PTAuth: Temporal Memory Safety via Robust Points-to Authentication

Reza Mirzazade Farkhani  Mansour Ahmadi  Long Lu
Motivation

Temporal memory errors remain to be the most commonly exploited software vulnerabilities.

State-of-the-art approaches rely on a huge amount of metadata which leads to high memory overhead.

The metadata is left unprotected.
Motivation

Dynamically detect temporal memory corruptions in the heap

Research questions:

1. Can we design a system with low-memory overhead and reasonable run-time overhead?
2. Can we guarantee the integrity of metadata without relying on other approaches?
System Design

PTAuth checks upon each pointer dereference:

1. Whether the pointer is pointing to the original or intended object
2. Whether the metadata or evidence proving the points-to relationship is genuine
System Design

PTAuth checks upon each pointer dereference:

1. Whether the pointer is pointing to the original or intended object
2. Whether the metadata or evidence proving the points-to relationship is genuine (how?)

**Pointer Authentication Code (PAC)**

Diagram showing the flow of data from Pointer to PAC to PACIA with a 64-bit sign and unused bits.
**Pointers & Objects**

\[
\text{ID} = \text{RandomID()} \ // \ 64 \text{-bit}
\]

\[
\text{AC} = \text{PACIA} < \text{BasePointer} > < \text{ID} >
\]
System Overview

5: object *pointer = alloc(...);

\[ AC_a = PACIA \ (Address_a, ID_a) \]

\[ \text{AUTIA} \ (Address_a, ID_a) \]

5: \texttt{*pointer = 0x7fff5694...;}
Backward Search

What if the pointer does not point to the beginning of the buffer?

Beginning of the buffer

ID

object

AC  pointer

objects are aligned to 16 byte
Optimization

```c
void quantum_gate2(quantum_reg * reg) {
    int i, j, k, iset;
    int addsize = 0, decsize = 0;
    if (reg->num > reg->max)
        printf("maximum", reg->num);
    else {
        for (i = 0; i < (1 << reg->hashw); i++)
            reg -> hash[i] = 0;
        for (i = 0; i < reg->size; i++)
            quantum_add_hash(reg->node[i].state, i, reg);
        ...
    }
}
```

The `reg` the pointer is authenticated before passing to the `quantum_gate2` function, no check on it is needed until the pointer is passed to another function as an argument.
Software Implementation of PAC Instructions

```c
#if PACENABLED
asm(
   "mov %:
   : "r"(ptr));
asm(
   "pacia %
   : "=r"(ptr)
   : "r"(id));
#else
ptr = __pacia(ptr, id);
#endif

long MASKBITS = 0 b000...000111111111111111;
void * __pacia(void * ptr, long id) {
   long ptrbits = (unsigned long) ptr & MASKBITS;
   long idbits = id & MASKBITS;
   long signature = ptrbits ^ idbits;
   signature = signature << 48;
   unsigned long ptrWithSign = (unsigned long) ptr | signature;
   return (void *) ptrWithSign;
}
```

FVP simulator

Pi 4
Runtime Overhead

26% overhead
Memory Overhead

- 2% overhead

- 401.bzip2
- 429.mcf
- 433.milc
- 445.gobmk
- 458.soplex
- 462.libquantum
- 464.hmmer
- 470.lbm
- 482.sphinx3

- CRCount
- DangSan
- PTAuth
Conclusion

- We provide a resilient and efficient **points-to authentication** scheme for detecting temporal memory corruptions.
- PTAuth repurposed **PAC** on ARMv8.3-A to detect temporal memory corruption.
- PTAuth provides metadata integrity for objects and pointers.
Thanks! Questions?

Reza Mirzazade Farkhani

mirzazadefarkhani.r@northeastern.edu