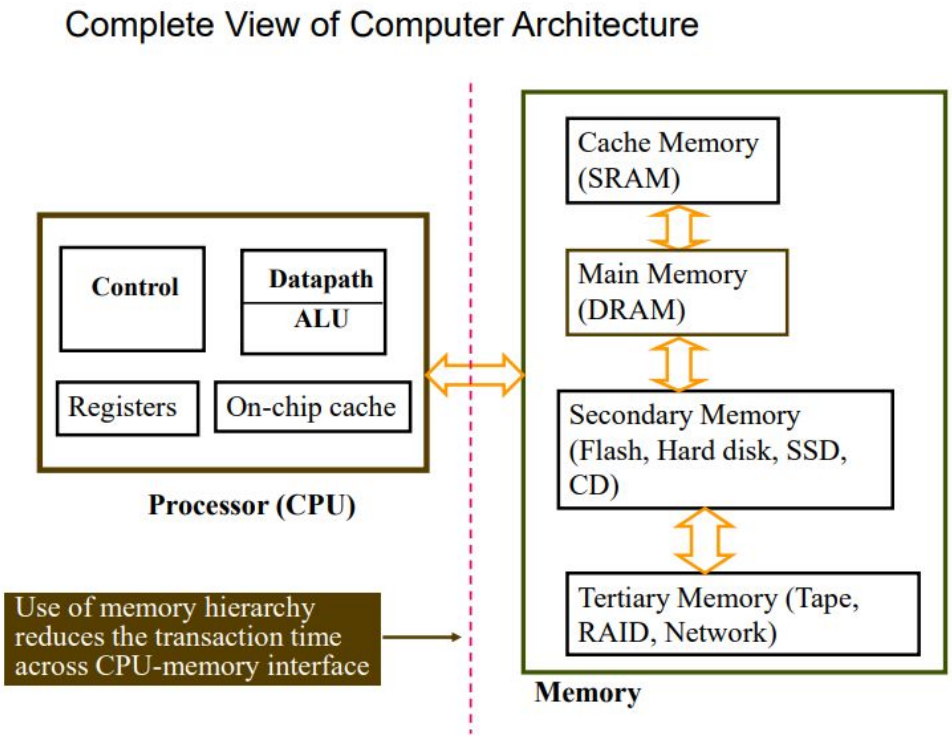
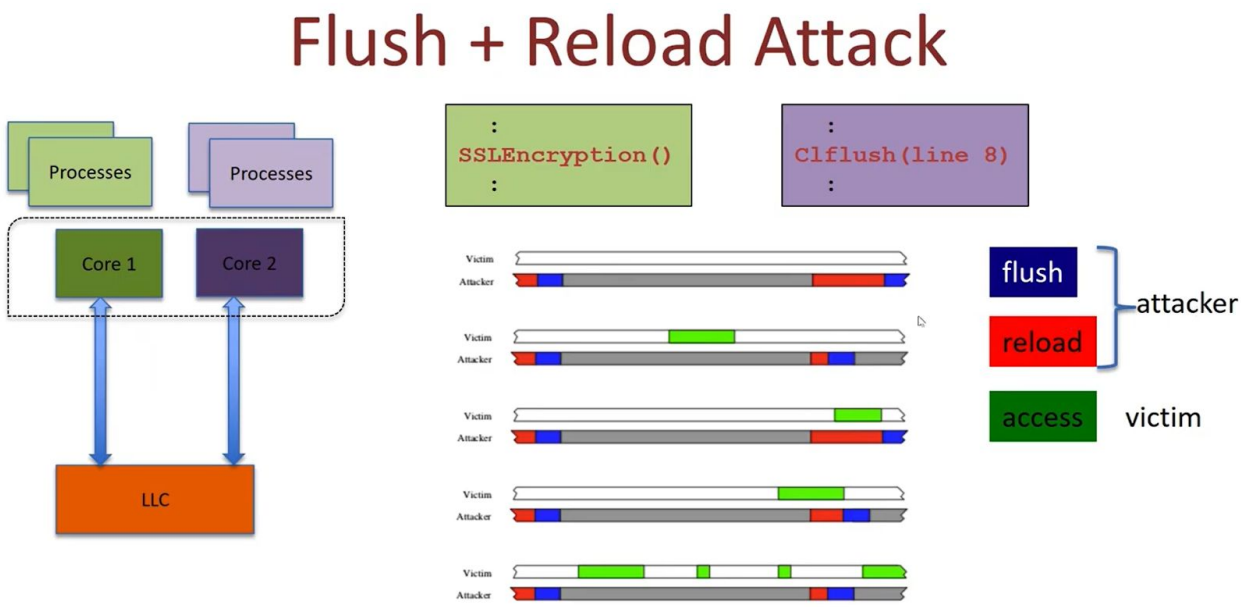
SECURITY ANALYSIS OF DEEP NEURAL NETWORKS OPERATING IN THE PRESENCE OF CACHE SIDE CHANNEL ATTACKS

Generally instructions (program) and data are stored in the same memory.

Memory is inherently slower than logic. The time spent in memory accesses can limit the performance.

architectures allow frequently used operands to reside in on-chip registers.





DeepRecon : an attack that reconstructs the architecture of the victim network using the internal information extracted via Flush+Reload, a cache side-channel technique. Once the attacker observes function invocations that map directly to architecture attributes of the victim network, the attacker can reconstruct the victim's entire network architecture.

As opposed to attacks that exploit vulnerabilities in software or algorithm implementations, side channel attacks utilize information leaks from vulnerabilities in the implementation of computer systems. Due to modern microprocessor architecture that shares the last-level cache (L3 cache) between CPU cores, cache side-channel attacks have become more readily available to implement. The fundamental idea of the attack is to monitor the access time to the shared contents, e.g., shared libraries or credentials, between a victim and an attacker while the attacker fills the cache set with

flushing the shared data from the cache. Once the victim accesses memory or shared data, the attacker can identify which memory addresses or shared data is accessed.

The technique assumes that an attacker can run a spy process on the same host machine. During monitoring, the attacker repeatedly calls the clflush assembly instruction to evict the L3 cache lines storing shared content and continually measures the time to reload the content. A fast reload time indicates the data was loaded into the cache by the victim whereas a slow reload time means the data is not used. From this information, the attacker determines what data is currently in use and identifies the control flow (order of function calls) of the victim's process.

Here we assume that the attacker and victim use the same opensource DL frameworks shared across users. Importantly, this assumption can be easily met because many popular DL frameworks such as Tensorflow or PyTorch1 are provided as open-source libraries, and this practice of sharing libraries across users is default on major operating systems, e.g., Windows, MacOS, and Ubuntu. Thus, our attacker can identify the addresses of functions to monitor in the instruction cache by reverse-engineering the shared framework's code.

Practical Example : The attacker induces the victim to install a chrome add-on (which runs at a user level) that passively monitors cache behaviors of the model and extracts the architecture. Then, the attacker trains a surrogate model with public datasets (including malware and benign software). With

the surrogate model, the attacker crafts her malware that evades detection and can continue to craft malicious files that will be classified as benign offline and without any further observations. As opposed to common black box attacks, our attacker lowers the possibility of being caught because she only monitors the victim model while it is in use and does not need to query the model.

ATTACK EXAMPLE:

Suppose an attacker aims to install malware on a victim's machine where an anti-virus system, based on a DNN model, is running. To evade malware detection in common black-box attacks, an attacker needs to drop crafted programs actively to monitor the model's decisions and synthesize an evasive sample based on the collected data. However, when the attacker drops multiple files, her behavior can be detected by the victim. This is further amplified by the need to query the model repeatedly to craft any more malicious files.

Attacker induces the victim to install a chrome add-on (which runs at a user level) that passively monitors cache behaviors of the model and extracts the architecture. Then, the attacker trains a surrogate model with public datasets (including malware and benign software). With the surrogate model, the attacker crafts her malware that evades detection and can continue to craft malicious files that will be classified as benign offline and without any further observations. As opposed to common black box attacks, our attacker lowers the possibility of being caught because she only monitors the victim model while it is in use and does not need to query the model.

DEFENSES TO DEEPRECON ATTACK :

Running decoy processes with tiny models : DeepRecon and other potential cache side-channel attacks on deep learning frameworks can only observe that a library function is called, but not by whom. By running an extra process (i.e., a decoy process) simultaneously with the actual process, we develop a simple but effective defensive strategy. The decoy process also invokes the target functions in the shared framework, which obfuscates the architecture attributes and computation sequences.

Oblivious model Computations : Another defense against DeepRecon is to obfuscate the order and number of the computations (i.e., function invocations) observed by our attacker using oblivious model computations. We propose two approaches. First, we can update the victim's architecture by adding extra layers. To minimize the side-effects such as performance loss, these layers return the output with the same dimensions as the input. Thus, we augment the original architecture by adding such layers at the random location to make the same architecture look different in the attacker's point of view.