. There are some lesser software mitigation strategies that can be implemented such as, ‘Parallel Software Load’ (Shamir *et al* 2016) which means ‘other computations performed in parallel will somehow mask the leakage of the decryption operation’ although Shamir *et al* (2016) does state that this countermeasure ‘actually might *help* the attacker since the lower leakage frequency is, the more sensitive the microphone capsule that can be used to perform the attack’. Finally, an algorithmic approach to mitigation, through ‘Cipher Text Randomization’, effectively prevents this type of attack as it takes the ciphertext (*c)* before decryption and generates a random 4096-bit value (*r)*, and multiplies it to a publicly available power (e.g. $c = r^e$), which is then decrypted using $r^e \cdot c$ and then multiplied by r^{-1} which leads to the ‘value sent to the modular exponentiation routine is completely random’.



There is generally no concern required regarding this family of attacks (for casual CPU users) as the equipment required to monitor the variations of sound produced by CPUs are quite sophisticated and require a large space, which a general user is unlikely to have where their PC is. This vulnerability can be considered for more large-scale systems like a bank or datacenter, in which equipment can be moved and used in an attempt to break the cryptographic sequence.

Timing Attack

A timing attack analyses the time taken to perform a cryptographic sequence/operation and use this information in first identifying the operation used, leading to discovering the secret key used, and finally decrypting the encrypted information/data. There are numerous mitigation strategies for this attack although they differ from the previous acoustic cryptanalysis family as they are primarily software related, as opposed to the physical mitigations previous. Research by Kocher (1996) showed the strength of these forms of attacks as rather than take advantage of any inherent fault or weakness in the system, it ‘gains information from the implementation of a cryptosystem’ as stated by Wong (2004).

Attack in Practice

Wong (2004) goes on to explain that ‘Timing attacks exploit the timing variations in cryptographic operations’, meaning that each variation of operation (depending on the input and secret parameter value) has a different length of time associated with the computations (caused by the performance optimisations within these devices, such as CPUs). The timing attack example in this guide follows the Kocher (1996) example provided by him in his paper, ‘Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems’ in which he explains the methods of attacks against various cryptographic operations. For the purpose of this guide however, the methods (along with equations) are not the focus, rather than the theory behind the attack, along with effects and mitigation techniques.

Kocher (1996) explains that ‘the attack can be tailored to work with virtually any implementation that does not run-in fixed time’ and that there is a sole

requirement for beginning any of the attacks, calculating x (also referred to as the secret key). Both Diffie-Hellman and RSA can be attacked in a similar manner as Kocher (1996) explains that both are defined as 'private-key operations' which, at their core, 'consist of computing $R = y^z \bmod n$'. Within this equation, the n is known by all, also referred to as public, y is discoverable by an eavesdropper, after which the attacker's primary goal is the discovery of the value of x, the secret key. For the attack to succeed, the victim must compute $y^z \bmod n$ over several iterations, with the value of y changing each time. Within these several iterations, the attacker will be privy to y, n, and the computation time (this can be prevented however, if new secret exponents (key) of x are chosen for each iteration). Using the above calculation, any non-fixed time cryptographic operations can be attacked with a timing attack, allowing any attacker to retrieve the secret key x, using it on the cipher text and then retrieving the data that was initially secured, this allows virtually any similar cryptographic operations to be broken by retrieving the secret key all through simply applying known values and the computation time into an equation.

Mitigation Strategies

Mitigations for this family of attacks have been proposed by a number of researchers, as well as being implemented by Intel in their CPU microcode updates. One mitigating strategy is referred to as the 'Shannon Mitigation Problem' as stated by Tizpaz-Niari *et al* (2019) which introduces a form of 'synthetic delay' into the programs. This, however, is described as impractical as 'it may result in unacceptable performance degradations of the response time'. Other mitigation strategies apply strictly to web-based applications and the operations they perform within the CPU as many timing vulnerabilities exist within web browsers (and many individual sites). The developers of the browser Chromium have implemented a form of isolation that significantly reduces the amount of information that is accessible to a website and as such, if a side-channel attack occurs any information that may be leaked from the side-channel is only relevant to the site and not to the wider device as a whole. Finally, Intel has implemented a number of microcode updates which have patched vulnerabilities involving side-channel attacks, particularly those that rely on the timing of operations (or variables), and the calculator of secrets. Timing vulnerabilities have been patched by ensuring the value that authenticates the cipher text (read secret key), also called the MAC value (Message Authentication Code) is compared to the expected value in its entirety rather than just a partial match (as originally an attacker would only need a partial value match, such as the first 4 bytes) which as Intel state, 'forces the attack to become a brute force search, which is orders of magnitude harder.'.

Cache Attack

A Cache attack is a broad term for an attack that accesses leaked information using a side-channel such as the CPU cache or cache memory. This information can be leaked by taking advantage of vulnerabilities within the micro-architecture of a CPU, with two notable attacks having been designated Spectre and Meltdown. This guide will outline the basics of a general cache attack, while going into further detail into both Spectre and Meltdown as specific examples of cache attacks.

Attack in Practice

A cache attack, like many other attacks, makes use of a side-channel within a device to leak particular information. Some of these channels are described as 'covert channels' which are communications channels that, as Mushtaq *et al* (2020) states, are 'not intended or designed to transfer information between a sender (process) and a receiver (process). These channels can be used to target the confidentiality of a system, as is the case when a program that runs inside Intel's SGX Enclave (which is considered the sender), is able to change the 'processor cache state based on some secret information it is processing' (Mushtaq *et al* 2020). Alongside physical attacks (which require access to the devices being attacked), Wang and Lee (2007) have written a paper on software-cache attacks, particularly on the prevention of said attacks. As they state, 'The attacks are easy to perform, effective on most platforms, and do not require special instruments or excessive computation power.' which show these types of attack as being a significant danger as they can be delivered either through a custom-tailored program, or as a part of another, existing program. Examples of software-based side channel attacks which target the cache of a computer device are the Spectre and Meltdown attacks, both of which can be run as a program on the target device (meaning anyone with access, both remote and local) can perform this attack and steal potentially sensitive information.

Spectre, as an attack, targets specific memory addresses that store information temporarily to leak said information. This guide takes the Spectre PoC (Proof of Concept) created by Paul Kocher, various peer-reviewed papers, and the cyber security learned platform 'TryHackMe' and creates a custom room to allow you to learn specifically about Spectre, answer questions on its technicalities, and learn the theories behind attack. Below is a link to a video demonstration of Spectre, this video shows the execution of the Spectre exploit on a CPU (specifically an i5 9600KF), and the leakage of information from a number of memory addresses (the information being 'This is the secret data in the cache!'. Spectre works by taking advantage of a particular functionality of the CPU called the *speculative execution*. This is a performance feature of most modern CPUs which, rather than waiting for each queued instruction to execute, the CPU uses a part of the microarchitecture called the *branch predictor* to predict the next few instructions, and then speculatively execute them, although this execution is considered *transient* as if the prediction is considered 'incorrect' the execution is reversed. This is the point at

which the Spectre exploit takes advantage, an attacker will code the Spectre program to train the *branch predictor* to mispredict the upcoming predictions, which then get reversed once executed, although the result of said instructions does not get reversed, meaning any information that was used by a process that used the instructions is then passed into the cache, which is then leaked by the Spectre attack.

Meltdown differs from Spectre, which targets arbitrary memory locations during the execution of a program, as it targets the whole of the memory during the execution of a program. While the section on Spectre included a demonstration, this section on Meltdown provides a link to the GitHub repository which contains the PoC of meltdown. While Spectre utilises *speculative execution* and *branch predictor*, Meltdown takes advantage of the fact that errors (exceptions) are only 'raised', which means to be visible within the microarchitecture of the CPU, when the instruction that created the fault is 'retired' or reversed. By using this, *transient* (see above) instructions that are queued ahead of the current instruction are able to be executed, basing their execution on results of instructions that throw a fault. This can lead to faulting instructions to create unauthorised results which then lead to the execution of unauthorised instructions. All incorrectly executed instructions are then retired in the CPU's 'in-order instruction-retirement mechanism to discard any architectural effects' although in many cases secrets are leaked through covert microarchitectural channels.

Mitigation Strategies

To mitigate such cache attacks there are a number of studies available, one of which is written by Page (2003), states that one method of mitigating chase-based side-channel attacks is by removing the cache in those particular systems, however Page (2003) does state that this 'is perhaps unrealistic since its inclusion in the first place implies it is required from a performance perspective'. A further mitigation outlined by Page is to 'Disable cache flushing', which is an essential part of the Spectre side-channel attack. This will prevent Spectre from executing fully as one essential aspect of the source code is the exploitation of 'clflush' which results in the retrieval and then removal of data stored in the cache. Along with the internal architectural mitigations mentioned previously, a physical mitigation strategy is also possible which, as Page (2003) states, involves adding a shielding element to the device in question, 'smart card processors may be packaged in such a way as to make attacks against them harder' being one example of simple yet effective physical mitigation. Finally, the implementation of an 'Order skewing' system within a processor, which are performed by devices called 'non-deterministic processors'. This involves running and executing instructions in a random, parallel order whilst retaining data dependencies and relationships between each instruction. As each cycle of the processor will have different sets of instructions run each time, the prediction (Spectre) or fault raising (Meltdown) will differ each time, making the attacks significantly harder to perform.

Power Analysis Attack

Power analysis attack against a processor relies on the power consumed to complete a particular process or cryptographic sequence. By analysing the power used in the encryption or decryption of a set of data, it is possible to calculate the cryptographic operation used. For example, Gamaarachchi and Ganegoda (2018) demonstrated this form of attack against a cryptographic algorithm created by the NSA (National Security Agency) called Speck, which they state was able to be broken in 'less than an hour'.

Attack in Practice

The sub-category of attack they use to attack Speck is called 'Correlation Power Analysis', or CPA, which analyses the 'correlation factor between the power samples and the Hamming weight' (Brier *et al* 2004). The power samples are the power readings recorded from the CPU whilst the 'Hamming weight' refers to the number of 1's in a string of bits (Henry Warren 2002). During the first step of the CPA attack, a value (*I*) is 'generated during the cryptographic operation (encryption or decryption)' (Gamaarachchi and Ganegoda 2018), using this, the following equation can be devised $I = f(d, k)$ in which *f* is the name of the function and *d* refers to the data (both cipher or plaintext) and *k* refers to the key-part (otherwise known as a subkey). Using the power measurement instruments, 'several hundred (or thousands depending on the system) of samples for *d*' are collected, which then have their intermediate (*I*) values calculated along with the 'power consumption for those intermediate values using a power model such as hamming distance model'. The hamming distance model is similar to the hamming weight, although there is a key difference, hamming weight looks at all values within a string of bits that are 1's, while the hamming distance looks at the number of bits that differ from one another within two binary strings (NIST 2006). This whole calculation is then repeated for all possible values of *k* (the subkey), at which point the 'real' and 'hypothetical' power consumption values are compared to discover the subkey with the highest correlation, allowing the precise and correct subkey to be discovered leading to the secret key being leaked, which is when the information has been encrypted can then be decrypted and leaked through a side-channel.

Mitigation Strategies

Gamaarachchi and Ganegoda (2018) have proposed a number of mitigations and countermeasures to power analysis attacks, in both software and physical (hardware) context. While they had tested several different potential mitigation strategies, they had discovered that some so-called 'mitigations' actually enhanced the power analysis attack. One more effective method of mitigation used a piece of hardware called an 'operational amplifier' which is a device introduced into a circuit to amplify the output by several hundred thousand times the difference between the inputs. By doing this, the number of power traces (see samples) required to perform

the attack increases from 50 to 4000, as well as the time taken to perform the attack from ‘5 minutes to 3.5 hours’. While this was a somewhat effective mitigation strategy compared to others within Gamaarachchi and Ganegoda’s (2018) paper, far more effective methods based in software were proposed. One attempted provided ‘reasonable immunity against power analysis attacks for its cost and simplicity’, this being ‘Random instruction injection’ which allowed a user to run random, arbitrary code or instructions that changed the power usage of the CPU, whilst avoiding any impact on the performance of the CPU. This ensured any attacker analysing the power output would receive ‘incorrect’ power samples and thus would incorrectly guess the secret key and encryption method.

These are some of the more effective methods although many do exist, particularly in the software context. If these methods are followed, those attempting to fully harden their cryptographic device/CPU against a power analysis attack would see a significant security increase at a (relatively) low impact to performance.

Allocation Based Attack

An allocation-based side-channel attack analyses the resources allocated to a particular cryptographic operation and uses this knowledge to determine both the sequence used, and then the secret key being implemented in order to leak the potentially sensitive data through a side-channel. This guide uses a study by Angel *et al* (2020) that shows the capability of an allocation-based side-channel attack through the use of PRAs (Private Resource Allocators), which they define as a new type of *cryptographic primitive*. A cryptographic primitive is ‘a low-level cryptographic algorithm used as a basic building block for higher-level cryptographic algorithms’ (NIST 2020).

Attack in Practice

Angel *et al* (2020) proposed that most, if not all, side-channel attacks expose information ‘as a result of a process in the system consuming a resource (e.g., sending a network packet, populating the cache, executing a conditional branch instruction)’. The key element of a system involved in this attack is the ‘resource allocator’ of the system, which among other things includes, ‘cluster managers, network rate limiters, storage controllers, data centre resource managers, flow coordinators, managers, etc.’, all of which contribute to the opportunity for the resource allocator to leak information about ‘how many (and which) other processes are requesting service through the allocation itself. This attack is extremely dangerous in the context on messaging services defined as ‘bidirectional anonymous messaging’ (otherwise known as metadata-private-messengers, or MPMs). This is because there are opportunities for an attacker to gain access to a great number of conversations just through compromising one contact as if one user’s contact is compromised, the attacker can learn about *other* conversations besides the one involving the compromised user.

Angel *et al* (2020) states that ‘Traffic analysis makes things worse’, providing an example which excludes users that are considered *idle* (those not partaking in conversation or the sending of information) by analysing the traffic of compromised users, and separating those that are busy from those that are not. Angel *et al* (2020) then went on to further separate users by the *round* agreed upon by both sender and recipient sessions (a round is the time agreed upon between sender and recipient to start the conversation). Channel allocation is capable of leaking information also, as within the MPMs that the PRAs work, there are *n* number of friends/contacts and *k* number of channels, with there generally being far more *n* to *k*. As the *n* outnumber the *k* should a contact attempt to create a session with a user, the current state of the receiver can be inferred from this side-channel, the information being leaked possibly being a busy signal, or indeed no response at all. This allows the inference that the receiver is contacting other users at the time of the call, which allows the opportunity to compromise further conversations (as mentioned previously, with one compromised user meaning many compromised conversations).

Mitigation Strategies

Along with the greatly detailed research on the proposed attack, Angel *et al* (2020) provides a significant mitigation against these forms of attacks. The basis of their paper is the implementation of the cryptographic primitive called a PRA (Private Resource Allocator) which ‘can be used to allocate resources (e.g., network bandwidth, CPUs) to a set of clients without revealing to the clients whether any other clients received resources’. They propose several iterations of this primitive that ‘provide guarantees ranging from information-theoretic to differential privacy’. They go on to outline the benefits of using this mitigation in regard to the allocation based side-channel attack as PRAs will increase ‘the network resources required to start a conversation by up to 16x (can be made as low as 4x in some cases), but add no overhead once the conversation has been established’. This shows that by using a PRA, the resources can be privately assigned to a process without fears of the information being leaked through a side-channel (although as stated the increase in resources can be up to 16x to the original amount required to start a conversation using Alpenhorn) with little to no overhead.