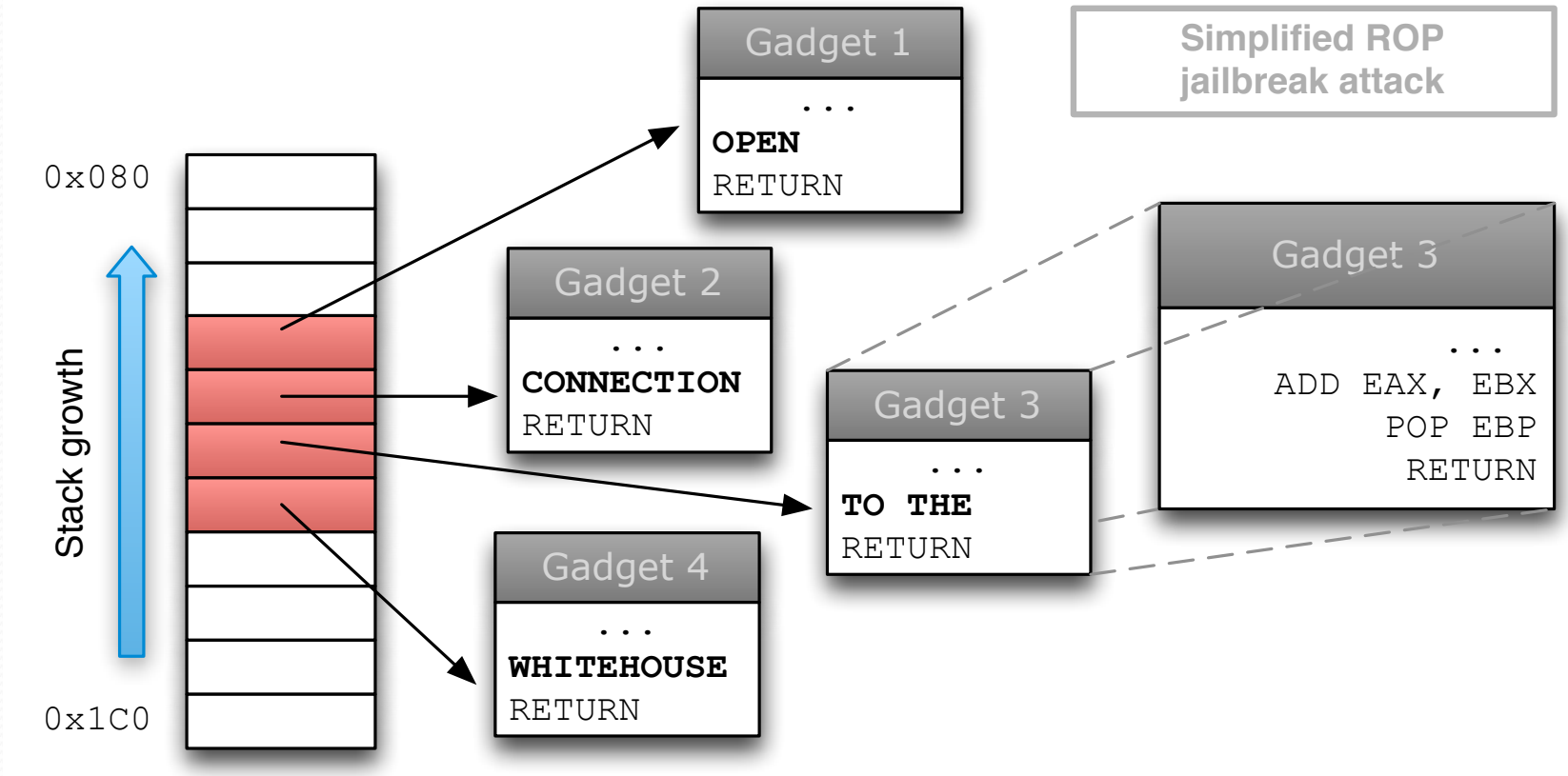
Microgadgets: Size Does Matter in Turing-complete ROP

Andrei Homescu
Michael Stewart
Stefan Brunthaler
Per Larsen
Michael Franz
University of California Irvine

Return-oriented Programming



Return-oriented Programming

- Original version: hand-picked set of gadgets from *libc*; downside: **specific to library (Debian 5 libc)**
- Later work uses automated scanners:
  - Scan for 1-instruction gadgets
  - Match using expression trees
  - Use postconditions
  - Formulate gadget outputs as boolean functions
    - SMT solver

Complicated Gadgets

- More than one instruction per gadget
- Complex instruction operands (memory offsets, immediates)
- Interference between instructions

Gadget Examples

`ADD EAX, [EBX+0x35CFE022]`	`03 83 22 E0 CF 35`
`LEA EAX, [EBX+2*EAX+0x20]`	`8D 44 43 20`
`INC [0x98560F0E]`	`FF 05 0E 0F 56 98`
`ADD EAX, EBX` `INC EAX`	`01 D8` `40`
`ADD EAX, 0x20` `POP EBP`	`83 C0 20` `5D`

Goals for Our Gadget Set

- **Ubiquity**
  - In as many binaries as possible
- **Computational Power**
  - Turing-complete
- **Simplicity**

Key Insight for Ubiquity

Smaller gadgets

occur

more frequently

Microgadget Set

- Focus on very small gadgets (2-3 bytes)
- Grouped into classes, by operation
- 1 instruction + `RET` in each gadget

- **Expectation:** find this set very frequently in large binaries (**ubiquity**)

Smallest Gadget

RET

C3

2 Byte Microgadgets

- `XCHG EAX,reg` for register-to-register moves
- `POP reg` for loading constants
- `LAHF / PUSHF` for loading flags
- `INC / DEC reg`
- `CLC /SAHF / DAA / ...` to clear carry flag
- `PUSHA` to copy `ESP` into another register
- `LEAVE` to change `ESP`
- `LODSD / STOSD` for memory accesses
- `PUSH reg` for control flow (function calls)

Turing-complete?

Turing-complete? NO!

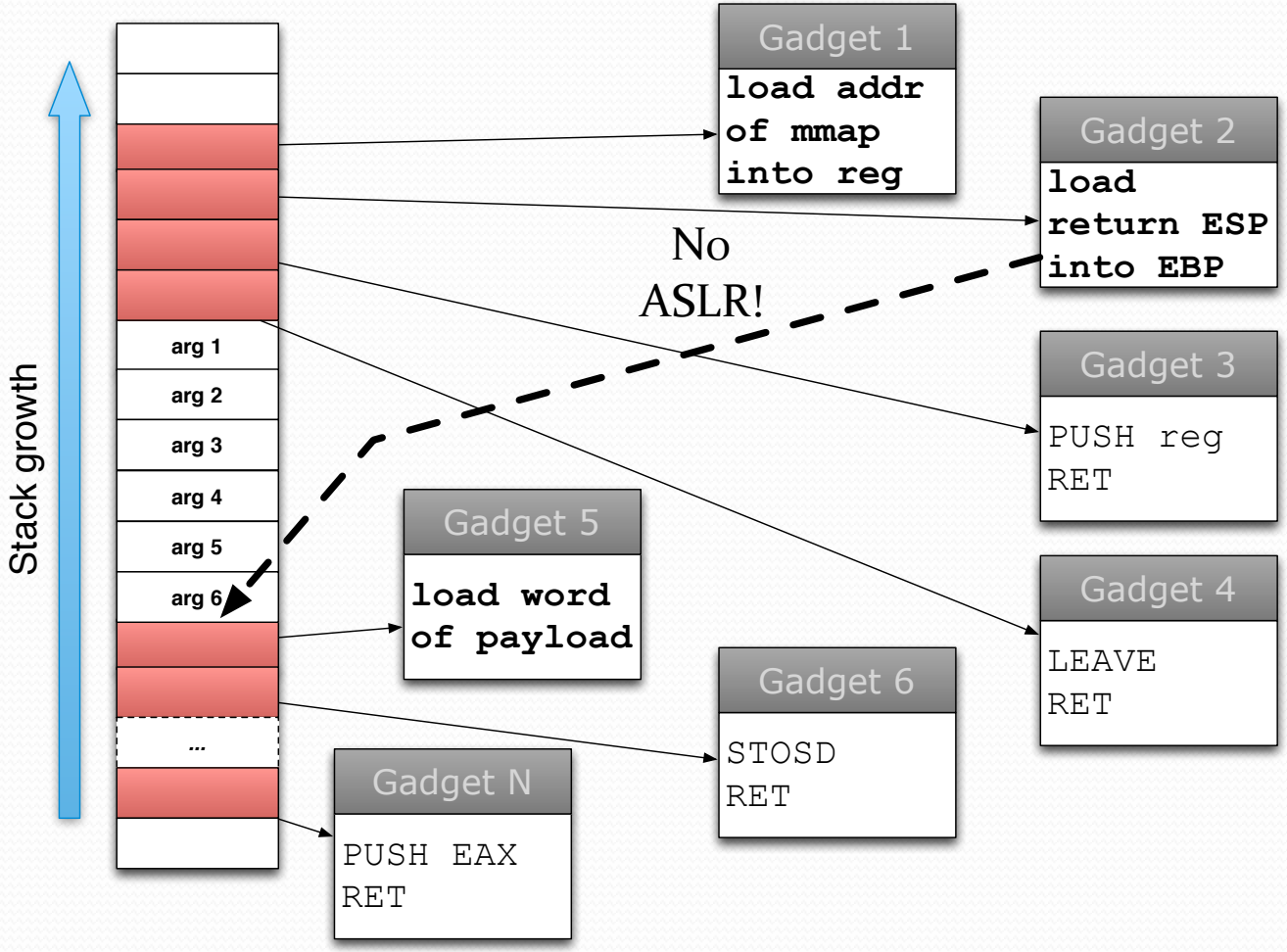
2 Byte Microgadgets

- `XCHG EAX,reg` for register-to-register moves
- `POP reg` for loading constants
- `LAHF / PUSHF` for loading flags
- `INC / DEC reg`
- `CLC /SAHF / DAA / ...` to clear carry flag
- `PUSHA` to copy `ESP` into another register
- `LEAVE` to change `ESP`
- `LODSD / STOSD` for memory accesses
- `PUSH reg` for control flow (function calls)

Memory store? YES!

Function calls? YES!

Attack example

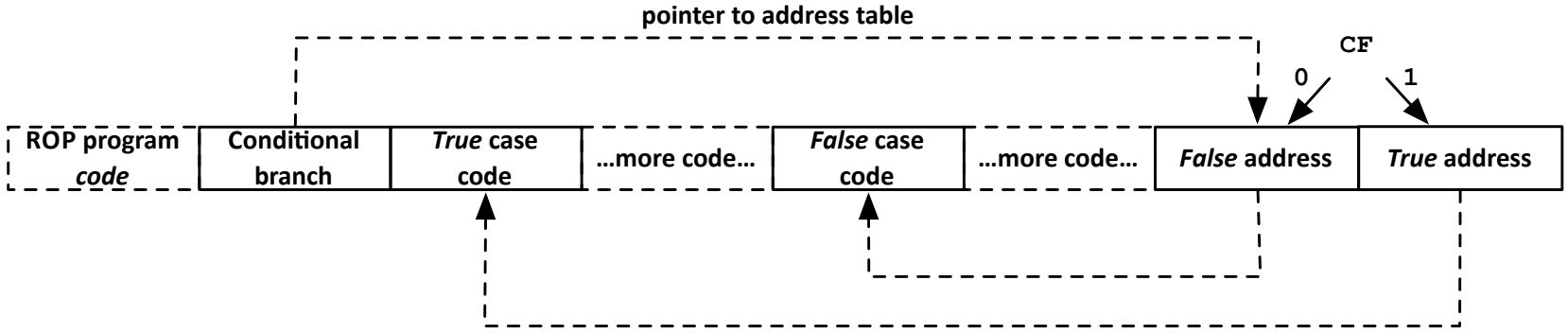


3 Byte Microgadgets

- `ADD / ADC / SUB / SBB`
- `XOR / NOT / NEG`
- `AND / OR`
- One memory load, if 2-byte version not available
- One memory store, if 2-byte version not available
- One operation to copy `ESP` at the beginning
  - For ASLR-proof `mmap` call: just add given value to `ESP`

Conditional Branches

- Needed for Turing-completeness; **tricky**
- Supported operation: "branch if A < B"
- Addition/subtraction => `CF` => branch
- Branch target is loaded from a table



Turing-complete?

Turing-complete? YES!

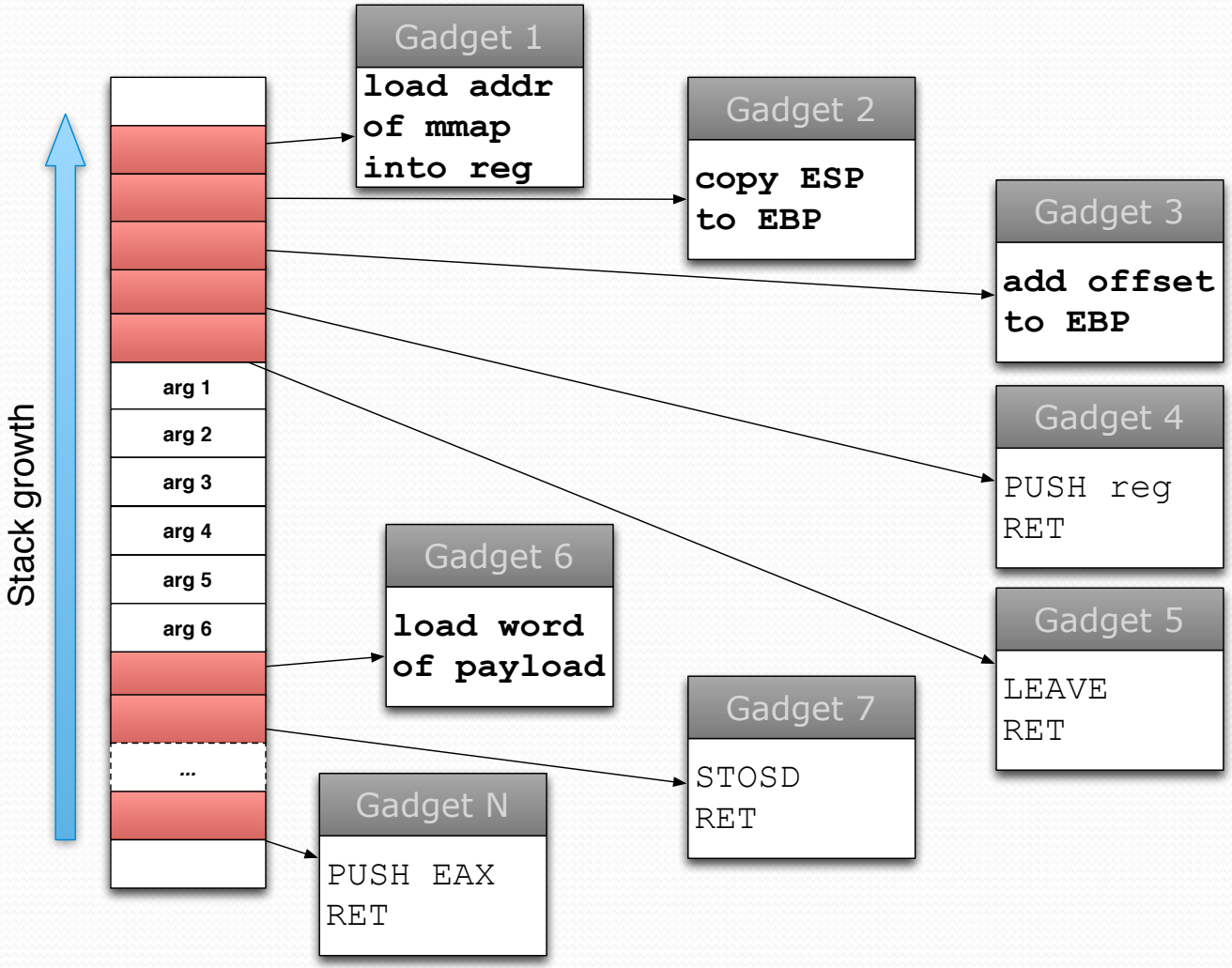
Turing-completeness

- Model implements `subneg` variant of OISC (one-instruction set computer)
- Only needed 3 operations:
  - Subtraction
  - Less-than comparison
  - Conditional branching
- `subneg` and `subleq` are known to be Turing-complete
- **Bonus operations:**
  - Addition
  - Boolean operations
  - Extra memory load/stores

Evaluation

Distribution	Total binaries						
	All	no XCHG c-mode	XCHG c-mode	no XCHG i-mode	XCHG i-mode	mmap no ASLR	mmap ASLR
Kubuntu 7.10	1337	404	262	27	8	17	9
Ubuntu 9.04	1492	434	212	31	4	21	6
Ubuntu 10.04	1587	497	164	35	10	24	12
Kubuntu 11.10	1655	565	271	45	15	39	15

Attack Example



Conclusion - Goals

- **Ubiquity**
  - In as many binaries as possible
- **Simplicity**
- **Computational Power**
  - Turing-complete

Future Work

- Make classes larger
- Port to 64 bits: **not trivial!**
- Extend model to other return/jump instructions
- Measure probability of binary from given distribution to contain microgadgets
  - Build distribution of n-grams in binaries

Questions?
Thank you!