Quiz 1 (Sep 30)

1. Vulnerability and Patch (02 Security Concept, Pages 11-12)
It is important to know that a security patch contains a fix for a vulnerability. By analyzing it, an attacker can understand how to exploit the vulnerability. For unpatched systems, they can be exploited by such attackers. (On page 12, there is a gap between the patch's release and the patch's deployment: one-day attack.
A security patch helps secure the system, but when it is deployed. Patch deployment is the key to security in practice. Deployment is often as important as creating the patch itself.

2-3. Side channel (02 Security Concept, Pages 11-12)
The figure on the right describes a side channel attack. Below is the detail.
On Android, libinput.so is a shared library that multiple applications use.
Since the shared library is shared between multiple applications, if one application uses a particular code and the code is cached, when another application wants to use the same code, they will be getting the code from the cache, meaning that the access time will be shorter. The graph on the top shows which code addresses (x axis: 0x840~0x1180) are frequently used on which functionalities (y axis: key, longpress, swipe, ...). And the below graph shows that with different functionalities, the speed for accessing 0x11040 becomes different.
Specifically, it is access 0x11040. See the top graph, and swipe has the darkest block at 0x11040, meaning that this is the code block mostly accessed only by the swipe. Tap and longpress also access some, but less than swipe.
See the graph at the bottom. When tap happened, the access time drops but very quick drops. Swipe has a longer drop, which means that more portion of the code were cached. (more drop == more cache, less drop == less cache).

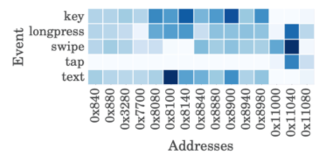


Figure 5: Cache template matrix for `libinput.so`.

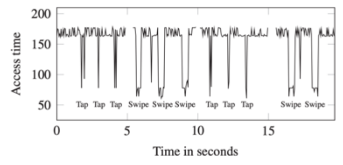
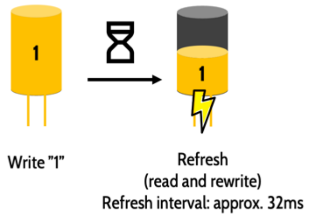


Figure 6: Monitoring address 0x11040 of `libinput.so` on the Alcatel One Touch Pop 2 reveals taps and swipes.

4. symmetric encryption vs asymmetric encryption (02 Security Concept, Pages 34-36)
While asymmetric encryption provides better security, symmetric encryption is often used because it is faster. Symmetric encryption relies only on the private key. If it is stolen, the entire security system is gone. Unfortunately, to encrypt, one needs to pass the private key, and during this delivery, the key can be stolen. Asymmetric encryption only needs to pass a public key for the encryption, which is ok to be stolen (making it more secure).

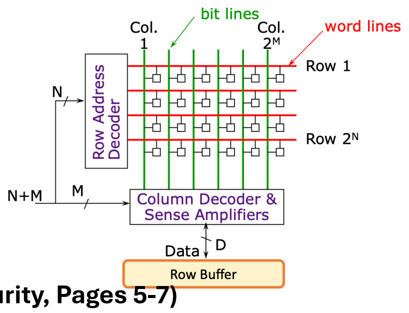
5. Hash (02 Security Concept, Pages 41)
Hashing is originally for fingerprinting a message. It is not for encryption.
The core security issue of hashing is the hash collision: hashing two different values can lead to the same hash. For instance, if someone is hashing two passwords, "abx" and "aby", a hash collision happens if hash(abx) == hash(aby). If a system uses and stores hash values instead of passwords (which most modern systems do), it means that an attacker can log in to an account with "aby" even if the user sets the password to "abx".

7. Cold boot attack (04 Software and Hardware Security, Pages 22-25)
A cold boot attack is possible because when the DRAM changes value 1 to 0 by simply not recharging it. If it wants the value 1, it frequently recharges. If it wants the value 0, it simply "not recharge" as it will quickly discharge. Under normal temperature, it discharges quickly. However, in low temperatures, it discharges very slowly, so that even after a long time, residue information can be recovered.

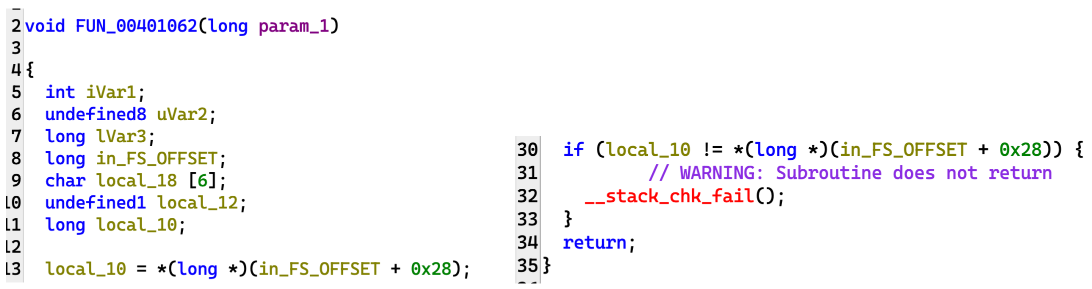


8. Row hammer (04 Software and Hardware Security, Pages 29-32)
A row hammer attack is possible because frequent row accesses can cause adjacent cells to become hotter, potentially flipping bits by charging the cell containing the bits (as the cells can also be charged by heat). Hence, to make it work, you must access the rows to physically fetch the data.
However, if accessed naively multiple times, the cache and row buffer will hold the previously accessed data and return it without physically accessing the rows. Hence, to make the row hammer work, one needs to

clear the cache (via the clflush instruction, which flushes the cache) and the row buffer (by accessing other rows alternatively).



9. "Non-executable Stack" or "DEP" (04 Software and Hardware Security, Pages 5-7)
There are two major stack buffer overflow protections. First is a stack guard, which stores a value (a canary value) on the stack between the local variable and the return address. If a local variable is overflowed and overwrites the return value, the stored canary value will also be changed. By checking the value of the canary, one can detect the buffer overflow. The following is an example. local_10 stores the canary value.



```
void FUN_00401062(long param_1)

{
  int iVar1;
  undefined8 uVar2;
  long lVar3;
  long in_FS_OFFSET;
  char local_18 [6];
  undefined1 local_12;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
```

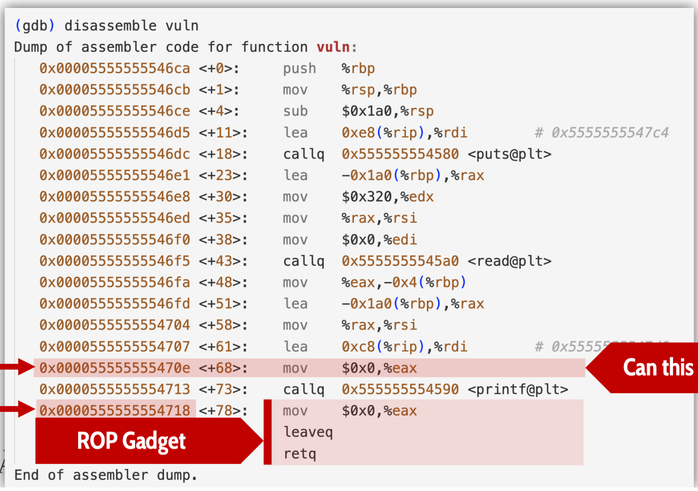
```
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
          // WARNING: Subroutine does not return
    __stack_chk_fail();
  }
  return;
}
```

Second is "Non-executable Stack" or "DEP". This simply makes any code on the stack not executable. It means attackers can inject code on the stack, but when they attempt to execute the injected code, execution is prevented.

10. ROP (04 Software and Hardware Security, Pages 9-13).
If an attacker can corrupt the stack, ROP is an idea that one can put a bunch of ROP gadget addresses that points to a few instructions followed by a return, such as **"mov eax, 0; ret"**. Once it jumps to a gadget, the last return instruction will pop the next ROP gadget, continuing the ROP attack. This is called ROP gadget chaining.

A good ROP gadget should achieve an objective (e.g., write a value to a register) safely. The goal is to execute the instructions safely and get to the ret instruction to jump to the next gadget. In the slide below, having a printf is not a good idea because printf has a lot of instructions that may crash if the arguments and environments for the printf are not properly set. Besides, "mov eax, 1; ret" and "mov eax, 0; add eax, 1; ret" are essentially the same, as they both are safe and achieve the same result.

Quiz 1 (Sep 30)



```
(gdb) disassemble vuln
Dump of assembler code for function vuln:
   0x00005555555546ca <+0>:     push   %rbp
   0x00005555555546cb <+1>:     mov    %rsp,%rbp
   0x00005555555546ce <+4>:     sub    $0x1a0,%rsp
   0x00005555555546d5 <+11>:    lea    0xe8(%rip),%rdi        # 0x5555555547c4
   0x00005555555546dc <+18>:    callq  0x555555554580 <puts@plt>
   0x00005555555546e1 <+23>:    lea    -0x1a0(%rbp),%rax
   0x00005555555546e8 <+30>:    mov    $0x320,%edx
   0x00005555555546ed <+35>:    mov    %rax,%rsi
   0x00005555555546f0 <+38>:    mov    $0x0,%edi
   0x00005555555546f5 <+43>:    callq  0x5555555545a0 <read@plt>
   0x00005555555546fa <+48>:    mov    %eax,-0x4(%rbp)
   0x00005555555546fd <+51>:    lea    -0x1a0(%rbp),%rax
   0x0000555555554704 <+58>:    mov    %rax,%rsi
   0x0000555555554707 <+61>:    lea    0xc8(%rip),%rdi        # 0x55555
   0x000055555555470e <+68>:    mov    $0x0,%eax
   0x0000555555554713 <+73>:    callq  0x555555554590 <printf@plt>
   0x0000555555554718 <+78>:    mov    $0x0,%eax
                                leaveq
                                retq
End of assembler dump.
```

ROP Gadget

Can this address be a functioning ROP Gadget?

No, it would crash inside "printf()". Why?