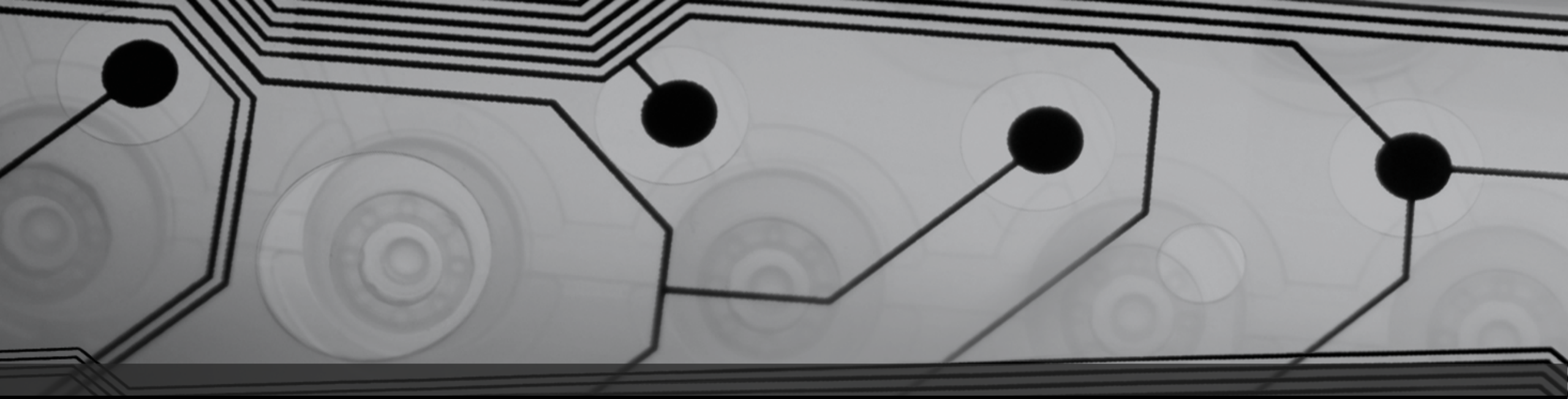


Attacking the Linux PRNG on Android

David Kaplan, Sagi Kedmi, Roee Hay & Avi Dayan
IBM Security Systems

MOTIVATION

motivation_keystore_buffer_overflow

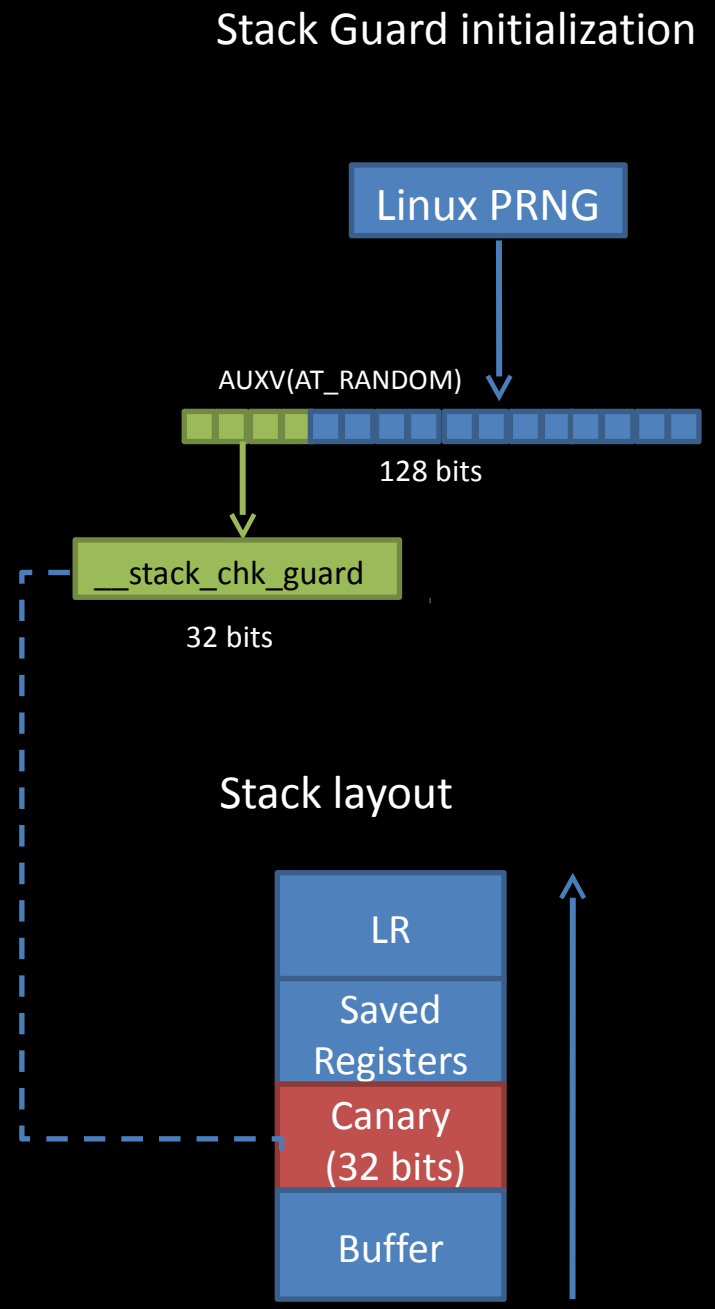
- We discovered CVE-2014-3100, a **stack-based Buffer Overflow** in keystore
  - Service responsible of securely storing crypto related data
- We had privately reported to Google and they provided a patch available in *KITKAT*.
- Exploit must overcome various defense mechanisms, including **Stack Canaries.**

```
/* KeyStore is a secured storage for key-value pairs. In this implementation,
 * each file stores one key-value pair. Keys are encoded in file names, and
 * values are encrypted with checksums. The encryption key is protected by a
 * user-defined password. To keep things simple, buffers are always larger than
 * the maximum space we needed, so boundary checks on buffers are omitted. */
```

motivation_keystore_buffer_overflow

Attacks on the Stack-Smashing Protection:

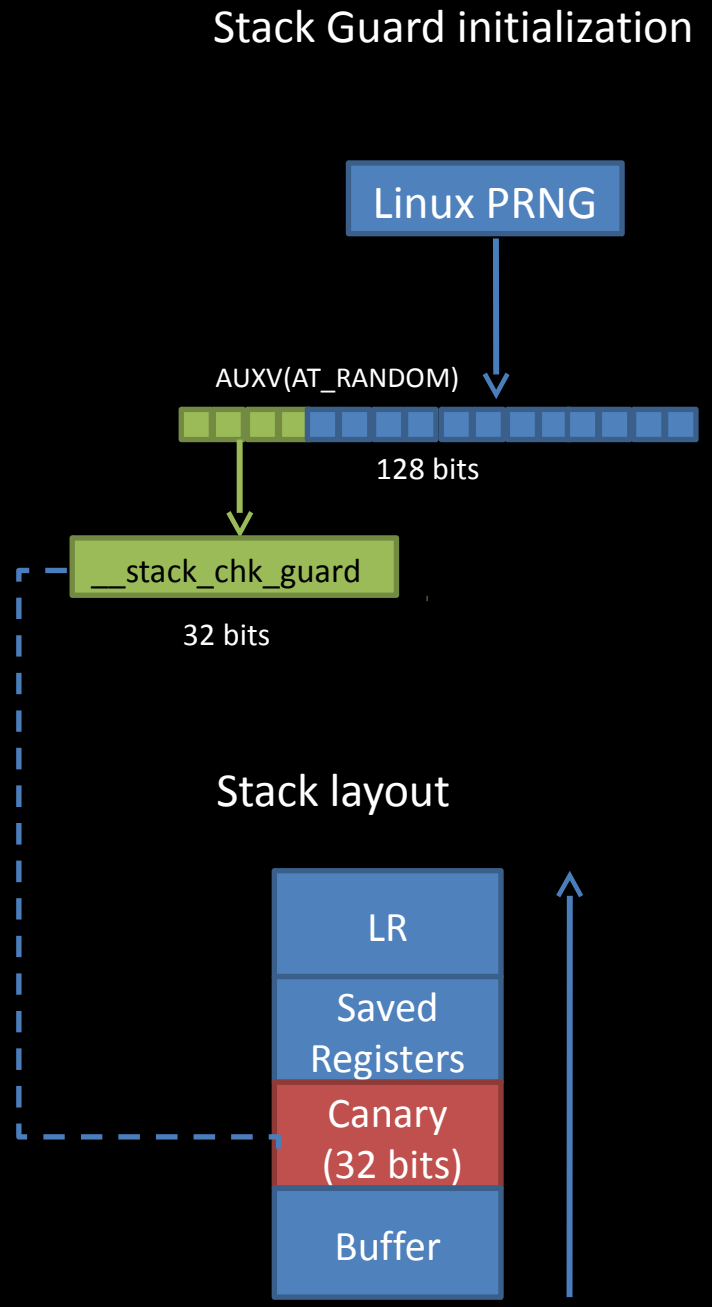
- Naive Online Bruteforce of the *Canary* Value
  - Impractical: 2^32 attempts on average.



motivation_keystore_buffer_overflow

Attacks on the Stack-Smashing Protection:

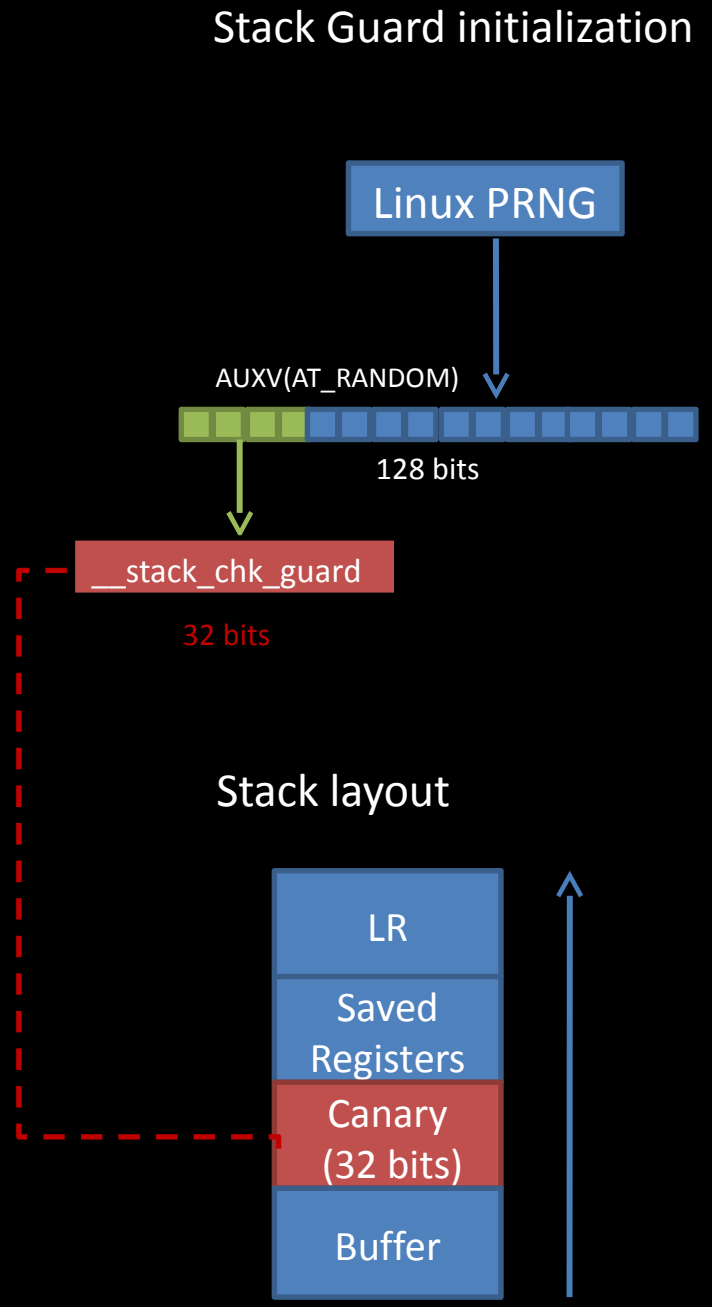
- Naive Online Bruteforce of the *Canary* Value
  - Impractical: 2^32 attempts on average.
- Online Learning of the *Canary* Value
  - By another info leak issue
  - Re-forking server:
    - Very efficient: 514 attempts until success on average



motivation_keystore_buffer_overflow

Attacks on the Stack-Smashing Protection:

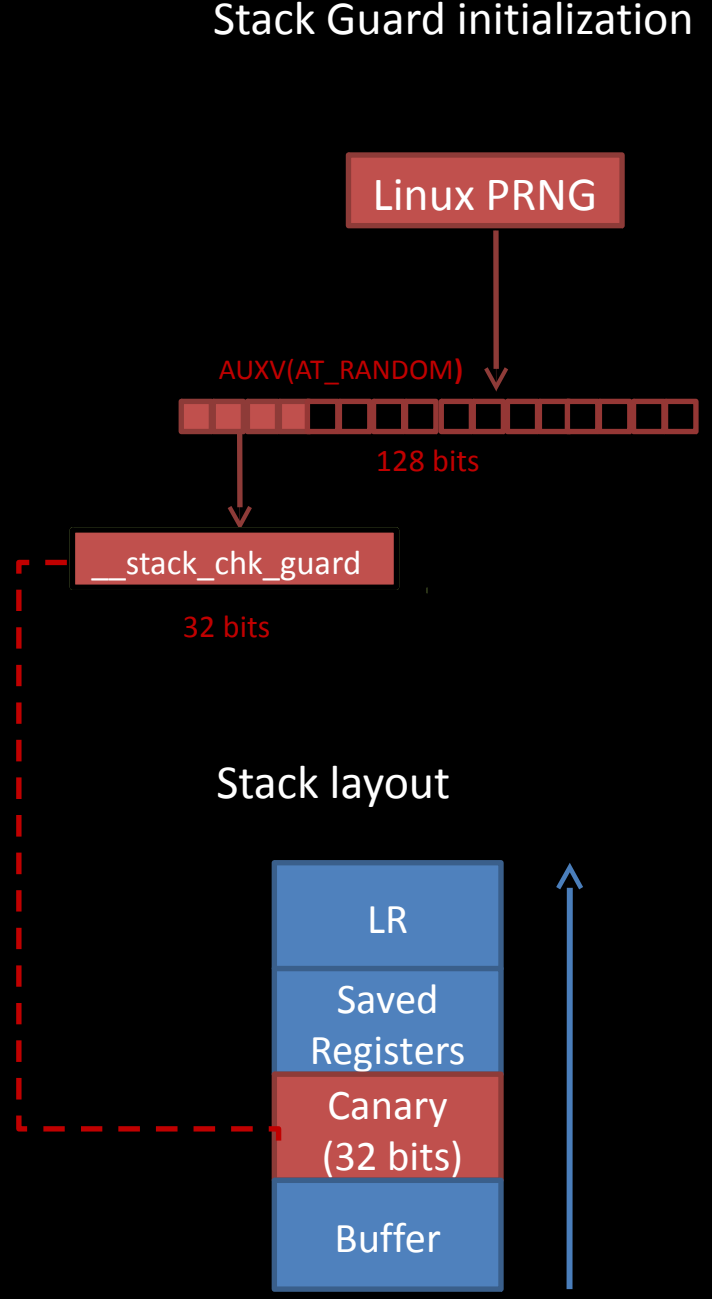
- Naive Online Bruteforce of the *Canary* Value
  - Impractical: 2^32 attempts on average.
- Online Learning of the *Canary* Value
  - By another info leak issue
  - Re-forking server:
    - Very efficient: 514 attempts until success on average
- Overwrite *__stack_chk_guard*
  - By overwriting some pointer



motivation_keystore_buffer_overflow

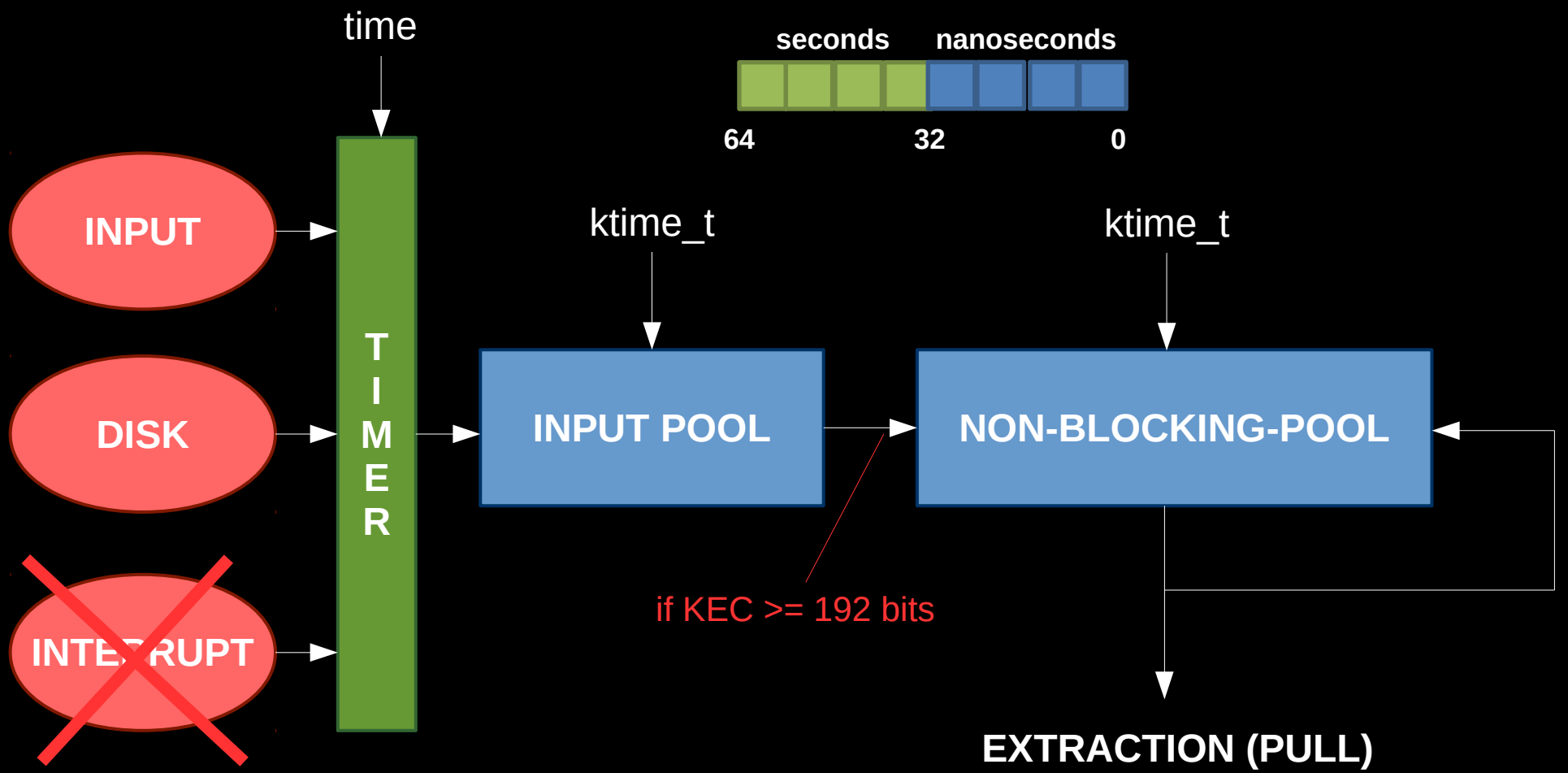
Attacks on the Stack-Smashing Protection:

- Naive Online Bruteforce of the *Canary* Value
  - Impractical: 2^32 attempts on average.
- Online Learning of the *Canary* Value
  - By another info leak issue
  - Re-forking server:
    - Very efficient: 514 attempts until success on average
- Overwrite *__stack_chk_guard*
  - By overwriting some pointer
- **Our attack**: *Offline* reconstruction of the PRNG's internal state



LINUX PRNG

entropy_sources



*KEC = Kernel Entropy Count

OUR WORK

contribution

Prior art on weakness in early boot *

Present practical run-time attack

Formalize attack

Demonstrate PoC against current mobile platforms

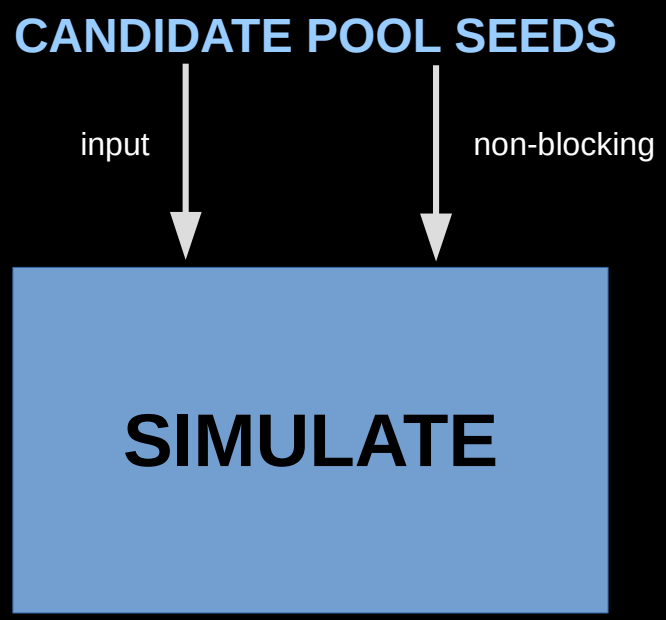
** Heninger et al. 2012, Becherer et al. 2009, Ding et al. 2014*

attack

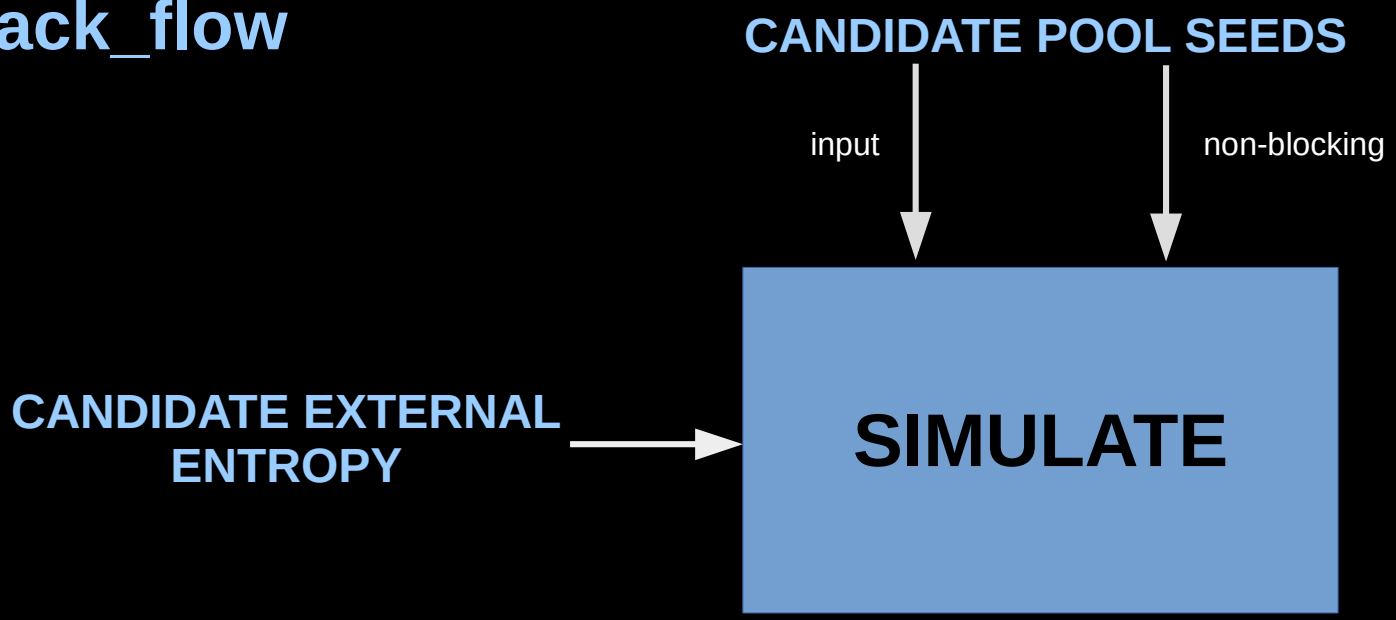
Given a **LEAK** of a value extracted from the non-blocking pool and **LOW ENTROPY AT BOOT**, the **STATE** of the PRNG can be determined prior to time of leak until external entropy is indeterminate

attack_flow

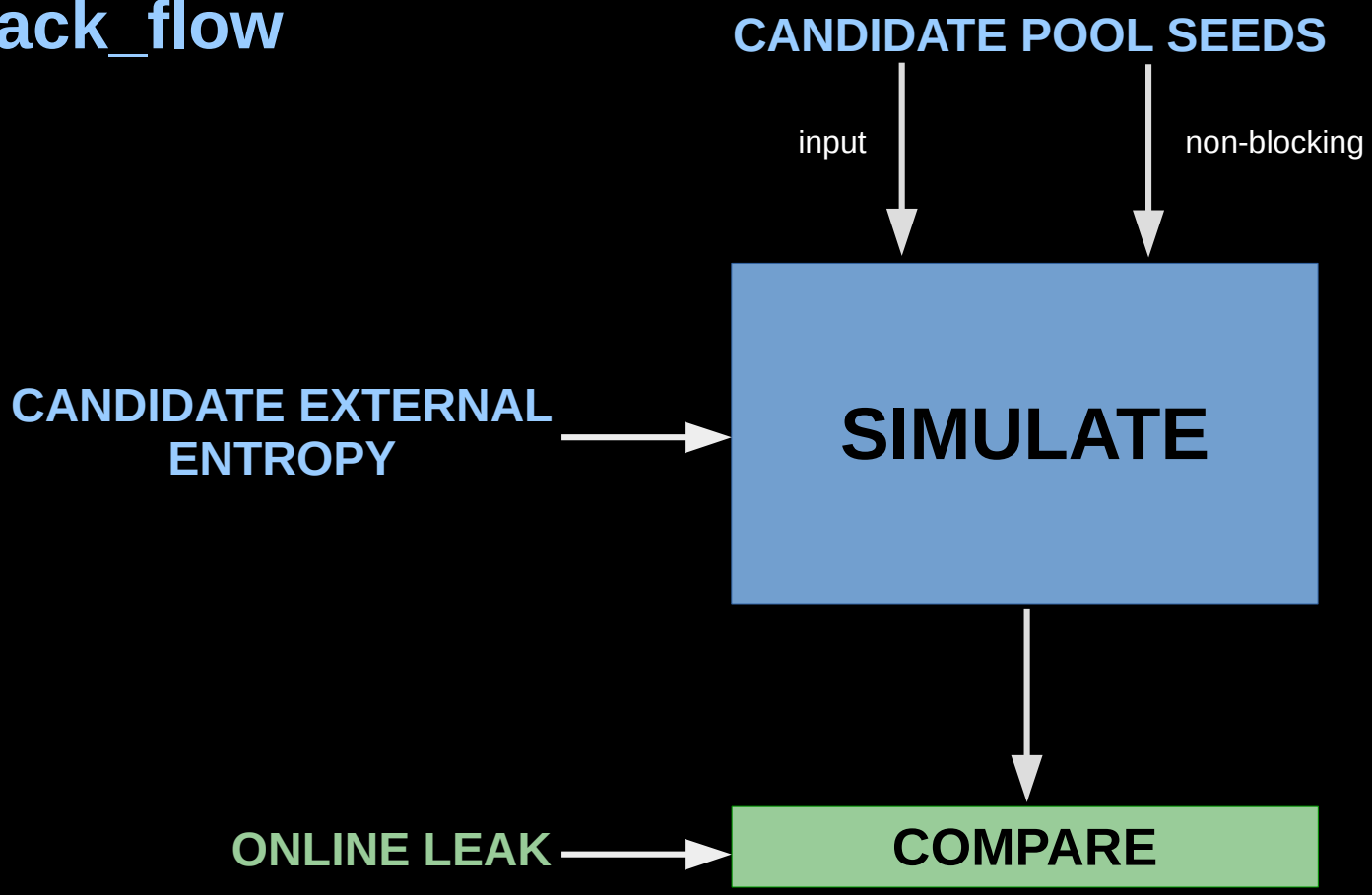
attack_flow



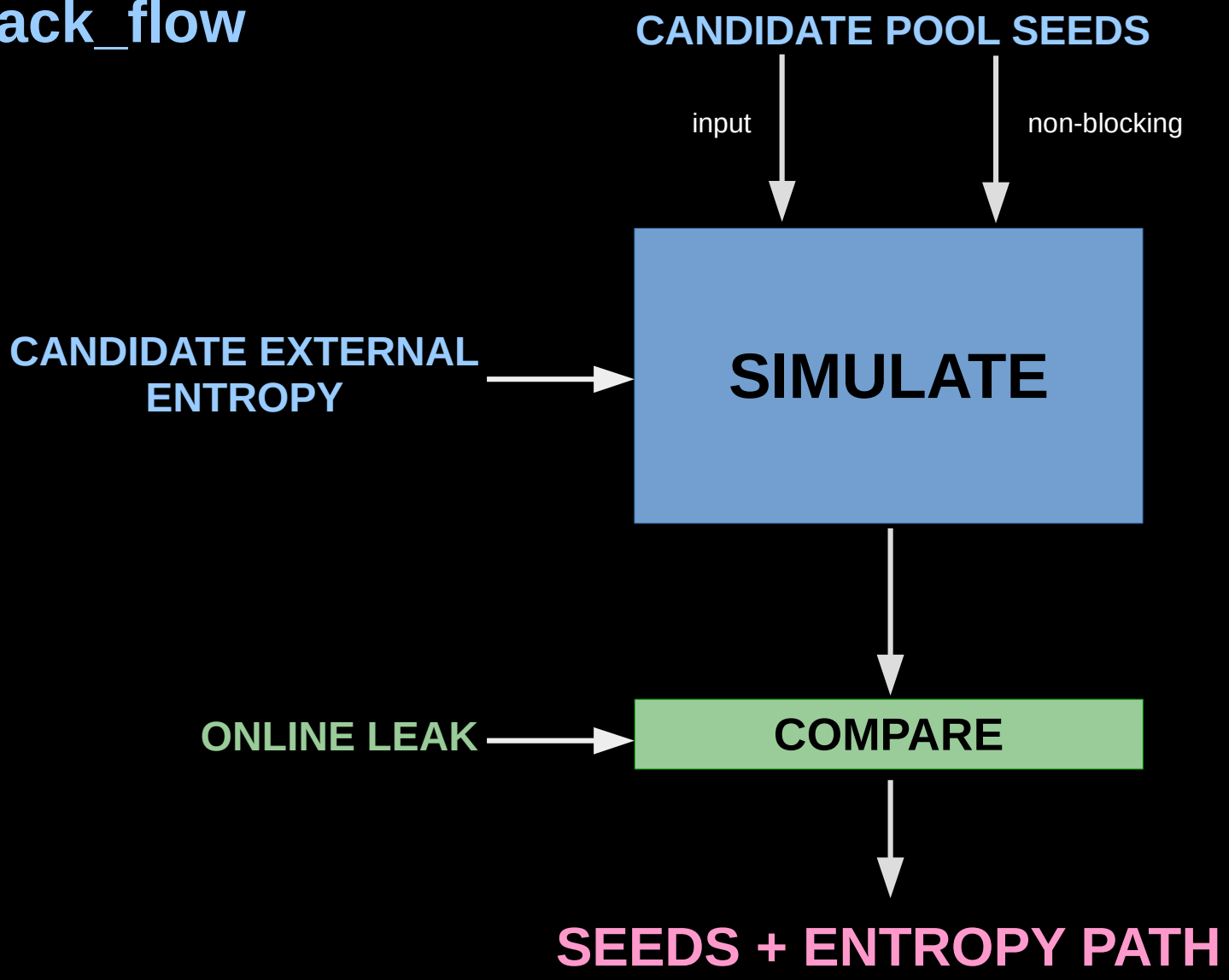
attack_flow


CANDIDATE POOL SEEDS
input
non-blocking
CANDIDATE EXTERNAL ENTROPY
SIMULATE

attack_flow

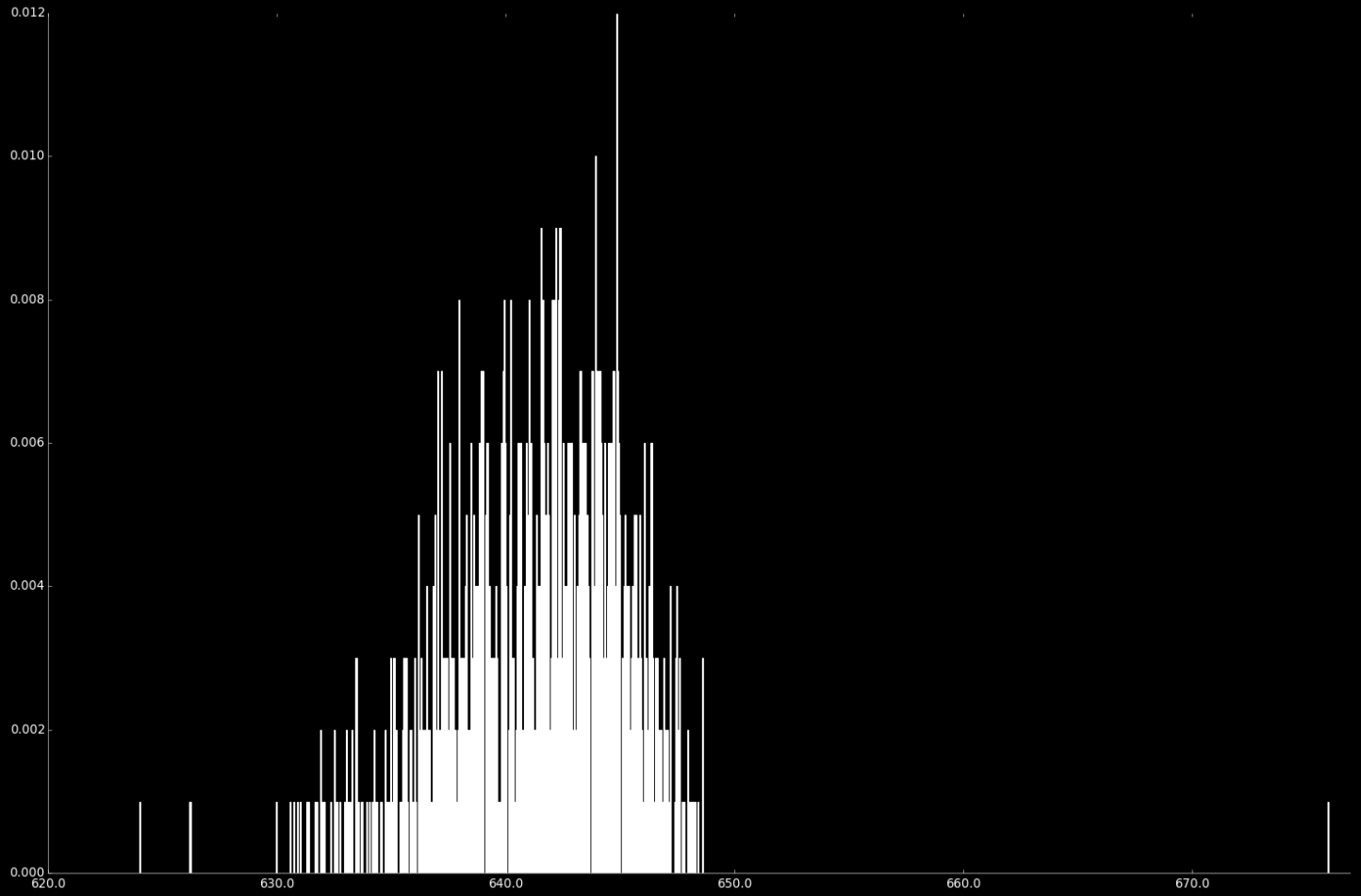

CANDIDATE POOL SEEDS
input
non-blocking
CANDIDATE EXTERNAL ENTROPY
SIMULATE
ONLINE LEAK
COMPARE

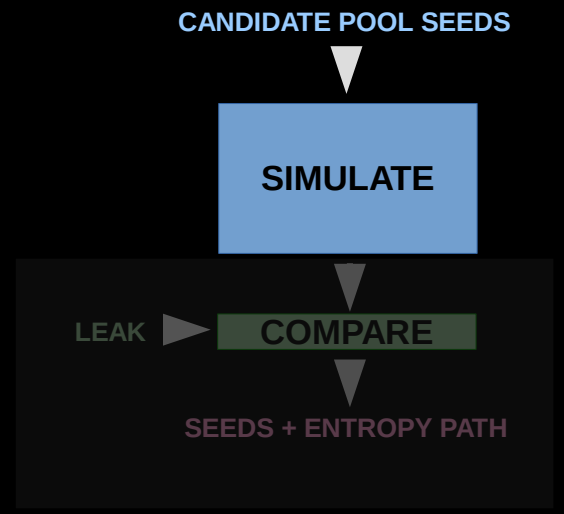
attack_flow


CANDIDATE POOL SEEDS
input
non-blocking
CANDIDATE EXTERNAL ENTROPY
SIMULATE
ONLINE LEAK
COMPARE
SEEDS + ENTROPY PATH

EXPERIMENT

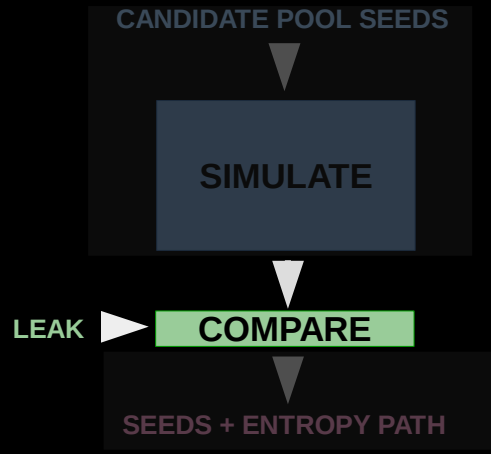
s4_non-blocking_seed





$$H(s_{nb}) = 23.5\,bits$$

prng_value_leak



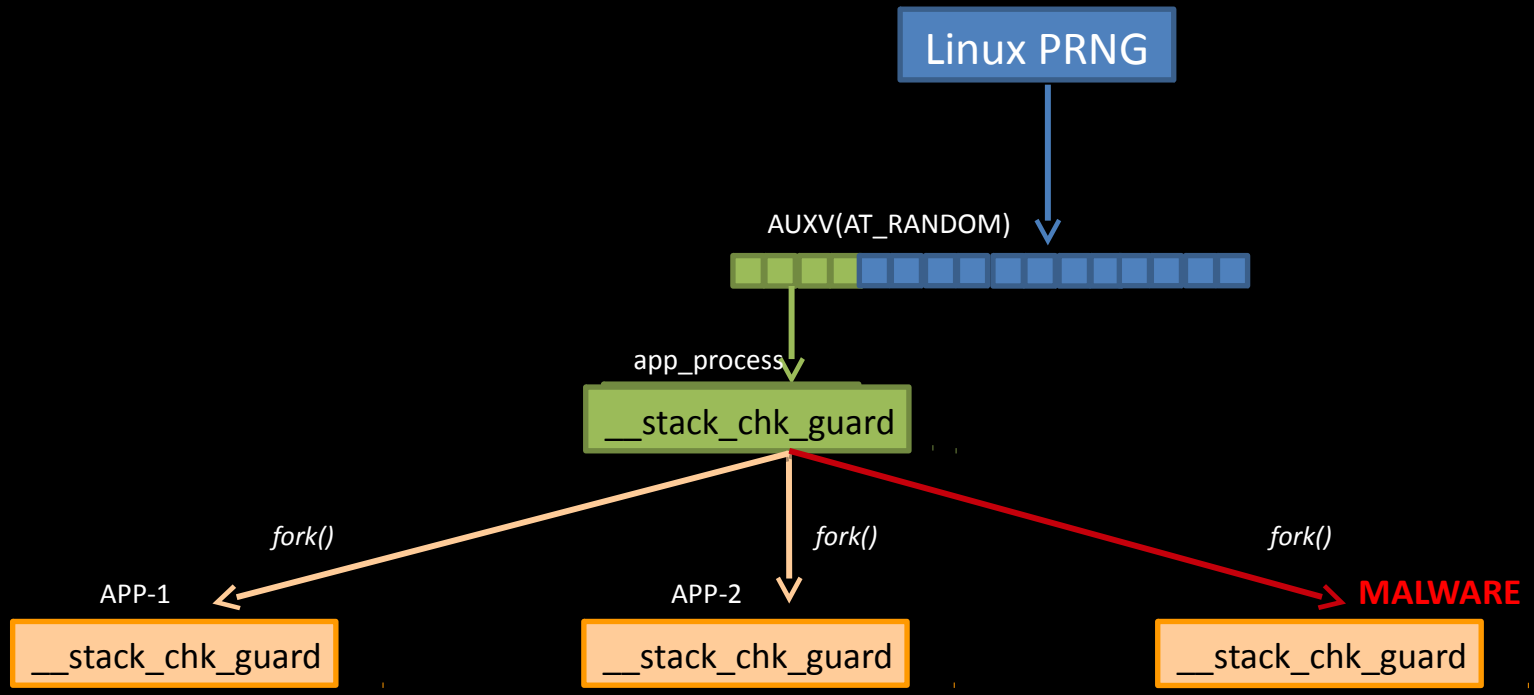
ANDROID ZYGOTE (app_process)

Apps fork()ed from Zygote process

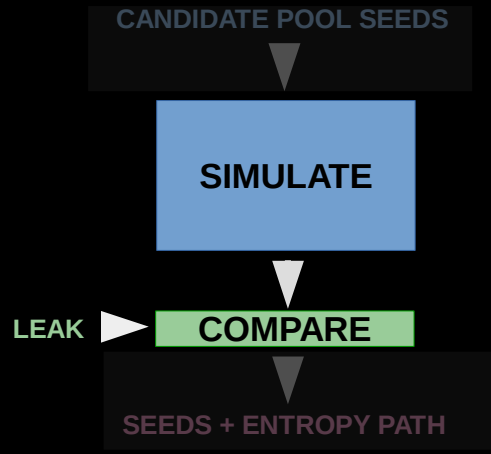
Same AUX VECTOR

AT_RANDOM – 16 bytes extracted from PRNG

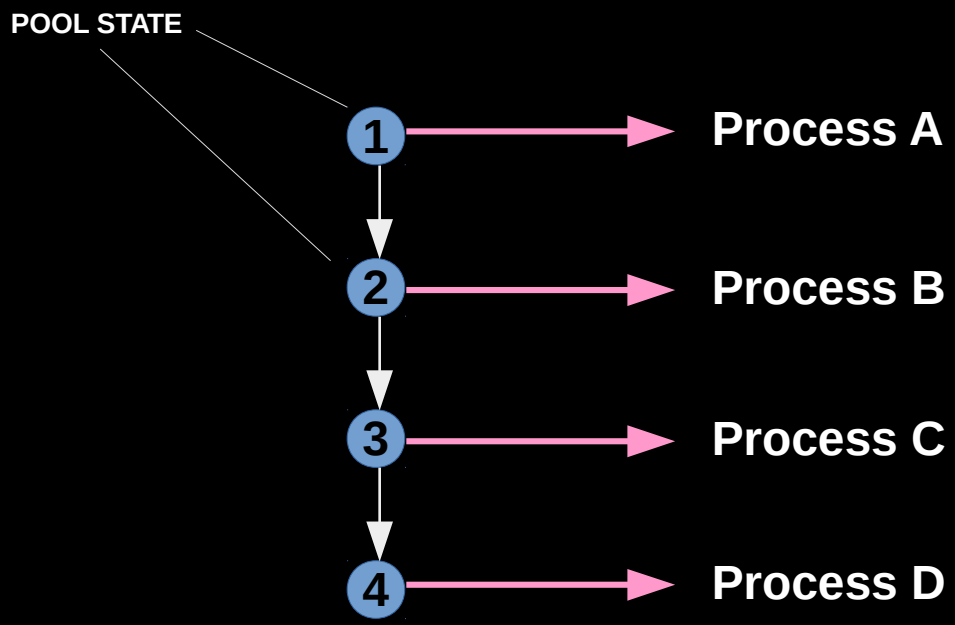
LEAK: Malicious app extracts AT_RANDOM from mem space



simulate_prng



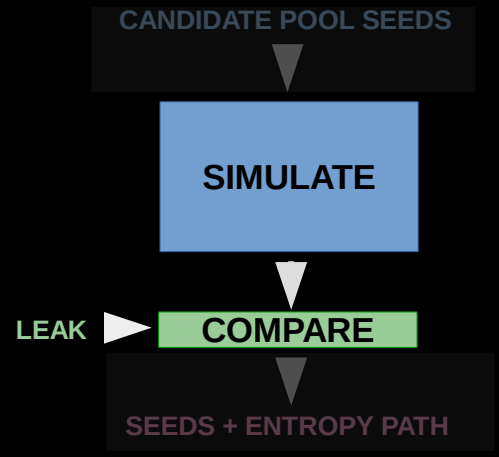
Issue 1: order of extraction non-deterministic due to concurrent execution



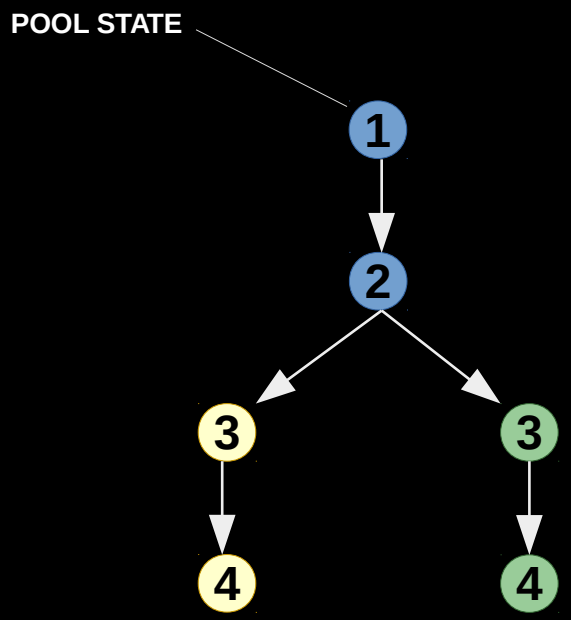
Which process is Zygote?

Solution: linear search in set of extracted blocks

simulate_prng



Issue 2: pool state corruption due to concurrent execution

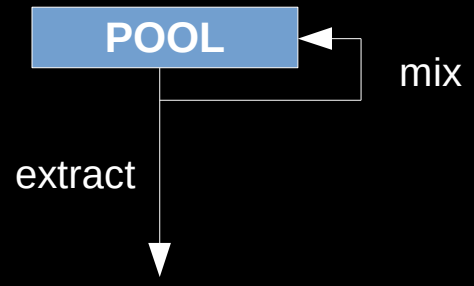


Yellow Path

- **Process A:** extract from pool
- **Process A:** mix into pool
- **Process B:** extract from pool
- **Process B:** mix into pool

Green Path

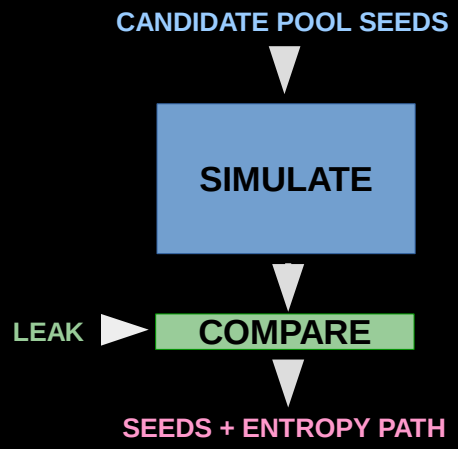
- **Process A:** extract from pool
- **Process B:** extract from pool
- **Process A:** mix into pool
- **Process B:** mix into pool



Most likely path in **20%** of cases

~uniform distribution otherwise

s4_demo


CANDIDATE POOL SEEDS
SIMULATE
LEAK
COMPARE
SEEDS + ENTROPY PATH

DEMO

wrap_up

Demonstrated practical attack against S4 – derandomization KeyStore canary

Another remote leak can be found in the paper

Affects majority of devices in the field

Mitigations in newer kernels + devices

Thank you