Vulnerabilities everywhere?
Capture the Flag!

- Compete in international hacking competitions
- Gain practical security experience
- Competitive, fun, challenging tasks
- Open, inclusive environment

Website:
- http://ctf.epfl.ch
- For now: http://dslab.epfl.ch/ctf/
- Ping us at CTF_EPFL@groupes.epfl.ch
Problem: broken abstractions?

C/C++

```c
void log(int a) {
    printf("Log: ");
    printf("%d", a);
}
void (*fun)(int) = &log;
void init() {
    fun(15);
}
```

ASM

```asm
log:
    ...

fun:
    .quad log

init:
    ...
    movl $15, %edi
    movq fun(%rip), %rax
    call *%rax
```
Software is highly complex

Low-level languages (C/C++) trade type safety and memory safety for performance

- Google Chrome: 76 MLoC
- Gnome: 9 MLoC
- Xorg: 1 MLoC
- glibc: 2 MLoC
- Linux kernel: 17 MLoC
Software is highly complex

~100 mLoC, 27 lines/page, 0.1mm/page ≈ 370m
CS-725: Language-based SoftSec

- Memory corruption and type safety flaws dominate the threat landscape.
- We approach research from three dimensions:
  - Sanitization (finding flaws through runtime monitors)
  - Fuzzing (testing software automatically)
  - Mitigation (protecting software at runtime)
- Websites:
  - https://nebelwelt.net/teaching/18-725-LSS/
Defense: Testing and Mitigations

Software Testing
- Discover bugs
- Development tool
- Result oriented

Mitigations
- Stop exploitation
- Always on
- Low overhead
Software Testing: Sanitization

- Detect policy violations for concrete executions
  - Spatial and temporal memory safety [AsiaCCS’18]
  - Type safety for casts [CCS’16, CCS’17]
  - Variadic argument safety [SEC’17]
  - Data flow integrity and argument integrity
Software Testing: Fuzzing

- Trigger flaws through thousands of executions
  - T-Fuzz: transform input and program [Oakland’18]
  - API Fuzz: infer API flow, create library fuzzers
  - ToggleFuzz: reduce sanitizer cost during fuzzing
Mitigation: Control-Flow Hijacking

- C++ type integrity and sensitive data integrity
- Compartment separation
HexHive: why is security awesome?

- Crosses all abstraction layers: from ISA to Web
- Concerns all languages: microcode, C, to JS
- Security mindset: understand flaw, mitigate
HexHive project pitch

- Fuzzing: increase code coverage (T-Fuzzing)
- Fuzzing: sanitizer-based coverage
- Fuzzing: data-flow aware mutation
- Sanitization: API flow graph
- Mitigation: Control-flow chaining
- Open security benchmark: systematization
- Side channel: JS-based ISA inference
  - Or: http://bit.do/hexhive18
Fuzzing
Sanitization
Mitigation

Mathias Payer
BC 106