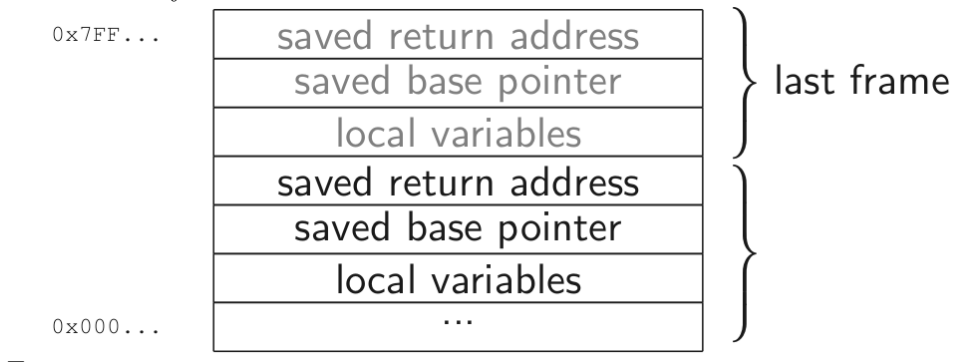


## Memory safety

- spatial violation
  - buffer overflow/read
  - out-of-bounds reads
  - null pointer dereference
- temporal violation
  - use after free
  - double free
  - use of uninitialized memory
- stack frame layout



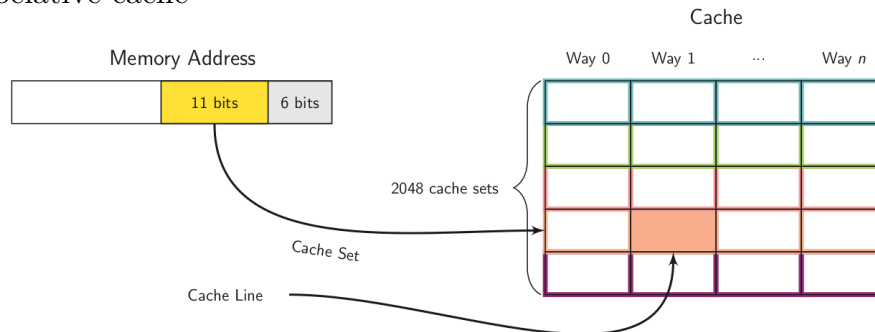
- prevent (some) buffer overflows with
  - stack canary
    - \* random number stored below stack frame and buffer
    - \* before returning check for overwrite
  - safe and unsafe stack
    - \* store buffers on unsafe stack
  - ASLR
    - \* randomizes location of memory
    - \* requires large enough randomization range
    - \* addresses must not be leaked
- control flow integrity CFI
  - Control-flow **graph** must be **correctly constructed**
  - **Function pointers** cannot be protected if destination set is large
  - Some functions (e.g., library functions) have **many call locations** and therefore return locations

## Side-channel attacks

- passively observe physical properties
- information leakage through side-effects
  - power consumption

- execution time
  - \* preventable through constant runtime and control flow
- CPU caches
  - \* fast on cache hit, slow on miss

- set associative cache



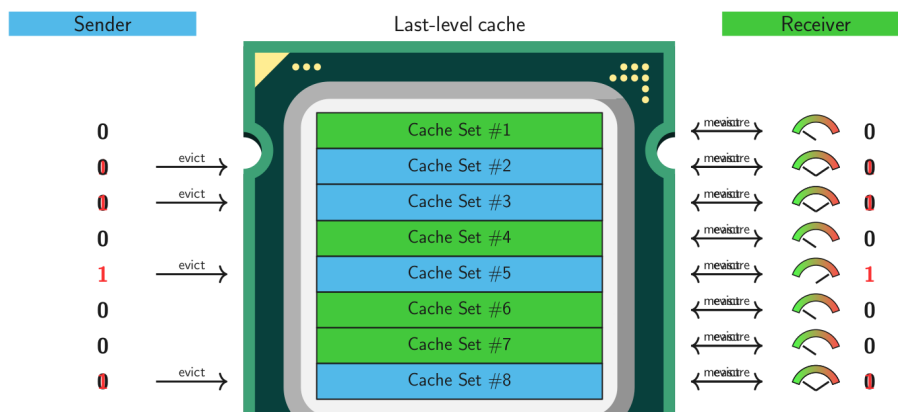
- each memory address has a designated cache set to be cached in

- Flush+Reload

- requires shared memory
- determine which memory locations have been accessed by measuring the time to access it
  - \* attacker flushes cache line
  - \* victim might access memory locations of this cache line
  - \* attacker reloads the cache line and measures the time
    - ◆ short => victim accessed the location
- keystrokes can be retrieved because they cause code execution in shared library (e.g. libgdk)

- covert channel

- two processes communicate over secret channel



## transient execution attacks

- meltdown

- read data at any address using out-of-order execution

- ```

char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;

```
- \*
    - combined with Flush+Reload
    - index of cache hit reveals data
    - preventable with KAISER/KPTI
      - \* unmap kernel pages in user space
      - \* kernel addresses are no longer present
  - NG-Foreshadow
    - leak data from L1 cache of host into VM
  - spectre
    - exploit control flow predictions
    - speculative execution runs
 

```

if (x < array1_size)
    y = array2[array1[x] * 256];

```
    - \*
      - \* array1 is a valid buffer
      - \* x is a way too big and causes an overread into the victim's memory
      - \* array2 is uncached shared memory
    - crashes because it accesses not allowed memory
      - \* still accesses the page (speculative execution) and rollbacks the operation
      - \* accessing page at array1[x] causes a cache hit

## Fault attack

- actively manipulate device to induce faults
- Row hammer
  - accesses to nearby DRAM rows cause cell to leak energy which may cause bit flips
- Glitching/Skipping attacks
  - may corrupt data/skip instructions
- Undervolting
- countermeasures
  - detect anomalies
    - \* active fine wire meshes across IC
    - \* power surge/temperature/light sensors?
  - double execution
    - \* unlikely to produce same fault twice

## Sandboxing and Isolation

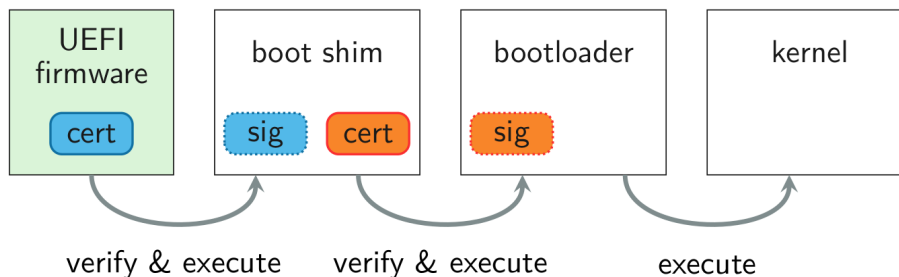
- Principle of Least Privilege

»Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job.«

- e.g. x86 Protection Rings
- drivers have higher privileges than user-space applications
  - \* only accept drivers signed by trusted vendor
  - \* root attacker cannot inject code into kernel

- Secure Boot

- UEFI ROMs, boot loader, kernel must be signed
- public key in firmware to verify signatures
- execute after verification



- Supervisor Mode Access/Execution Prevention

- prevent access to user-space data
- prevent execution of user-space code
- opposite of KAISER/KPTI

- Sandboxing

- restricted environment to execute program in
- resources strictly controlled
  - \* own filesystem
  - \* no network connection
  - \* limited memory
  - \* limited CPU time
  - \* ...
- Language-Level-Sandboxing
  - \* restrict untrusted code on the language level
    - ◆ e.g. JavaScript
  - \* no dangerous functionality (I/O, syscalls, ...)
    - ◆ ask user for permission if needed
  - \* interpreter
  - \* eBPF verifies certain properties before executing code
    - ◆ termination

- ◆ no loops/recursion
    - halting problem
  - ◆ jumps may not form loops
  - ◆ only allowed functions
- Rule-based Execution
  - \* define what an application is allowed to do
    - ◆ white/blacklists
  - \* e.g. seccomp
  - \* good policies are hard to create but secure and efficient
- Container
  - \* OS-level virtualization
  - \* isolated user-space instances
  - \* each container is assigned resources
    - ◆ memory
    - ◆ folder
    - ◆ ...
  - \* only see assigned resources
  - \* shared OS, separate libraries/dependencies
  - \* Control Groups?
  - \* Namespaces
    - ◆ isolate system resources between processes
      - cannot see other processes
      - own mount
      - own network stack
      - ...
- Virtualization
  - \* no shared kernel since process runs in own OS
  - \* emulate entire system => massive overhead
    - ◆ bare metal
      - run directly on hardware
    - ◆ hosted
      - on top of host OS
  - \* VM escape
    - ◆ access to host and other VMs
- Isolation
  - isolate application from system
    - \* trusted application on untrusted system
  - applications

- \* sensitive data
- \* distrust against cloud provider
- \* intellectual property
- \* rights management
- Trusted Computing Base TCB
  - \* CPU and firmware usually
  - \* kernel and system programs usually too
    - ◆ protected by protection rings
- Trusted-Execution Environment
  - \* secure area within CPU
  - \* guarantee integrity and confidentiality for code + data
  - \* still shared hardware
  - \* small overhead
  - \* CPU is the only TCB
  - \* memory encrypted and inaccessible to OS
  - \* does not protect against side channel attacks
- Hardware Isolation with Hardware Security Modules HSM
  - \* external, dedicated hardware
    - ◆ nothing shared
  - \* protects high-value cryptographic keys
  - \* crypto processor for
    - ◆ key generation/management
    - ◆ signatures
    - ◆ data en/decryption
    - ◆ strong RNG
    - ◆ secure timestamp