CrypTag: Thwarting Physical and Logical Memory Vulnerabilities using Cryptographically Colored Memory

Pascal Nasahl, Robert Schilling, Mario Werner, Jan Hoogerbrugge, Marcel Medwed, Stefan Mangard

AsiaCCS'21, June 7-11, 2021

IAIK – Graz University of Technology

Motivation

Memory Safety Vulnerabilities

- MITRE: 3 out of 10 are memory vulnerabilities [MIT19]
- Microsoft: 70% security-related bug fixes [Mil19]

Memory Safety Vulnerabilities

- MITRE: 3 out of 10 are memory vulnerabilities [MIT19]
- Microsoft: 70% security-related bug fixes [Mil19]
- Entry point for various attacks

Memory Safety Vulnerabilities

- Logical Memory Safety Vulnerabilities
- Physical Memory Safety Vulnerabilities

Phyiscal Memory Safety Vulnerabilities

- Exposed external memory
- Cold-boot [Hal+08], Bus sniffing [Nur20]
- Software-based attacks
- Cloud and IoT

Thwarting Phyiscal Memory Safety Vulnerabilities

- Confidentiality & Integrity
- Memory Encryption
- Average runtime overheads between 5 % and 109.8 %
- Broadly available in Intel and AMD processors

Logical Memory Safety Vulnerabilities

- Memory vulnerabilities exploit a memory bug
- Classified in spatial and temporal memory bugs
- Temporal error: dereferencing a dangling pointer
- Spatial error: out-of-bounds access

Logical Memory Safety Vulnerabilities

- Use the pointer: [Sze+13]
  - Modify a data pointer
  - Modify code and data
  - Modify a code pointer
  - Output data

Thwarting Logical Memory Safety Vulnerabilities

- Data-flow integrity
    - DFI, HDFI: Enforcing data-flow graph

Thwarting Logical Memory Safety Vulnerabilities

- Data-flow integrity
  - DFI, HDFI: Enforcing data-flow graph
- Code-pointer integrity
  - CPI: Store in safe region

Thwarting Logical Memory Safety Vulnerabilities

- Data-flow integrity
  - DFI, HDFI: Enforcing data-flow graph
- Code-pointer integrity
  - CPI: Store in safe region
- Code- and data-pointer integrity
  - PARTS: Integrity of all code- and data-pointers

Memory Safety

- **Prevents** all spatial and temporal memory errors

Memory Safety

- **Prevents** all spatial and temporal memory errors
- Spatial memory safety
  - Softbound, Hardbound: Bounds for each object

Memory Safety

- **Prevents** all spatial and temporal memory errors
- Spatial memory safety
  - Softbound, Hardbound: Bounds for each object
- Temporal memory safety
  - Watchdog: Metadata stored in shadow memory

Memory Safety

- **Prevents** all spatial and temporal memory errors
- Spatial memory safety
  - Softbound, Hardbound: Bounds for each object
- Temporal memory safety
  - Watchdog: Metadata stored in shadow memory
- **Large** performance overhead

Memory Safety

- **Prevents** all spatial and temporal memory errors
- Spatial memory safety
  - Softbound, Hardbound: Bounds for each object
- Temporal memory safety
  - Watchdog: Metadata stored in shadow memory
- **Large** performance overhead
- **Hardware support is needed!**

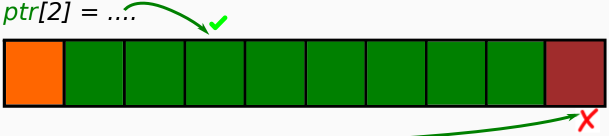
Memory Coloring

- `Lock-and-key` approach

Memory Coloring

- `Lock-and-key` approach

*char *ptr = new char[8];*
ptr[2] = ...

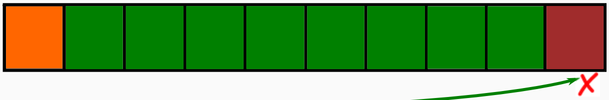


ptr[8] = ...

Memory Coloring

- `Lock-and-key` approach

*char *ptr = new char[8];*
ptr[2] = ...



ptr[8] = ...

- Memory Allocation: lock object with a distinct color
- Memory Access: access object with the correct color

Tagged Memory

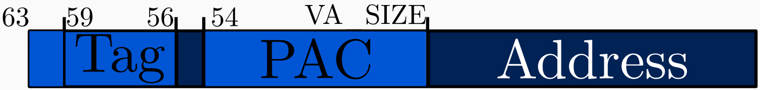
- Assigning metadata to memory chunks
- Each N-bytes of memory are tagged with a M-bit tag

Tagged Memory

- Assigning metadata to memory chunks
- Each N-bytes of memory are tagged with a M-bit tag
- Hardware available: Memory Tagging Extension (MTE) in ARMv8.5
- A 4-bit tag for every 16-bytes of memory

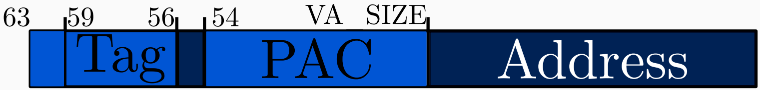
Tagged Memory

- Assigning metadata to memory chunks
- Each N-bytes of memory are tagged with a M-bit tag
- Hardware available: Memory Tagging Extension (MTE) in ARMv8.5
- A 4-bit tag for every 16-bytes of memory
- Tag is transported in the upper, unused bits of the pointer



Tagged Memory

- Assigning metadata to memory chunks
- Each N-bytes of memory are tagged with a M-bit tag
- Hardware available: Memory Tagging Extension (MTE) in ARMv8.5
- A 4-bit tag for every 16-bytes of memory
- Tag is transported in the upper, unused bits of the pointer



- Google's MemTagSanitizer utilizes MTE for memory coloring

Tagged Memory Overhead

- Color needs to be stored in memory
- ARM MTE: 3%

Tagged Memory Overhead

- Color needs to be stored in memory
- ARM MTE: 3%
- Detection probability of 93%

Tagged Memory Overhead

- Color needs to be stored in memory
- ARM MTE: 3%
- Detection probability of 93%
- High detection probability for tag sizes of 16-bits
- Increases memory overhead to 12%

Tagged Memory Overhead

- Color needs to be stored in memory
- ARM MTE: 3%
- Detection probability of 93%
- High detection probability for tag sizes of 16-bits
- Increases memory overhead to 12%
- Security $\leftrightarrow$ Memory Overhead
- Mainly used for debugging

CrypTag

CrypTag

- Goal: Enforcing **physical** and **logical** memory safety
- Maximize security guarantees and keep overhead at a minimum

CrypTag

- Goal: Enforcing **physical** and **logical** memory safety
- Maximize security guarantees and keep overhead at a minimum
- Combining transparent memory encryption and memory coloring

Concept

- Each memory object is tagged with a color
- Color is transported in upper bits of the pointer

Concept

- Each memory object is tagged with a color
- Color is transported in upper bits of the pointer
- Color tweaks the encryption of the memory object
- Each memory object is encrypted with a distinct color
- Accessing memory object with correct color decrypts it

Concept

- Each memory object is tagged with a color
- Color is transported in upper bits of the pointer
- Color tweaks the encryption of the memory object
- Each memory object is encrypted with a distinct color
- Accessing memory object with correct color decrypts it

  → No color storage overhead

  → No memory traffic overhead

  → Increase color size

Color Mismatch

- Memory encryption
  - Color mismatch decrypts with wrong tweak
  - Security policy **S1**

Color Mismatch

- Memory encryption
  - Color mismatch decrypts with wrong tweak
  - Security policy **S1**
- Memory encryption and authentication
  - Color mismatch triggers an authentication error
  - Security policy **S2**

Implementation

Hardware Support

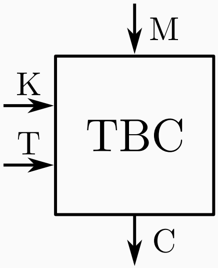
- Minimal hardware changes

Hardware Support

- Minimal hardware changes
- Instruction to set color in unused upper bits of a pointer
- MMU ignores theses bits in address translation
- Cache is extended to store the color
- CrypTag allows sub-cache line granularity

Memory Encryption

- Based on a system with transparent memory encryption
- Encryption or encryption and authentication
- Tweakable block cipher
- MEMSEC [Wer+17]
  - **S1**: QARMA
  - **S2**: ASCON



Software Support

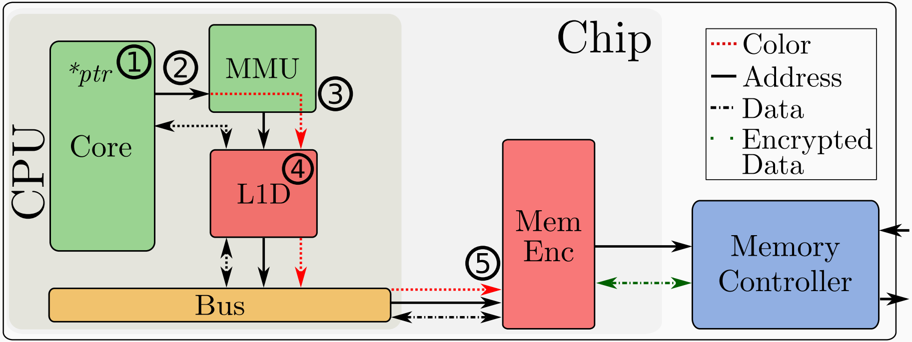
- Protection of heap, local, and global data

Software Support

- Protection of heap, local, and global data
- Automatic instrumentation:
  - LLVM toolchain for local and global data
  - Tiny runtime library for heap allocations

```c
void* __wrap_malloc(size_t size) {
  size = roundup(size);
  void *ptr = __real_malloc(size);
  if (ptr == NULL) return NULL;
  return mstp(ptr);
}
```

CrypTag


Chip
CPU
*ptr
1
2
3
4
5
Core
MMU
L1D
Bus
Mem
Enc
Memory
Controller
Color
Address
Data
Encrypted
Data

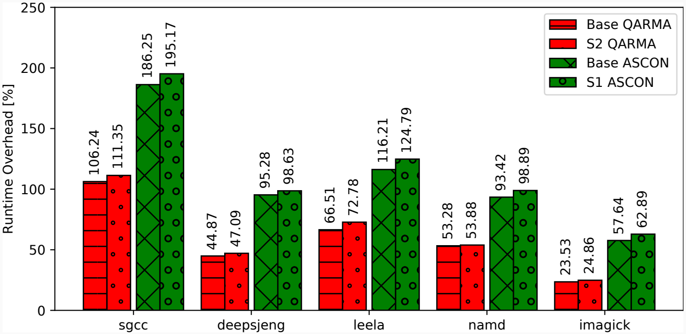
Evaluation

Hardware Overhead

- Hardware overhead of less than 93%
- Tag generation and transportation
- Cache overhead
  - Between 1.56% and 19.53%

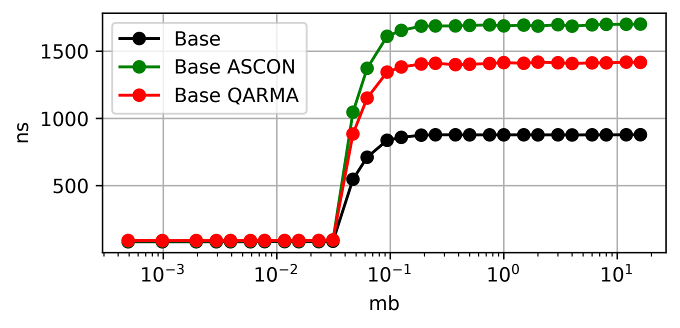
Runtime Overhead

- SPEC2017: 5.2% and 6.1%
- SciMark2: 3.9% and 4.79%
- MiBench: 1.5% and 4.9%



Prototype Limitations

- On top of the memory encryption overhead
- MEMSEC: up to 110%
- Commercial solutions [Rob20]: 5% to 26%



Security Discussion

Security of CrypTag

- CrypTag is a probabilistic scheme
- Large tag sizes enables a high detection probability

Security of CrypTag

- CrypTag is a probabilistic scheme
- Large tag sizes enables a high detection probability
- Spatial memory safety:
  - **S1**: Pseudorandom value
  - **S2**: Authentication error

Security of CrypTag

- CrypTag is a probabilistic scheme
- Large tag sizes enables a high detection probability
- Spatial memory safety:
  - **S1**: Pseudorandom value
  - **S2**: Authentication error
- Temporal memory safety:
  - **S1**: Pseudorandom value
  - **S2**: Authentication error

Security of CrypTag

- CrypTag is a probabilistic scheme
- Large tag sizes enables a high detection probability
- Spatial memory safety:
  - **S1**: Pseudorandom value
  - **S2**: Authentication error
- Temporal memory safety:
  - **S1**: Pseudorandom value
  - **S2**: Authentication error
- Physical memory safety

Conclusion

Conclusion

- Extension to systems already featuring a transparent memory encryption
- Memory coloring scheme utilizing transparent memory encryption
- Low performance ($< 6.2\%$) and hardware overhead ($< 1\%$)
- Larger tag sizes (e.g., 25-bits)
- Suitable as a security countermeasure
- RISC-V implementation and custom LLVM-based toolchain

Thank you!

CrypTag: Thwarting Physical and Logical Memory Vulnerabilities using Cryptographically Colored Memory

Pascal Nasahl, Robert Schilling, Mario Werner, Jan Hoogerbrugge, Marcel Medwed, Stefan Mangard

AsiaCCS'21, June 7-11, 2021

IAIK – Graz University of Technology

References

J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In: USENIX Security Symposium. 2008.

M. Miller. Trends, Challanges, and Strategic Shifts in the Software Vulnerability Mitigation Landscape. In: BlueHat IL (2019).

MITRE. CWE Top 25 Most Dangerous Software Errors. 2019.

H. Nurmi. Sniff, there leaks my BitLocker key. 2020.

A. Roberto-Maria. Memory Protection for the ARM Architecture. 2020.

L. Szekeres, M. Payer, T. Wei, and D. Song. SoK: Eternal War in Memory. In: IEEE Symposium on Security and Privacy – S&P. 2013.

M. Werner, T. Unterluggauer, R. Schilling, D. Schaffenrath, and S. Mangard. Transparent memory encryption and authentication. In: Field Programmable Logic and Applications – FPL. 2017.