

CS412 Software Security

Summary



Mathias Payer

EPFL, Spring 2019

- Software lifecycle
- Security policies
- Software bugs
- Attack vectors
- Mitigations
- Program testing
- Case studies: web, mobile

- Software lives and evolves
- Security must be first class citizen
 - Secure requirements/specifications
 - Security-aware design (what are threats?)
 - Secure implementation (code reviews)
 - Testing (red team, fuzzing, unit)
 - Updates and patching

- Memory safety: spatial and temporal memory safety
 - SoftBound: spatial safety through disjoint metadata for pointers
 - CETS: temporal memory safety through versioning of pointer/object
- Type-safe code restricted to access authorized locations
 - Keep per-object disjoint metadata
 - Check all type casts

- Memory safety bugs allow program state modification
 - Spatial: bounds violation
 - Temporal: validity
- Type confusion reuses memory as different type
- Integer overflow results in miscomputation (pointer arithmetic)

- Exploitation is an art
- Constrained resources (buffer size, limited control, limited information), must control the application state
- Execute outside of defined program semantics
- Control-flow hijacking: code injection versus code reuse
- Data-only attacks

- Data Execution Prevention: stops code injection (not code reuse)
- Address Space Layout Randomization: probabilistic, prone to memory leaks
- Stack canaries: probabilistic, prone to direct overwrites
- Safe Exception Handling: exception handler reuse
- Fortify source: format string protection
- Stack integrity: stack canaries, shadow stacks, safe stacks
- CFI: protect forward edge control-flow hijacking
- CPI: enforce memory safety for code pointers and sensitive pointers
- Sandboxing to enforce privilege domains

- Find bugs before attacker
- Manual testing: write unit tests
- Sanitizers allow early bug detection (memory/type safety focus)
- Fuzz testing automates and randomizes testing (e.g., AFL, libFuzzer)
- Symbolic/concolic testing allows full coverage analysis

- Daemons are long running, complex, and exposed
- Command/SQL injection is code injection
- XSS allows execution of malicious JavaScript
- Android security evolved over time, hardened base system
- Applications are vetted centrally, installed from market

Summary summary

- Software security is complex
- Think across all layers of the stack
- Planning is important
- Exam: 2hrs, 120 points, 7 questions
 - 1/3 2/3 split before/after midterm
 - Reverse engineering question!