

SecLabel: Enhancing RISC-V Platform Security with Labelled Architecture

Zhenyu Ning^{1,2}, Yinqian Zhang³, and Fengwei Zhang²

¹Wayne State University, ²Southern University of Science and Technology,

³The Ohio State University





Outline

- Introduction
- Pointer Integrity
- Memory Boundary Protection
- Dynamic Taint Analysis
- Implementation
- Conclusion



Outline

- **Introduction**
- Pointer Integrity
- Memory Boundary Protection
- Dynamic Taint Analysis
- Implementation
- Conclusion



Introduction

- The RISC-V architecture is well-known for its open nature.
 - Open Source, No License fee
 - Open to new design and extension
- Open to challenge.
 - Security problems in x86 and ARM architecture remains on RISC-V platforms.
 - E.g., pointer integrity, memory boundary protection, and dynamic taint analysis.



Introduction

Any effective defense on RISC-V?

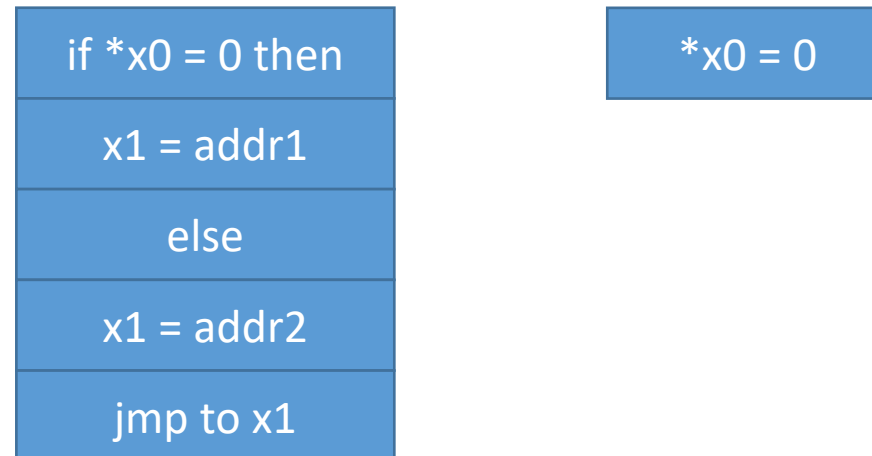


Outline

- Introduction
- ***Pointer Integrity***
- Memory Boundary Protection
- Dynamic Taint Analysis
- Implementation
- Conclusion

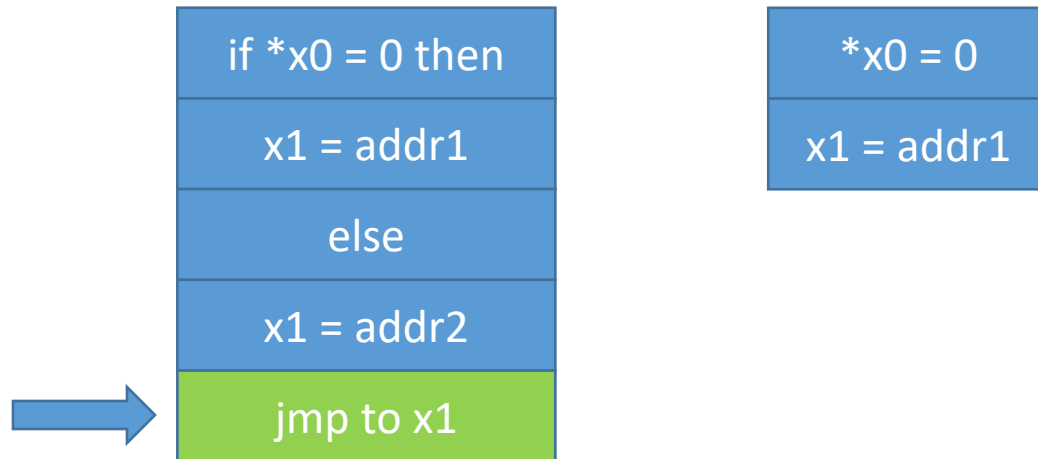
Pointer Integrity

- To ensure that the pointer is not corrupted.
 - Code-pointer Integrity and Data-pointer Integrity.



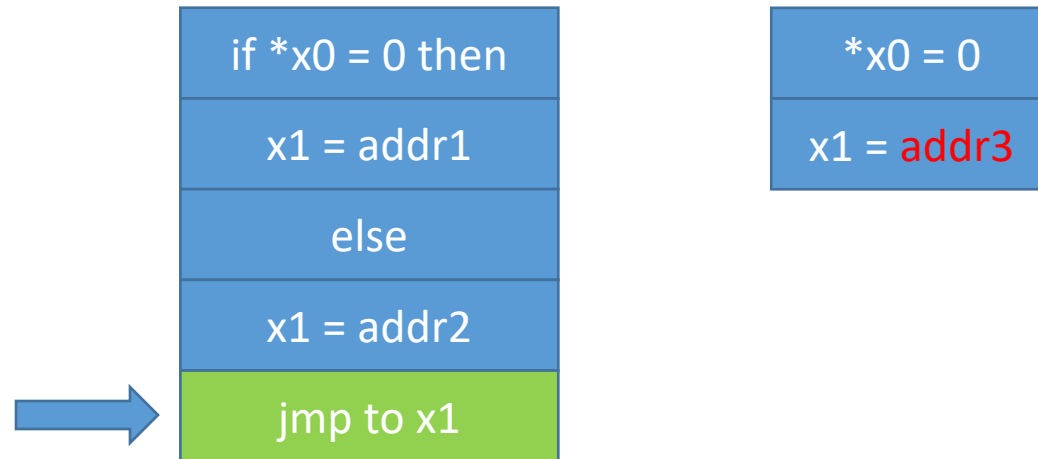
Pointer Integrity

- To ensure that the pointer is not corrupted.
 - Code-pointer Integrity and Data-pointer Integrity.



Pointer Integrity

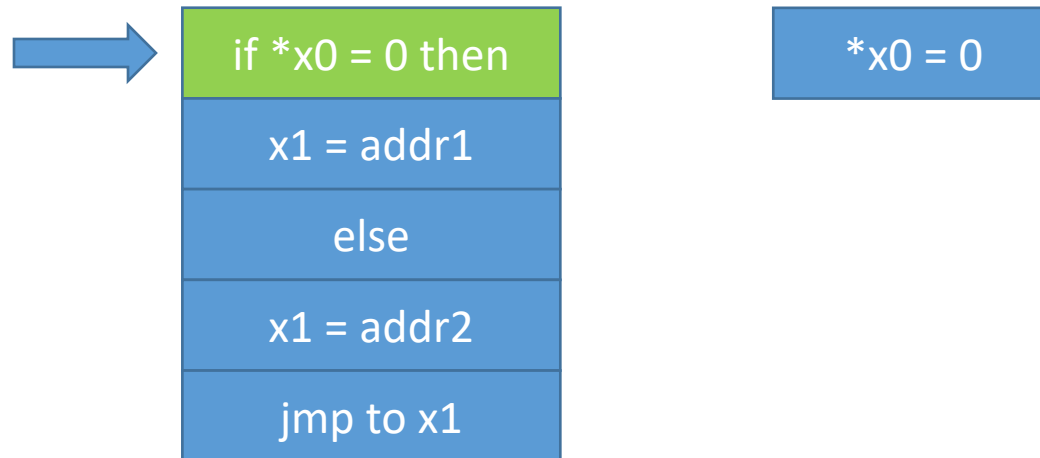
- To ensure that the pointer is not corrupted.
 - Code-pointer Integrity and Data-pointer Integrity.



**Code-pointer
Attack**

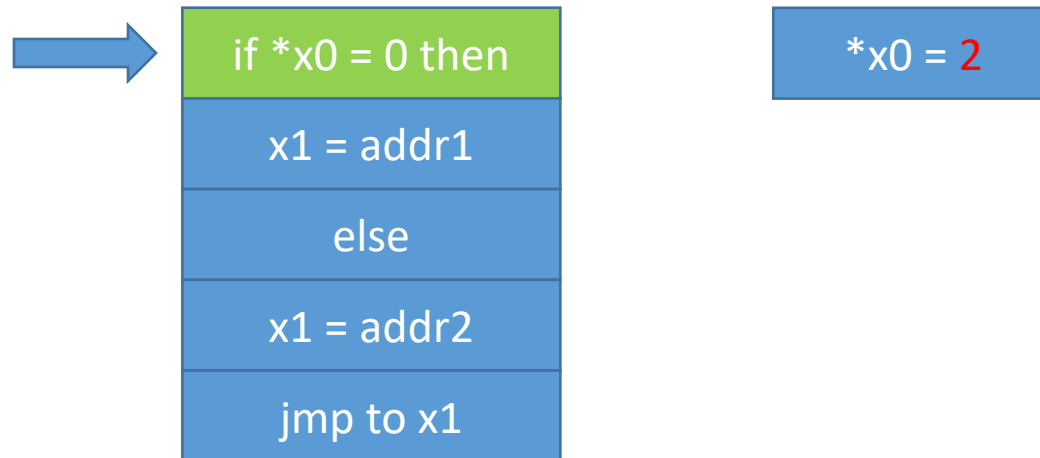
Pointer Integrity

- To ensure that the pointer is not corrupted.
 - Code-pointer Integrity and Data-pointer Integrity.



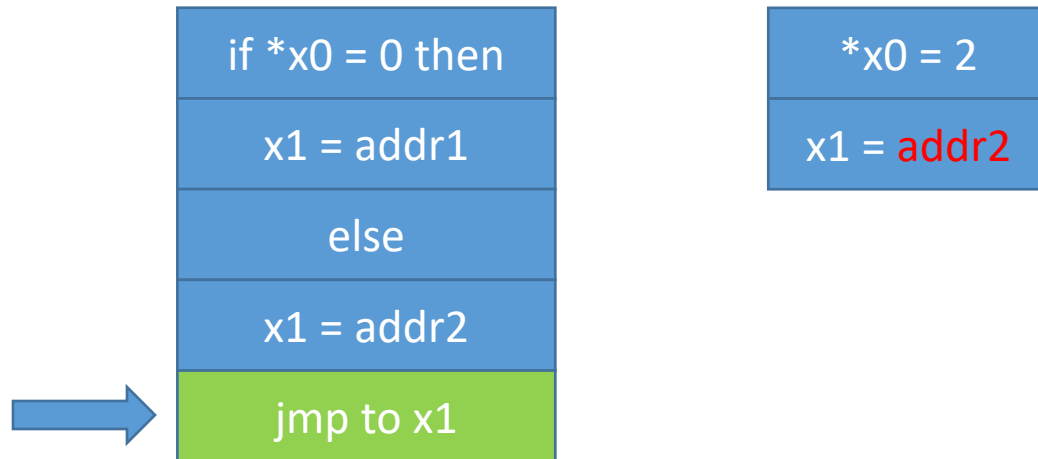
Pointer Integrity

- To ensure that the pointer is not corrupted.
 - Code-pointer Integrity and Data-pointer Integrity.



Pointer Integrity

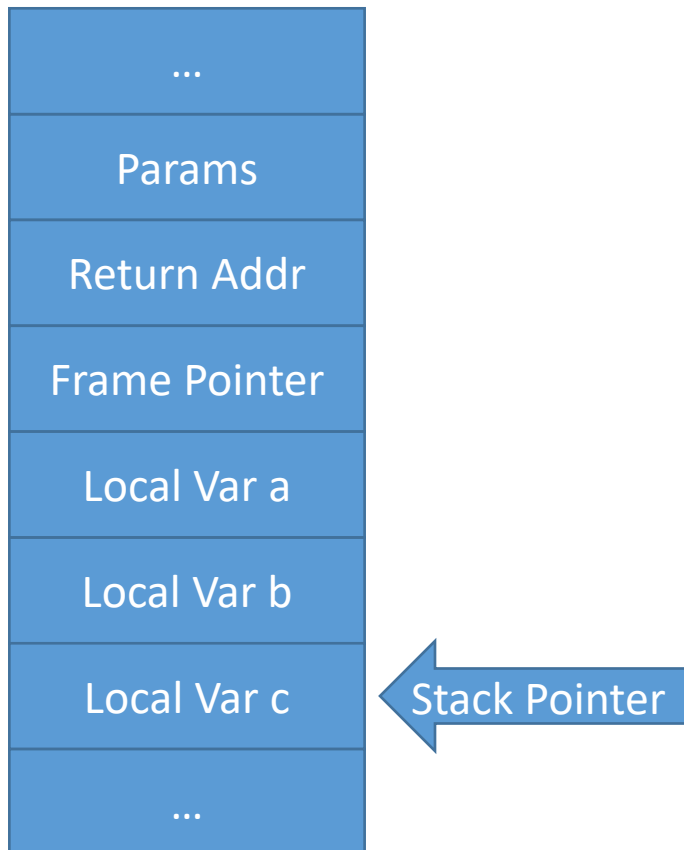
- To ensure that the pointer is not corrupted.
 - Code-pointer Integrity and Data-pointer Integrity.



**Data-pointer
Attack**

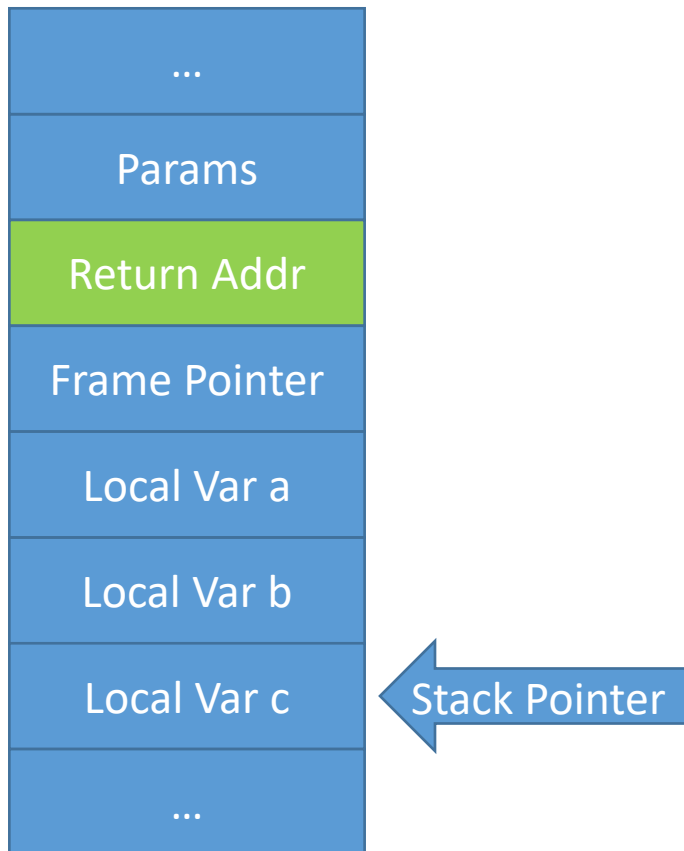
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.



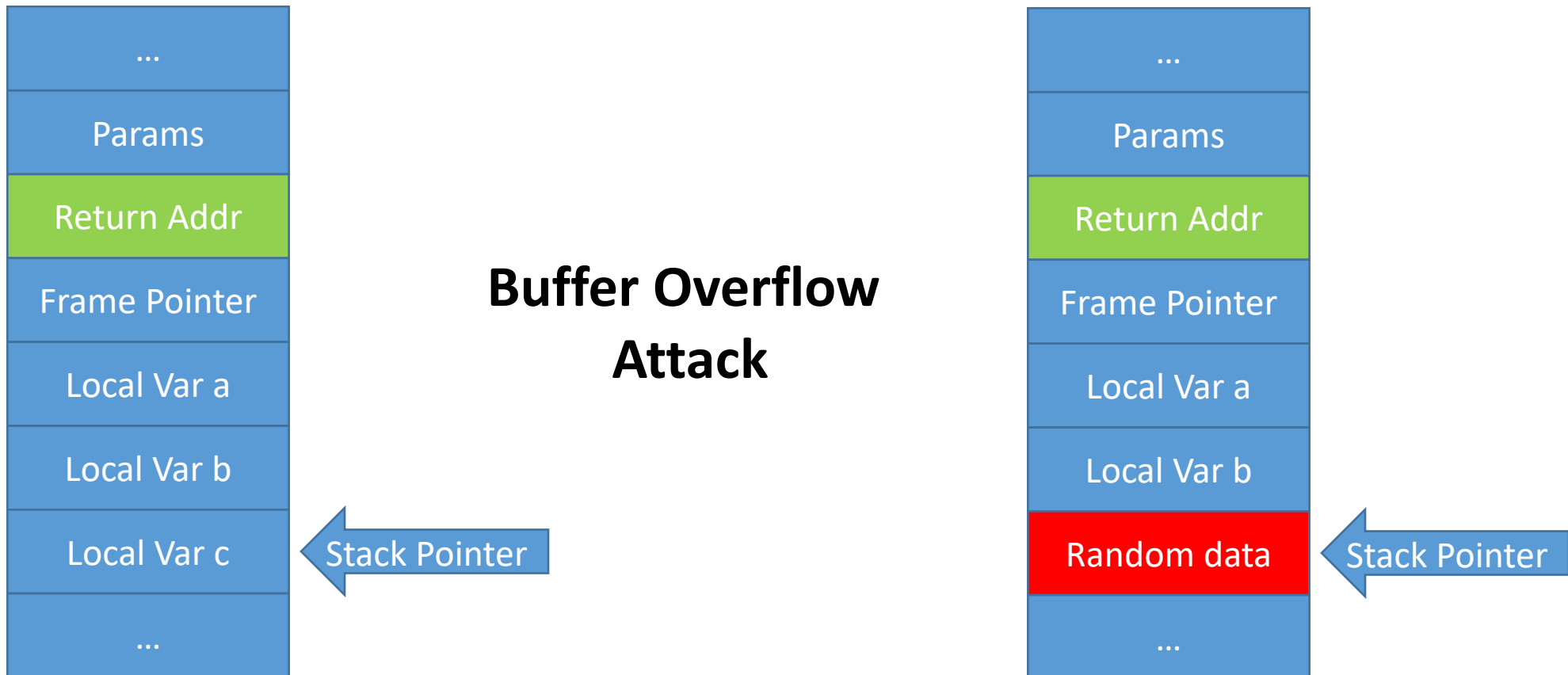
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.



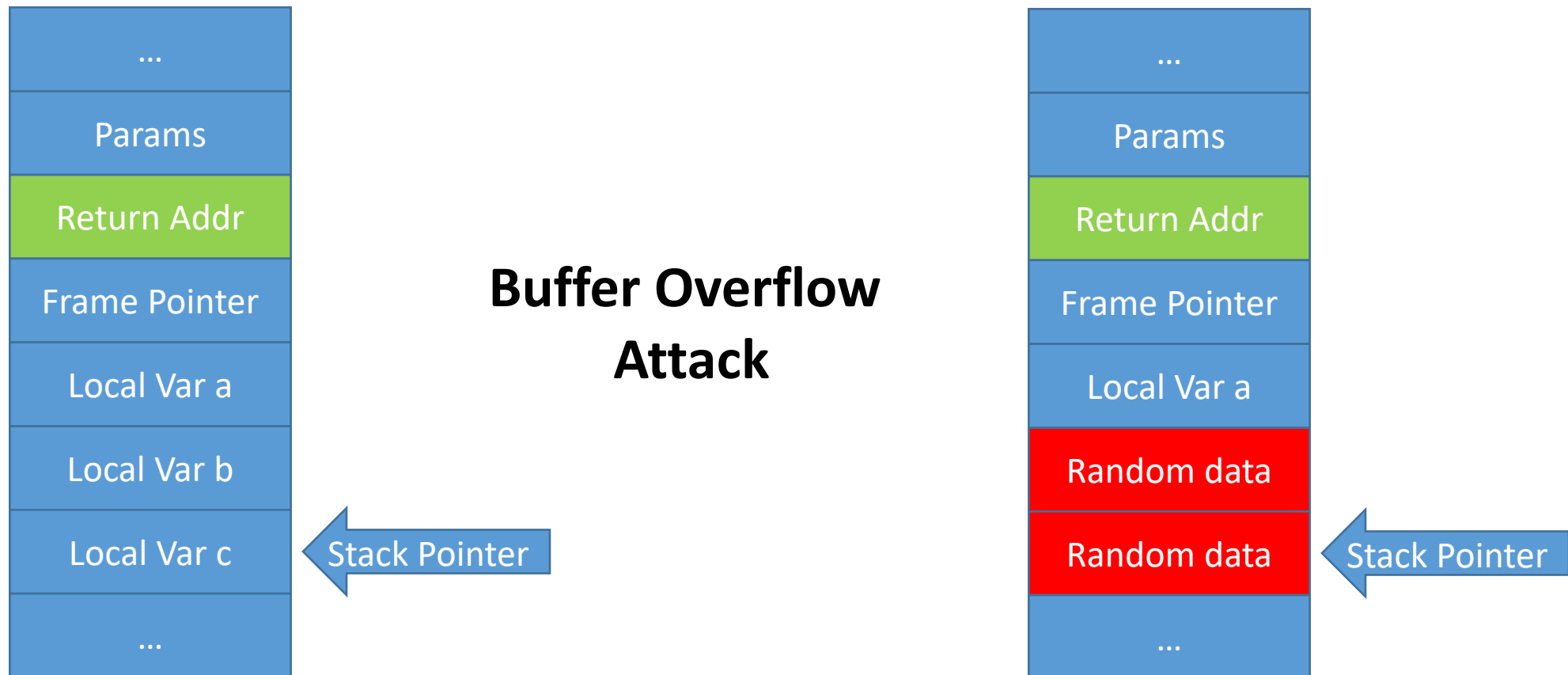
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.



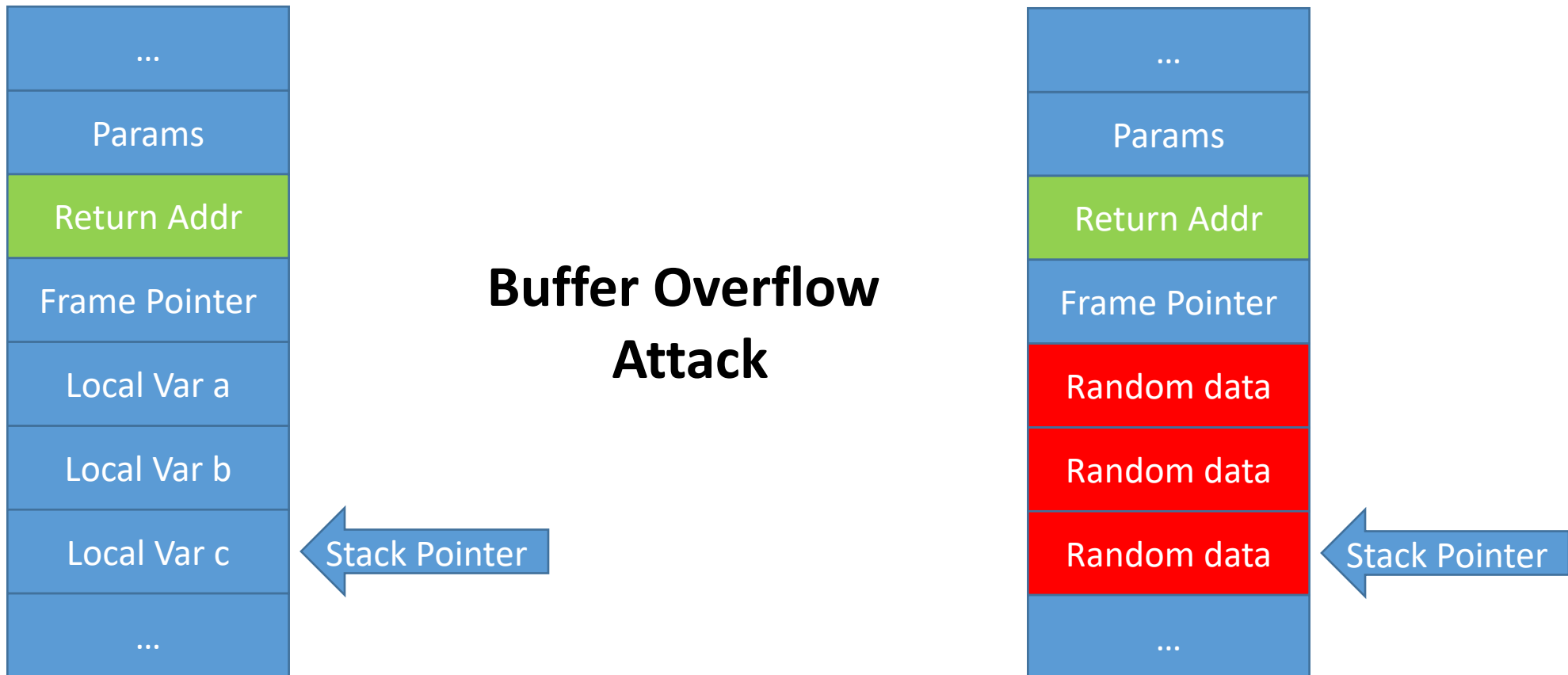
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.



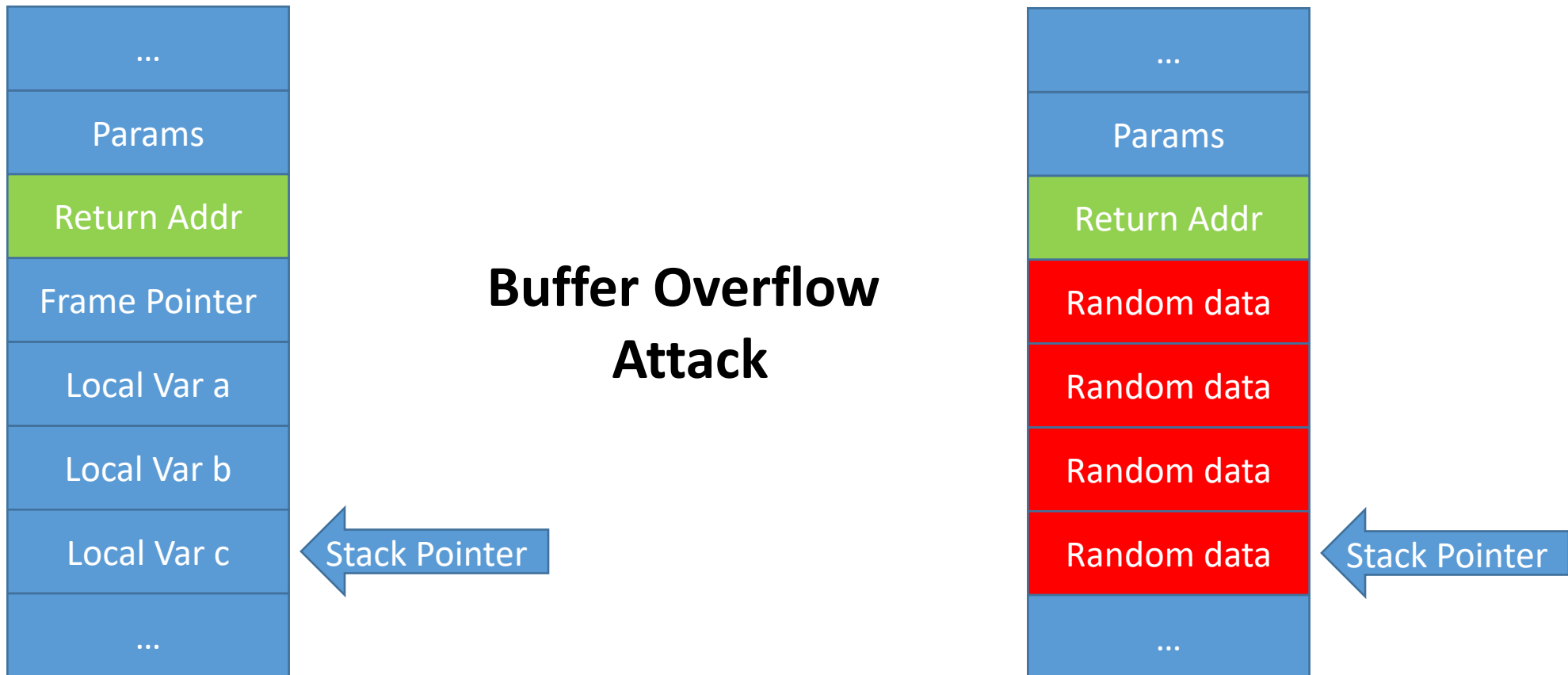
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.



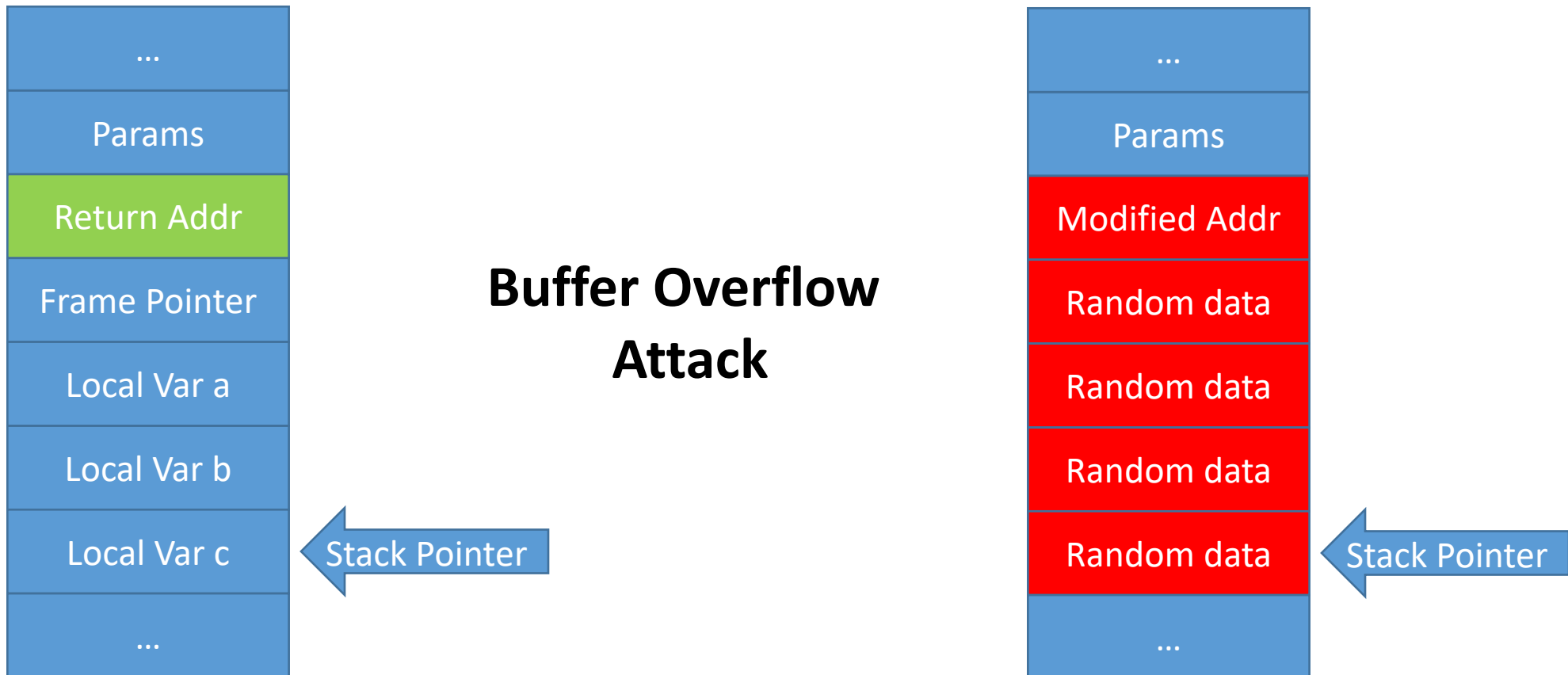
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.



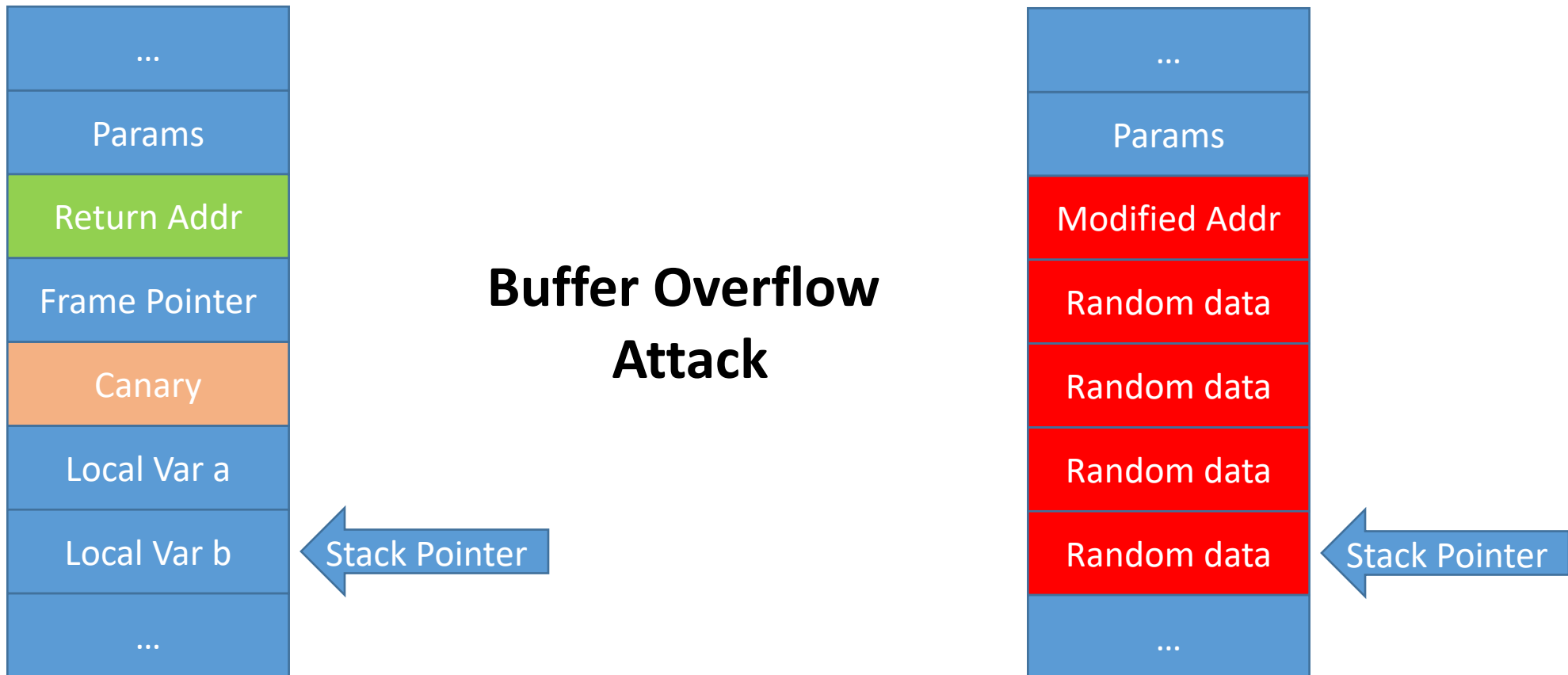
Pointer Integrity: Buffer Overflow

- Start of the attack: In most cases, a buffer overflow vulnerability.



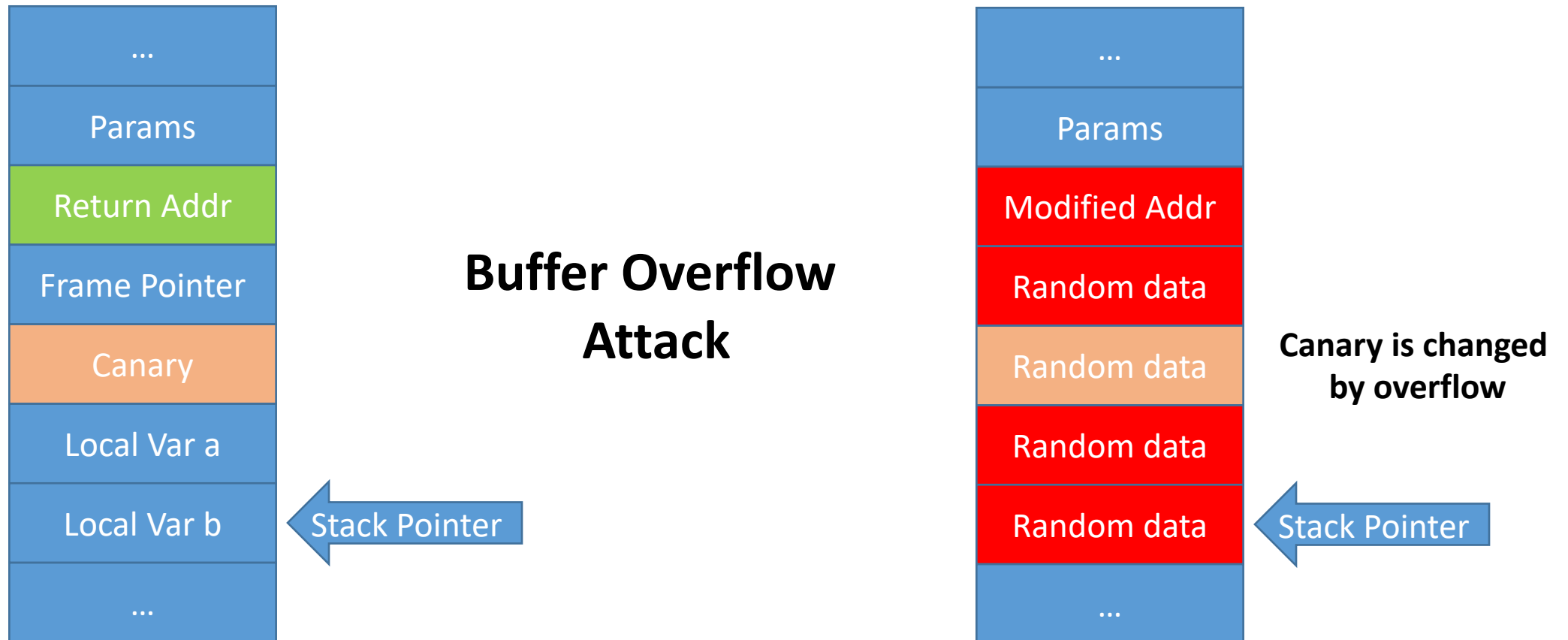
Pointer Integrity: Canary

- **Stack Canary^[1]**: The most widely used defense to buffer overflow attack.



Pointer Integrity: Canary

- **Stack Canary^[1]**: The most widely used defense to buffer overflow attack.





Pointer Integrity: Canary

- **Stack Canary**^[1]: The most widely used defense to buffer overflow attack.
- Weakness:
 - Easy to bypass^[2]
 - Not efficient to defend against data-pointer attack

Pointer Integrity: PAC

- **Pointer Authentication Code**^[3] is introduced in 64-bit ARMv8.3 architecture.

63

0



A pointer in 64-bit system

Is it really necessary to use a 64-bit address?



Pointer Integrity: PAC

Is it really necessary to use a 64-bit address?

- 2^{64} bit = 16384 PB = 16.8 millions TB = 17.2 billions GB
- **Summit:** 10 PB memory
- **Sunway TaihuLight:** 1.32 PB memory
- **Linux:** Up to 128 TB virtual memory
- **Windows:** Up to 16 TB virtual memory

Pointer Integrity: PAC

- **Pointer Authentication Code**^[3] is introduced in 64-bit ARMv8.3 architecture.

63

0



A pointer in 64-bit system

Pointer Integrity: PAC

- **Pointer Authentication Code**^[3] is introduced in 64-bit ARMv8.3 architecture.



- Pointer Value + 64-bit Context Value + 128-bit Secret Key => PAC
- Up to 48 bits for virtual address, and at least 7 bits for PAC



Pointer Integrity: PAC

- PAC is good, but the deployment is painful.
 - The mechanism is released with ARMv8.3 architecture since 2016.
 - ARM does not release any processor with ARMv8.3 till now.
- The only processors with PAC support are Apple A12 and A13.
 - Closed ecosystem.
 - No available to system developers.



Pointer Integrity: RISC-V

- RISC-V based PAC
 - A group of new hardware instructions
 - Forge PAC, examine PAC, strip PAC
 - New registers for storing the 128-bit secret key
 - Secret keys for data pointers and code pointers
 - Hardware-based crypto engine
 - Generate PAC from pointer and 64-bit context value

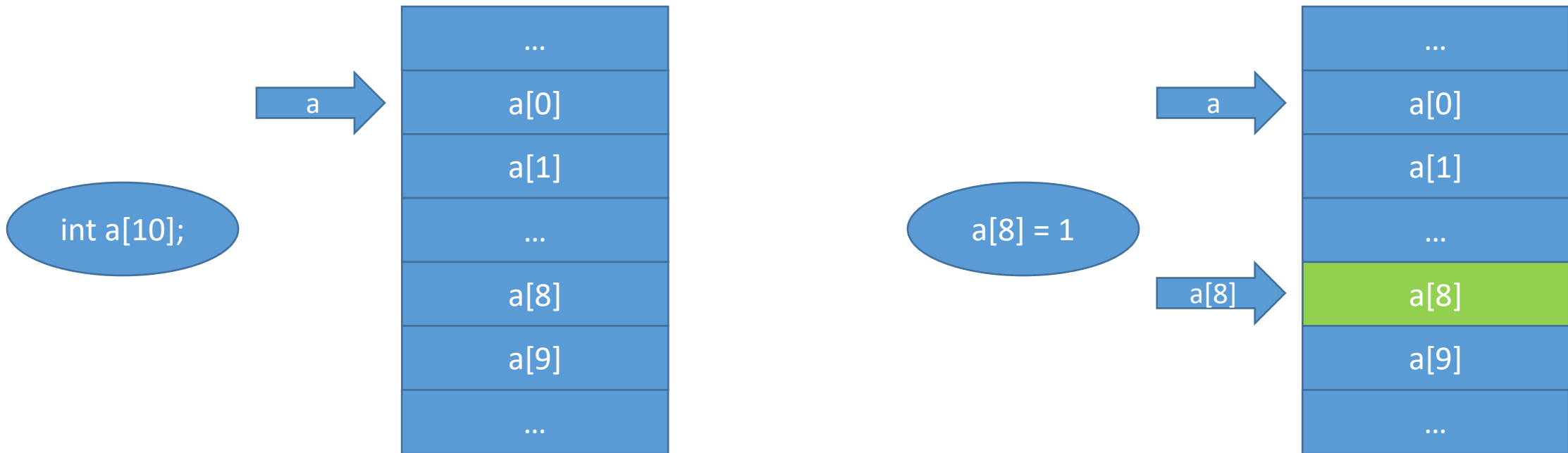


Outline

- Introduction
- Pointer Integrity
- **Memory Boundary Protection**
- Dynamic Taint Analysis
- Implementation
- Conclusion

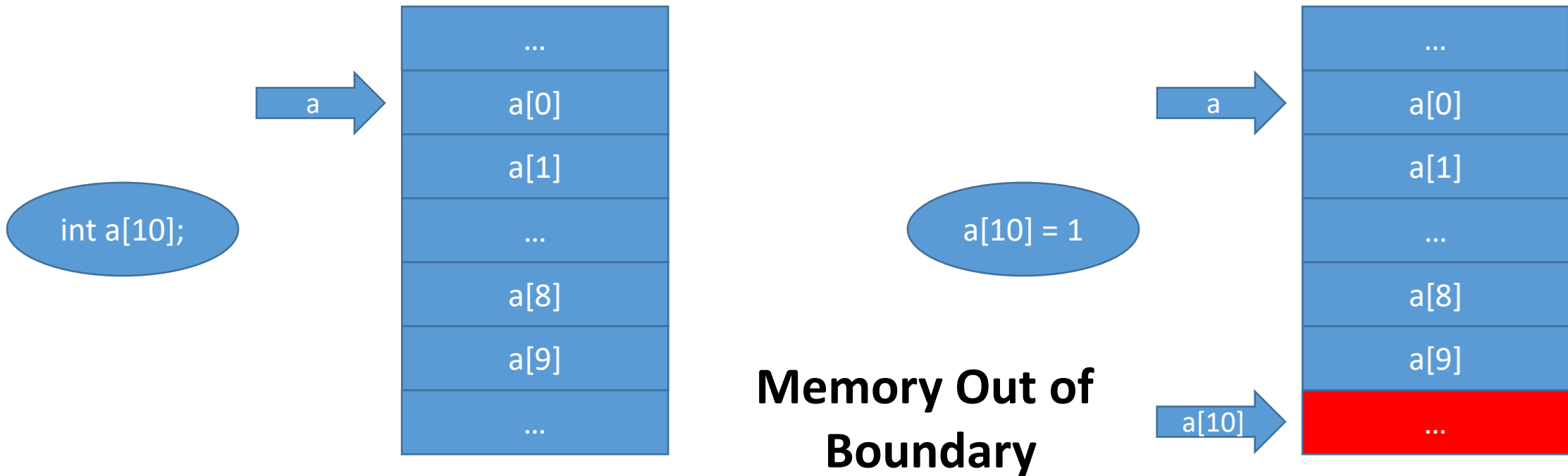
Memory Boundary Protection

- To ensure the memory access won't go out of its expected boundary.



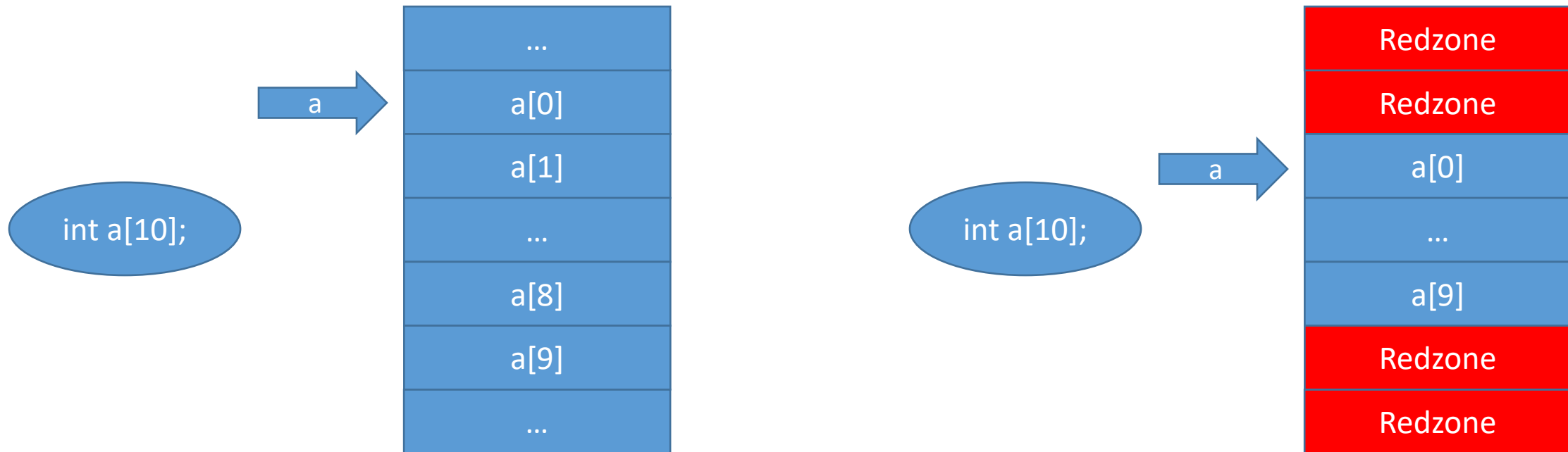
Memory Boundary Protection

- To ensure the memory access won't go out of its expected boundary.



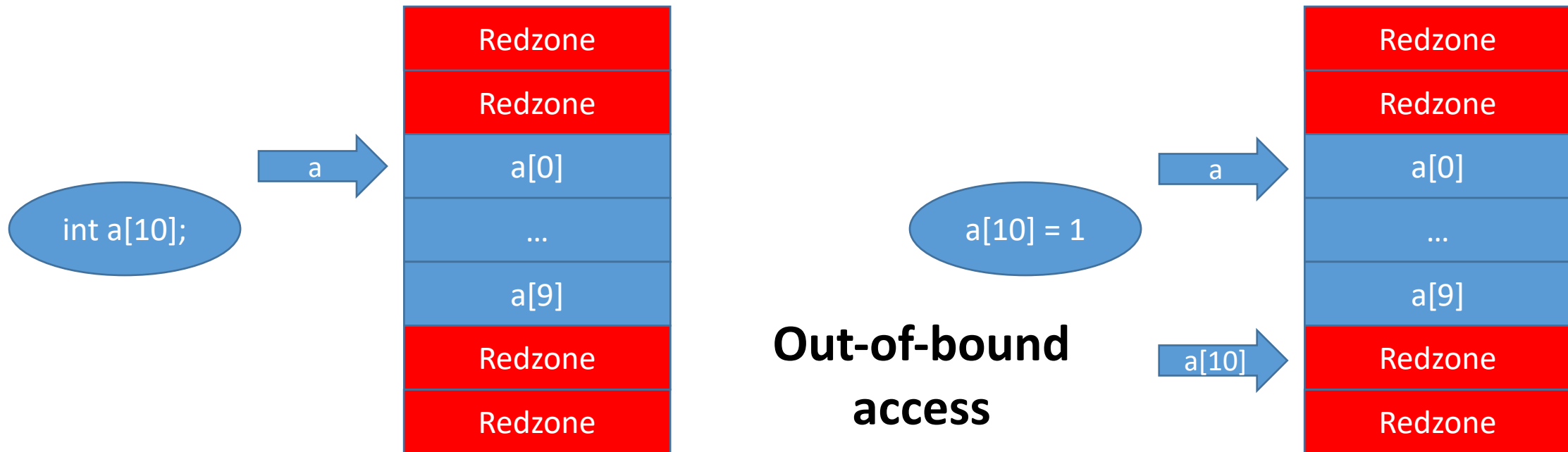
Memory Boundary Protection: Address Sanitizer

- **Address Sanitizer^[4]**: Use redzones to detect out-of-bound access.



Memory Boundary Protection: Address Sanitizer

- **Address Sanitizer^[4]**: Use redzones to detect out-of-bound access.



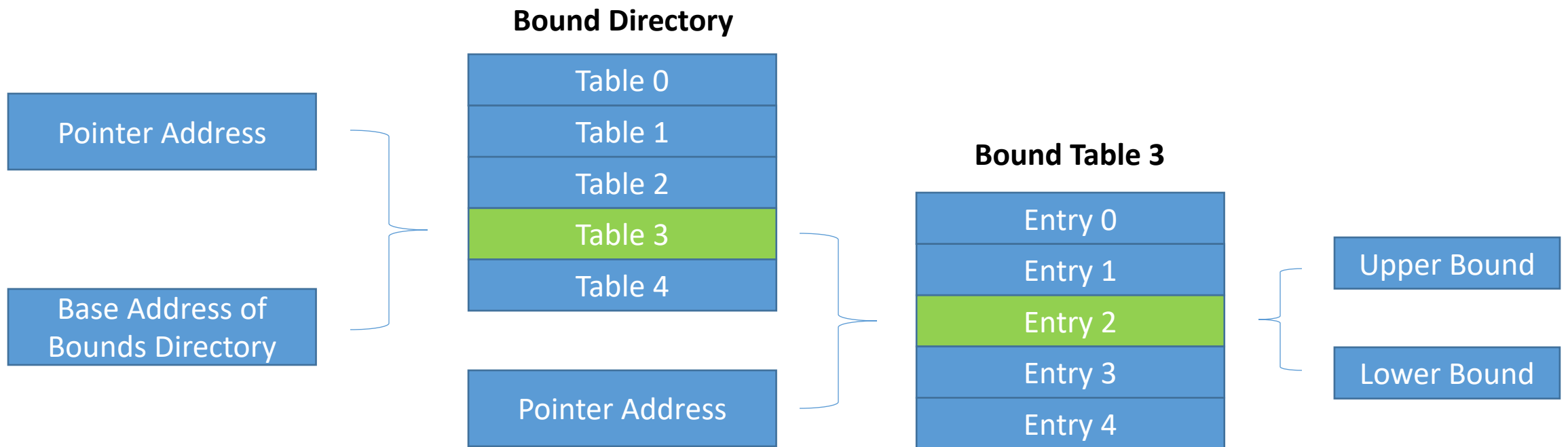


Memory Boundary Protection: Address Sanitizer

- **Address Sanitizer^[4]**: Use redzones to detect out-of-bound access.
- Weakness:
 - Large memory overhead
 - Large performance overhead
 - False negative is possible

Memory Boundary Protection: Intel MPX

- **Intel MPX^[5]**: An architecture extension dedicated for memory bound protection.





Memory Boundary Protection: Intel MPX

- **Intel MPX^[5]**: An architecture extension dedicated for memory bound protection.
- Weakness:
 - Performance overhead for two-layer translation
 - Multithread not support
 - Not production ready, support will be removed from GCC 9

Memory Boundary Protection : RISC-V

- RISC-V based Memory Boundary Protection



- Use the head bits for memory bounds
 - 9 bits if PAC is implemented
 - 16 bits if PAC is not implemented
 - More bits in 128-bit RISC-V architecture^[6]



Outline

- Introduction
- Pointer Integrity
- Memory Boundary Protection
- **Dynamic Taint Analysis**
- Implementation
- Conclusion

Dynamic Taint Analysis

- Analysis the information flow of specific objects.
- Example scenario: Privacy leakage detection

```
.....  
char* password = getInput();  
char* copied = copy(password);  
printf("copied: %s\n", copied);  
.....
```

Taint Source

getInput

Taint Sink

printf

Dynamic Taint Analysis

- Analysis the information flow of specific objects.
- Example scenario: Privacy leakage detection

```
.....  
char* password = getInput();  
char* copied = copy(password);  
printf("copied: %s\n", copied);  
.....
```

Taint Source

getInput

Taint Sink

printf

Tainted Variable

password

Dynamic Taint Analysis

- Analysis the information flow of specific objects.
- Example scenario: Privacy leakage detection

```
.....  
char* password = getInput();  
char* copied = copy(password);  
printf("copied: %s\n", copied);  
.....
```

Taint Source

getInput

Taint Sink

printf

Tainted Variable

password
copied

Dynamic Taint Analysis

- Analysis the information flow of specific objects.
- Example scenario: Privacy leakage detection

```
.....  
char* password = getInput();  
char* copied = copy(password);  
printf("copied: %s\n", copied);  
.....
```

Taint Source

getInput

Taint Sink

printf

Tainted Variable

password
copied

Taint Path Founded!

Dynamic Taint Analysis

```
.....  
char* password = getInput();  
char* copied = copy(password);  
printf("copied: %s\n", copied);  
.....
```

Taint Source

getInput

Taint Sink

printf

Tainted Variable

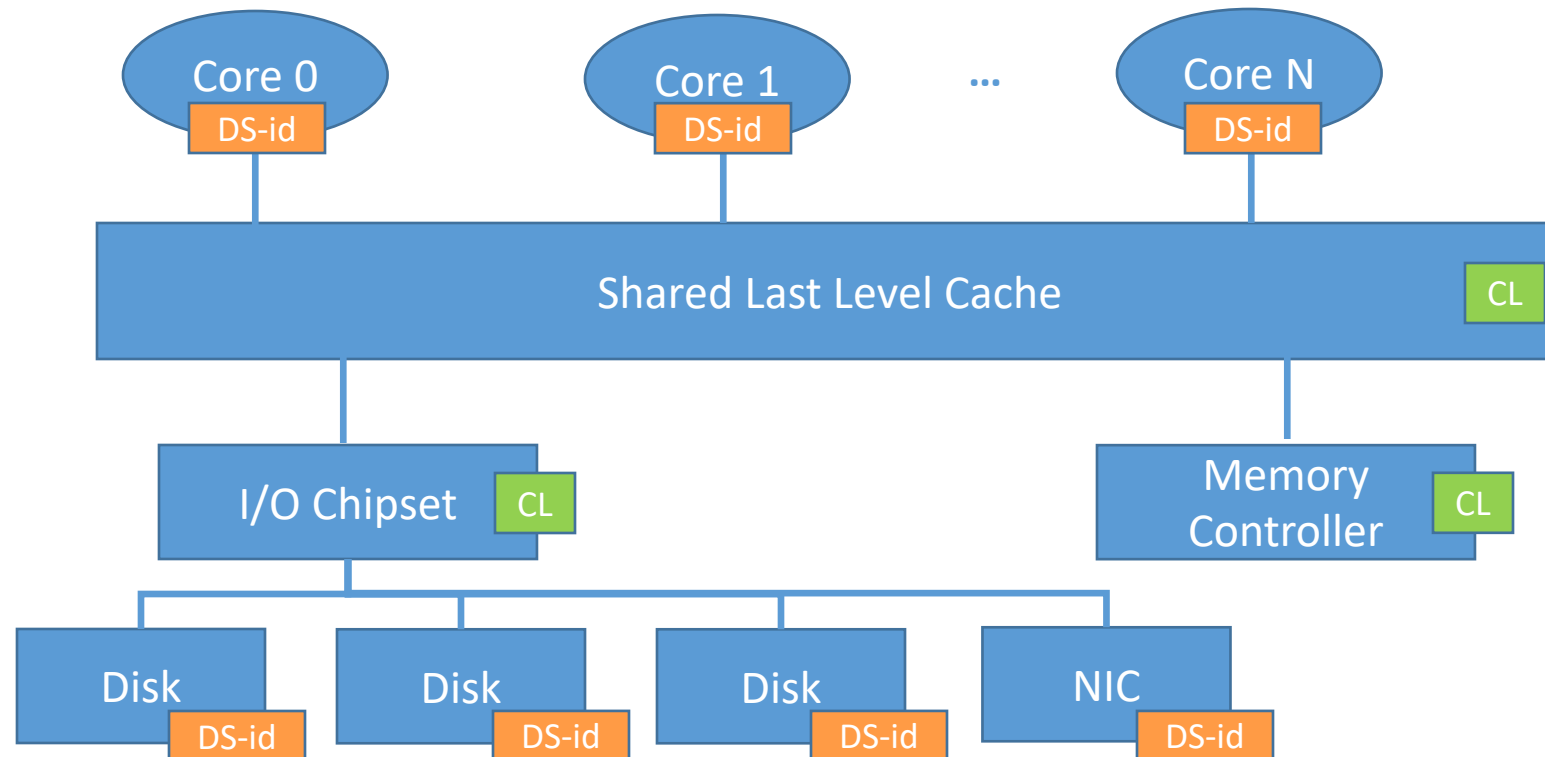
password

copied

- How to learn the taint propagation from *password* to *copied*?
- Heavy instrumentation
- Add tons of instructions to monitor the data flow

Dynamic Taint Analysis

- **Labelled RISC-V Architecture^[7]**: Every hardware request is attached with a label.



Dynamic Taint Analysis

- **Labelled RISC-V Architecture^[7]**: Every hardware request is attached with a label.
- Use the label to represent taint flag
 - Automatically propagation via hardware support
 - No instrumentation required
- Use the Control Logic (CL) to achieve detection

What about the propagation outside of hardware request?



Dynamic Taint Analysis

What about the propagation outside of hardware request?

- Allocate a few bits from the unused bits in 64-bit pointer
 - In coarse-grained analysis, 1 bit is sufficient
 - This bit automatically transfers during the execution of data operation instructions.
 - Feed to the *DS-id* register during hardware request

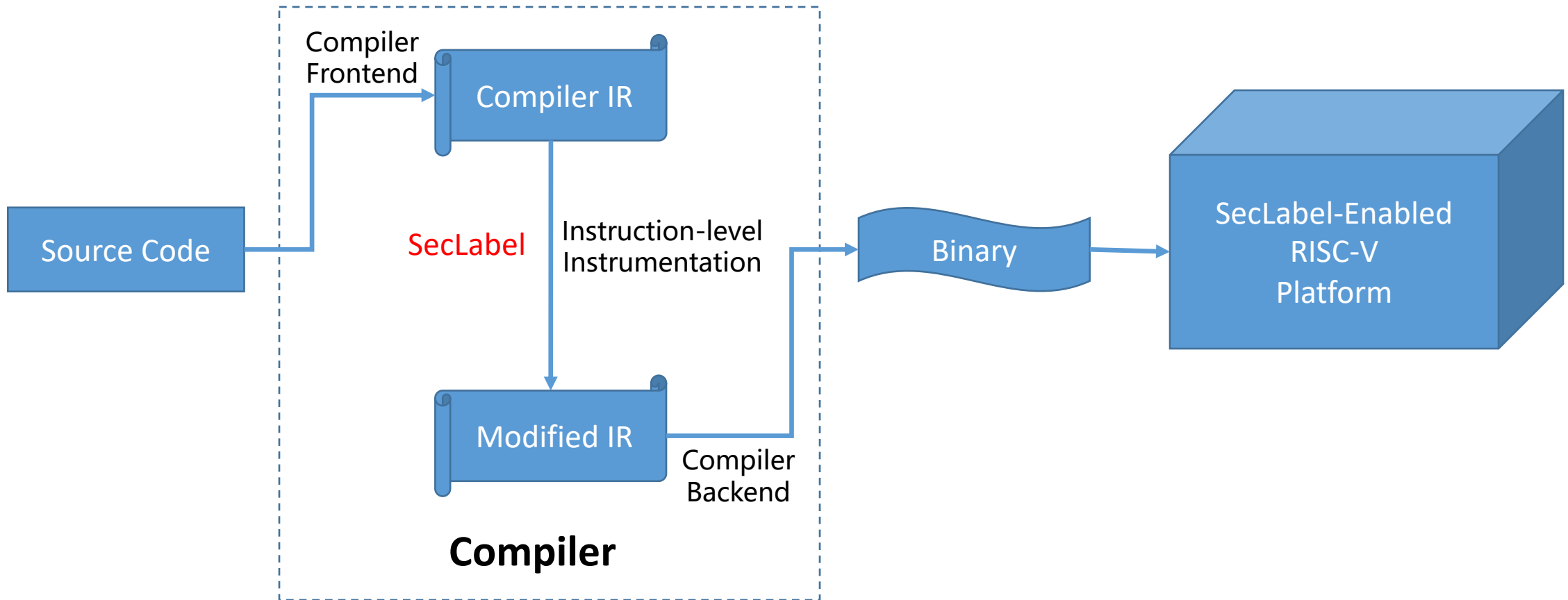


Outline

- Introduction
- Pointer Integrity
- Memory Boundary Protection
- Dynamic Taint Analysis
- **Implementation**
- Conclusion

Implementation

- **SecLabel**: Enhancing RISC-V Platform Security





Outline

- Introduction
- Pointer Integrity
- Memory Boundary Protection
- Dynamic Taint Analysis
- Implementation
- **Conclusion**

Conclusion

- In light of the PAC in ARMv8.3, we can leverage the open feature of RISC-V and implement similar mechanism for pointer integrity.
- With addition bits in the head of a pointer address in 64-bit or 128-bit RISC-V architecture, an enhanced memory boundary protection can be deployed.
- Combining the labelled RISC-V architecture and unused bits in an address, we are able to facilitate the existing dynamic taint analysis.



Reference

- [1] Cowan, Crispian, et al. "Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks." *USENIX Security Symposium*. Vol. 98. 1998.
- [2] Richarte, Gerardo. "Four different tricks to bypass stackshield and stackguard protection." *World Wide Web* 1. 2002.
- [3] Liljestrand, Hans, et al. "PAC it up: Towards pointer integrity using ARM pointer authentication." *28th USENIX Security*. 2019.
- [4] Serebryany, Konstantin, et al. "AddressSanitizer: A fast address sanity checker." *Presented as part of the 2012 USENIX Annual Technical Conference*. 2012.
- [5] Oleksenko, Oleksii, et al. "Intel MPX explained: An empirical study of intel MPX and software-based bounds checking approaches." *arXiv preprint arXiv:1702.00719*. 2017.
- [6] Wallach, Steve. "128-bit addressing in RISC-V and security." *5th RISC-V Workshop*. 2016.
- [7] Yu, Zihao, et al. "Labeled RISC-V: A new perspective on software-defined architecture." *CARVV*. 2017.

Thanks!



ningzy2019@mail.sustech.edu.cn