SoK: Shining Light on Shadow Stacks

Nathan Burow, Xinping Zhang, Mathias Payer

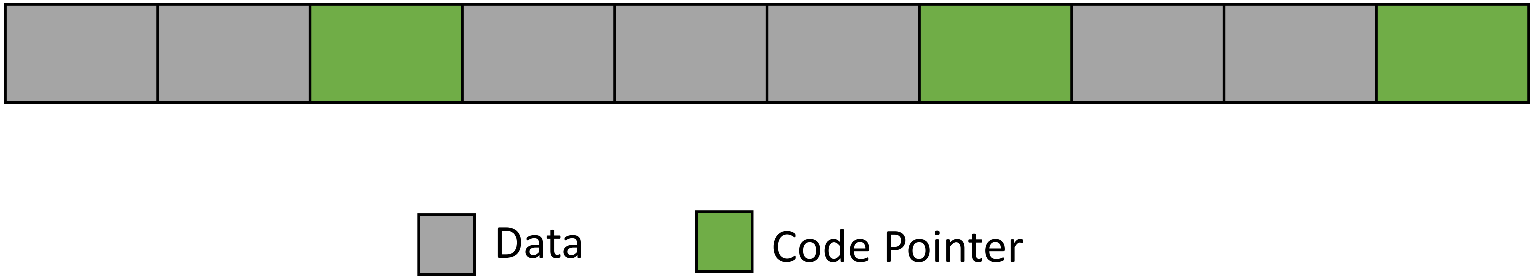
PURDUE UNIVERSITY

hexhive

EPFL

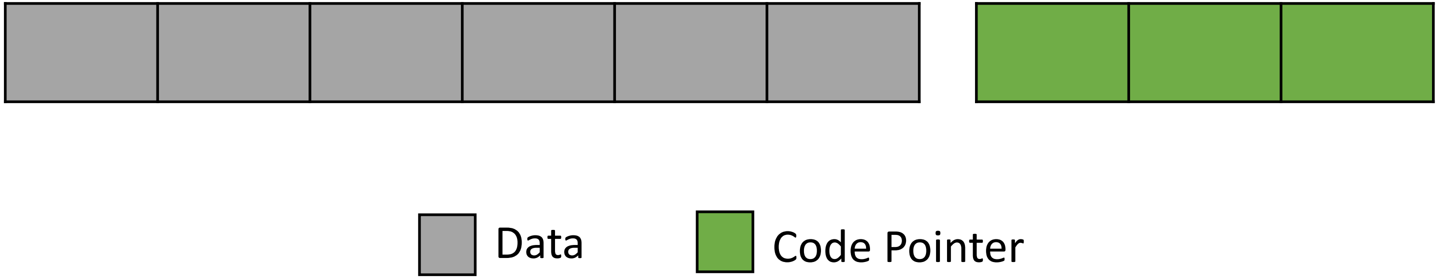
Control-Flow Hijacking (CFH)

- Microsoft: 70% of bugs are memory corruptions
- Control and Data Planes are interleaved
- Memory corruption → Control-Flow Hijacking



Control-Flow Hijacking (CFH)

- Microsoft: 70% of bugs are memory corruptions
- Control and Data Planes are interleaved
- Memory corruption → Control-Flow Hijacking

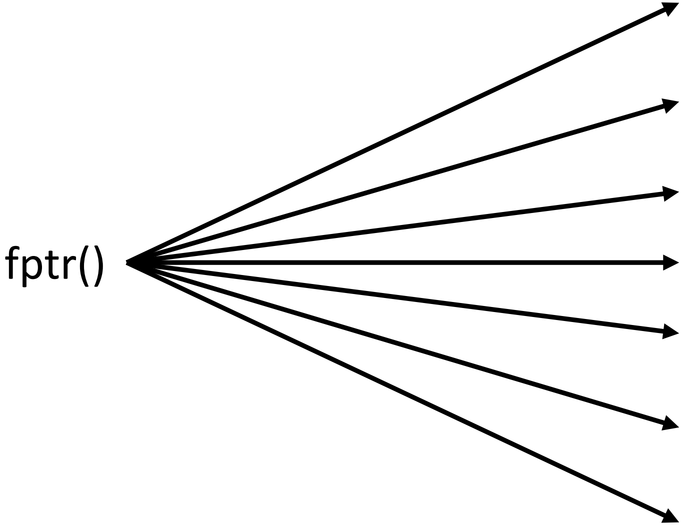


Forward Edge

- Function pointers; virtual calls
- Control-Flow Integrity (CFI) – statically calculates target sets

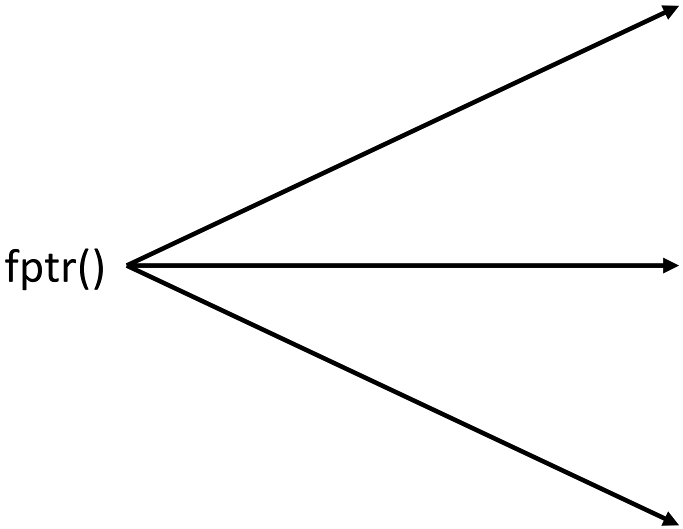
Forward Edge

- Function pointers; virtual calls
- Control-Flow Integrity (CFI) – statically calculates target sets



Forward Edge

- Function pointers; virtual calls
- Control-Flow Integrity (CFI) – statically calculates target sets

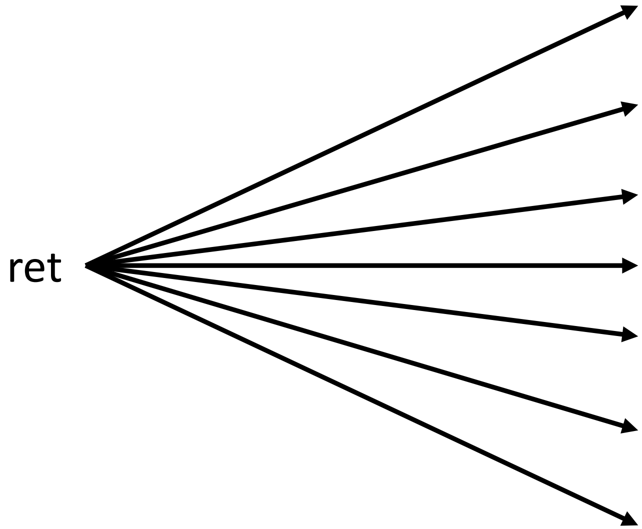


Backward Edge

- Return Instructions
- Does CFI style analysis work?

Backward Edge

- Return Instructions
- Does CFI style analysis work?



Backward Edge

- Return Instructions
- Does CFI style analysis work?

NO

Backward Edge

- CFI style target sets include **every** call site for the function
- Target sets are too large to provide meaningful protection

Security requires integrity for return addresses!

CFH Mitigation Today

- Seminal CFI paper by Abadi et. al. called for shadow stack
- See Burow et al CSUR 2017[1]
- Deployed versions by Microsoft / Google only cover forward edge

No equally strong defense for backward edge!

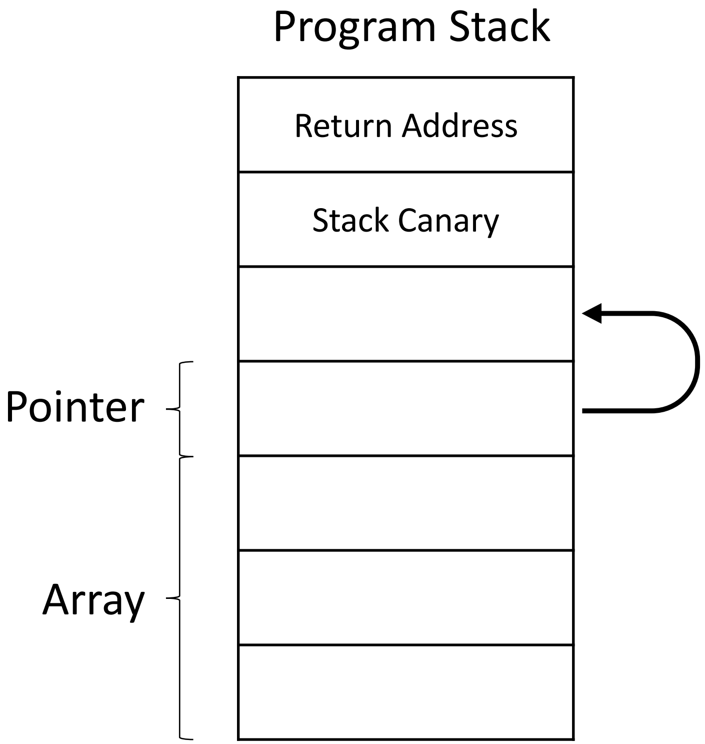
[1] Burow et. al. "Control-flow integrity: Precision, security, and performance." CSUR 2017.

Shadow Stacks

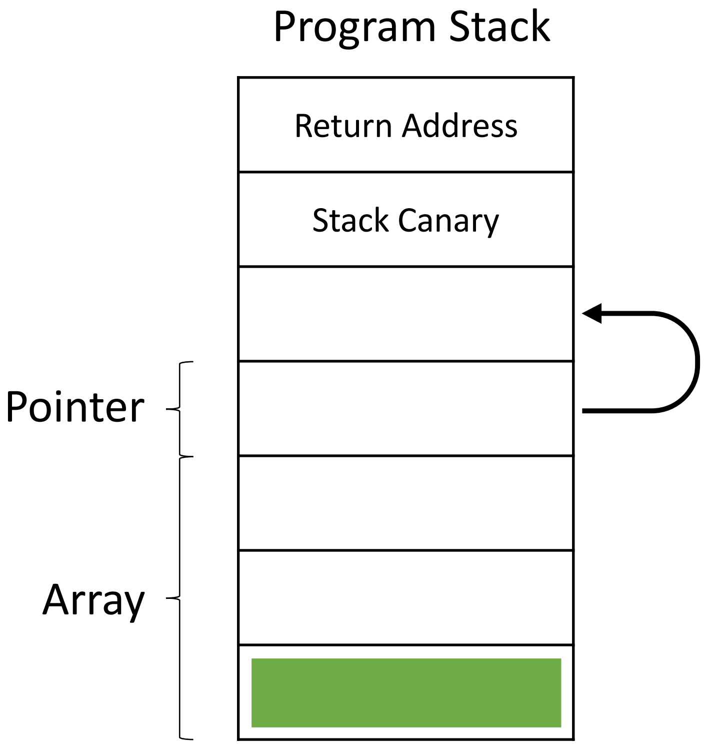
- Separate return addresses from data plane
- Provide integrity protection for return addresses
- Performant and highly compatible

Need to deploy Shadow Stack with CFI!

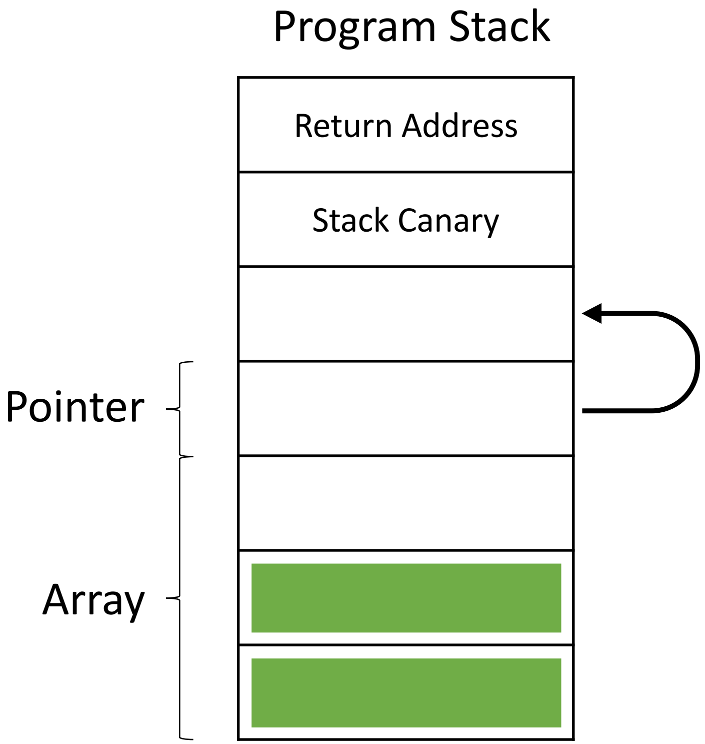
Control-Flow Hijacking Illustrated



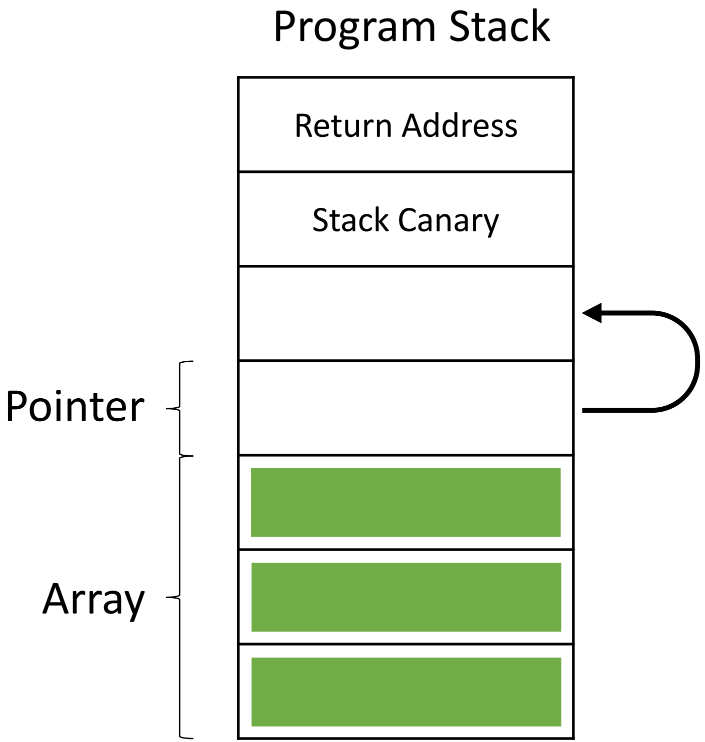
Control-Flow Hijacking Illustrated



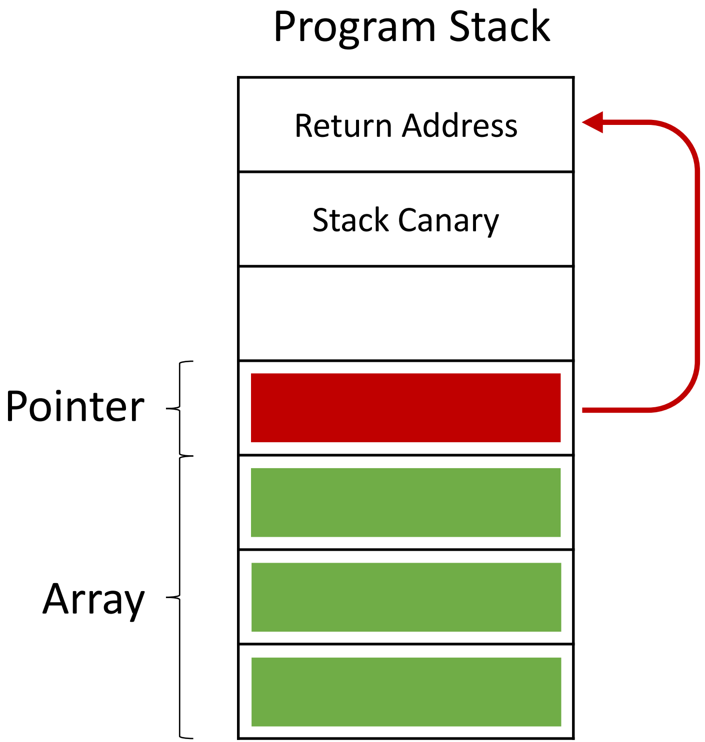
Control-Flow Hijacking Illustrated



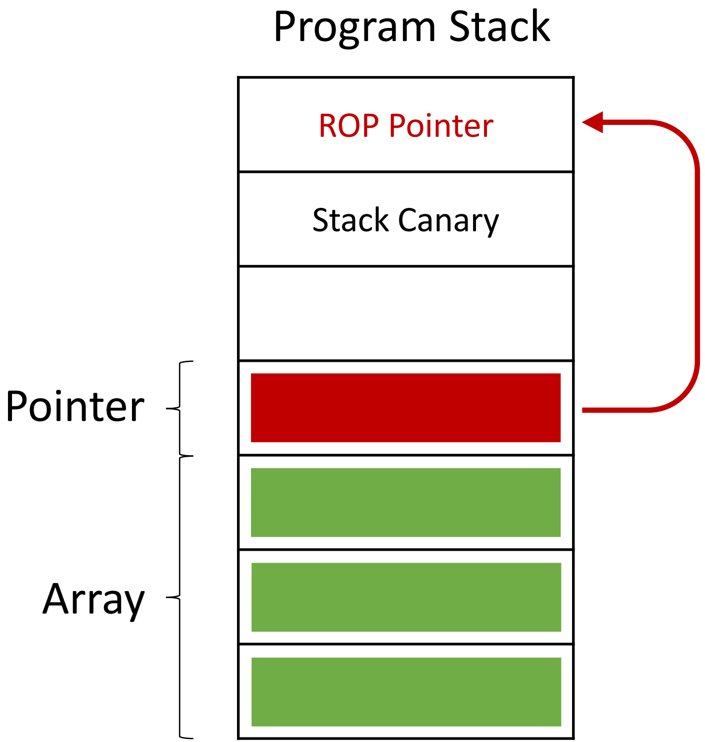
Control-Flow Hijacking Illustrated



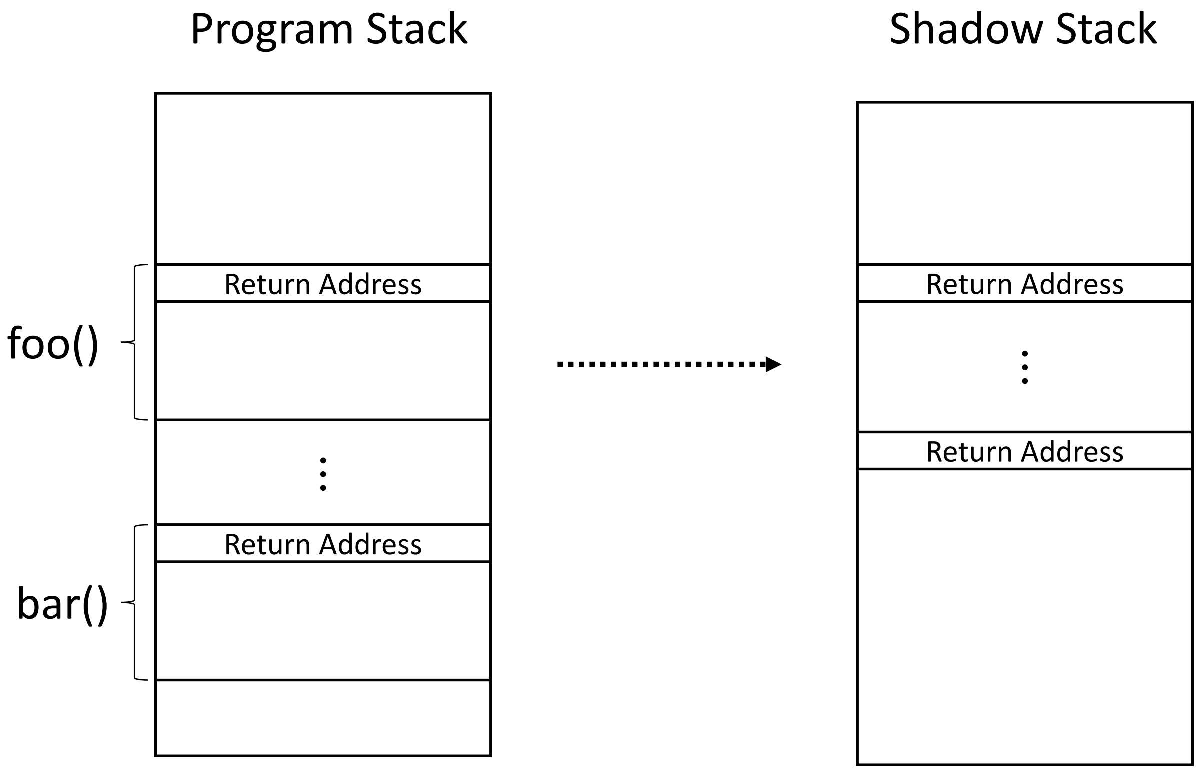
Control-Flow Hijacking Illustrated



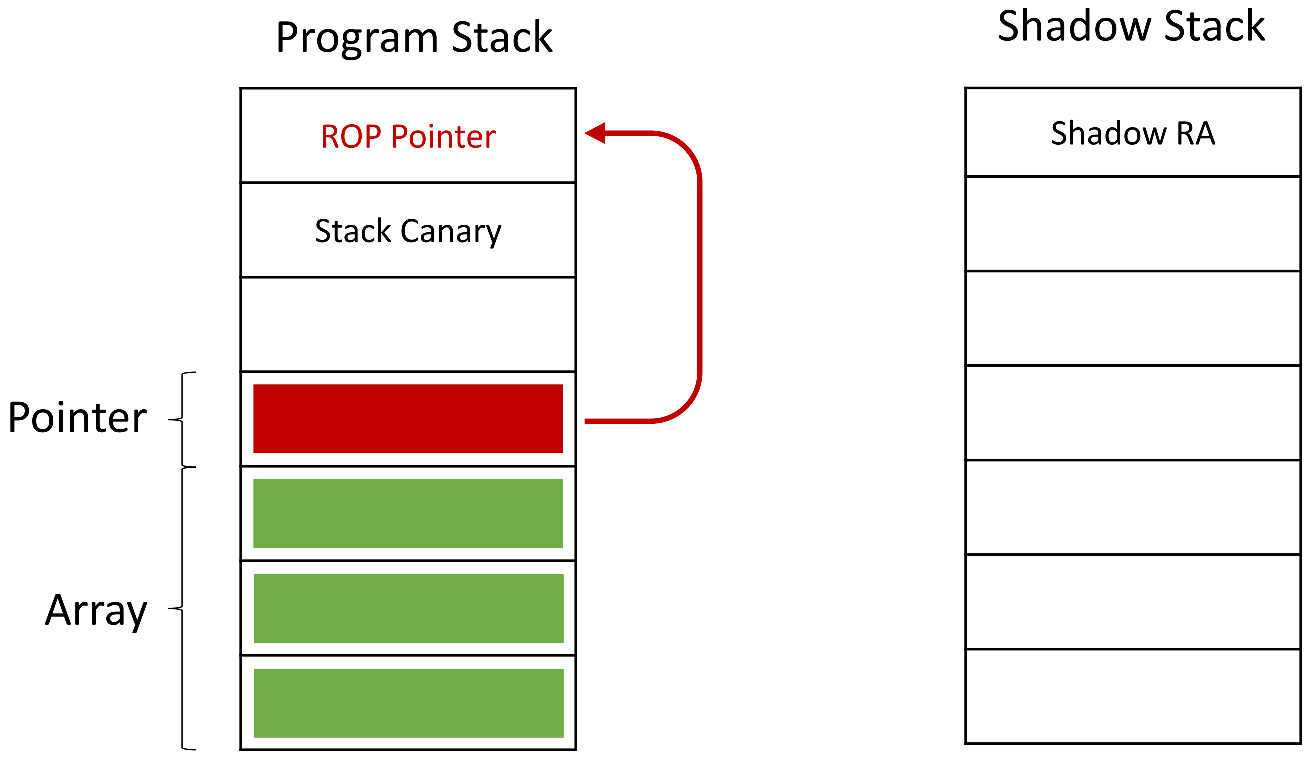
Control-Flow Hijacking Illustrated



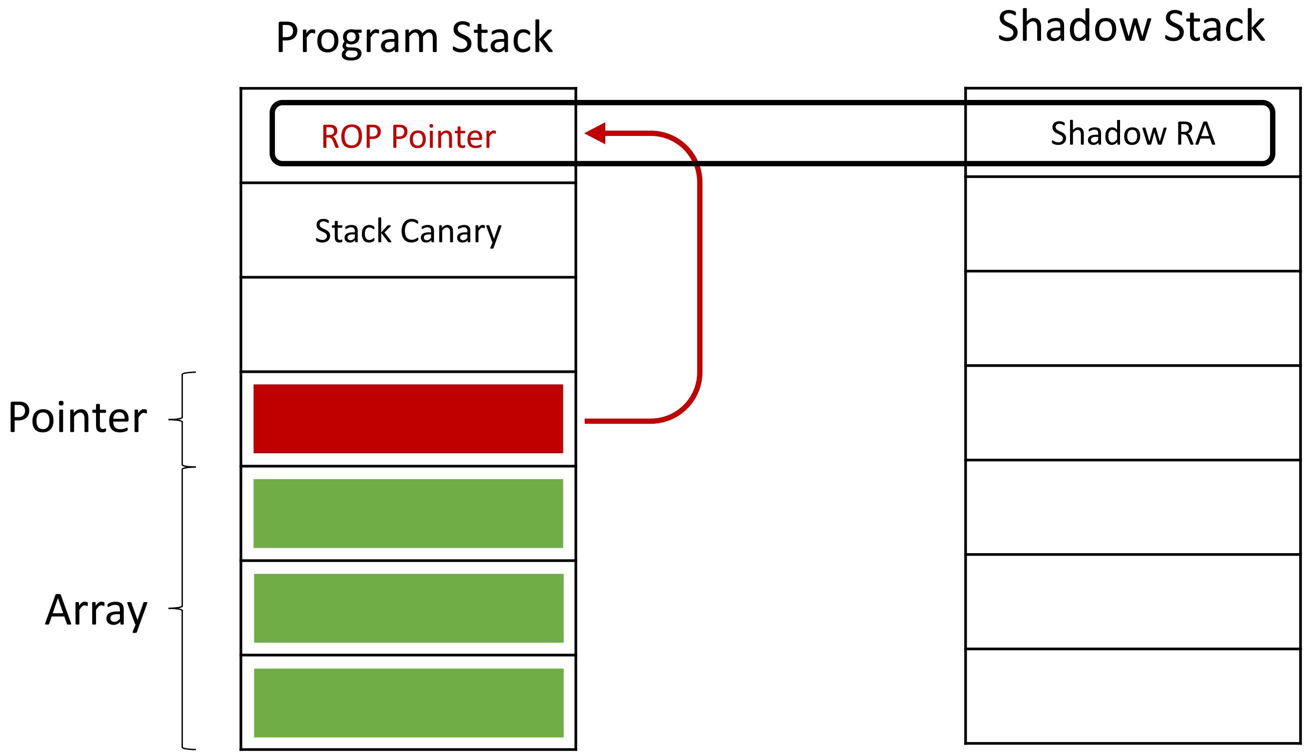
What is a Shadow Stack?



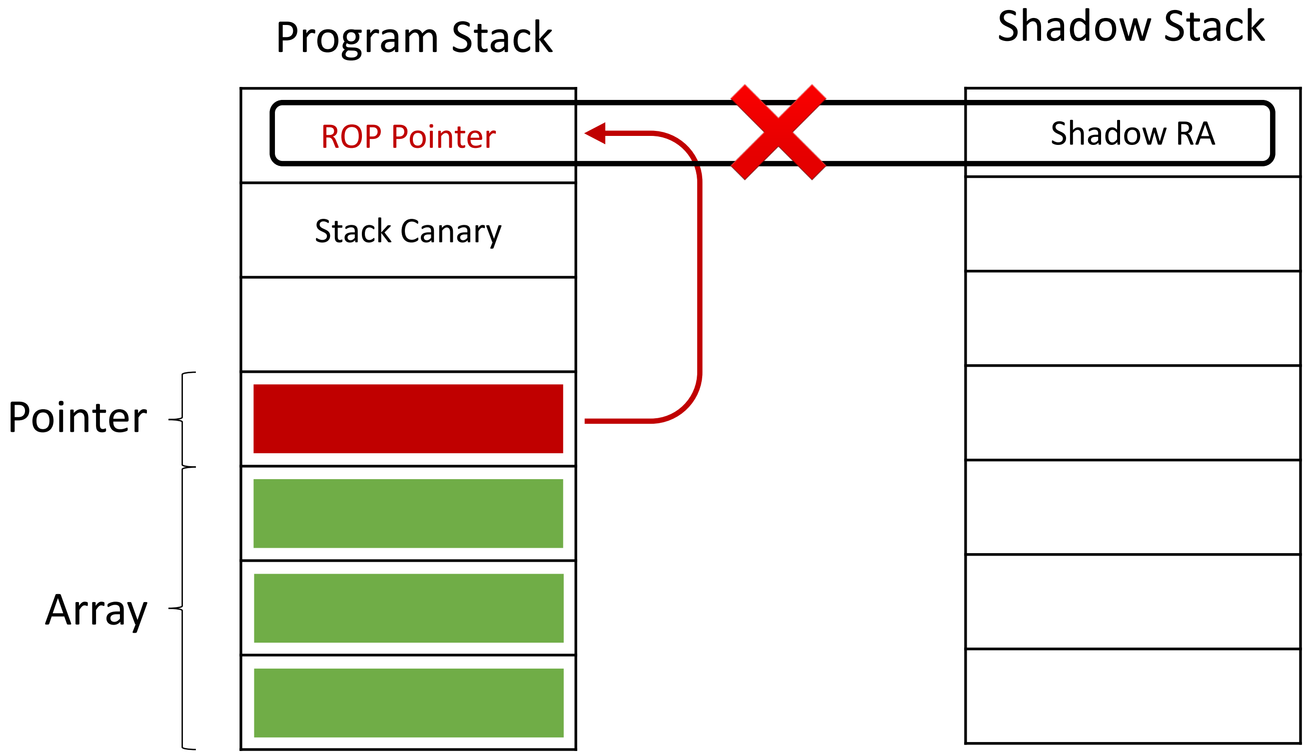
Shadow Stack Defense



Shadow Stack Defense



Shadow Stack Defense



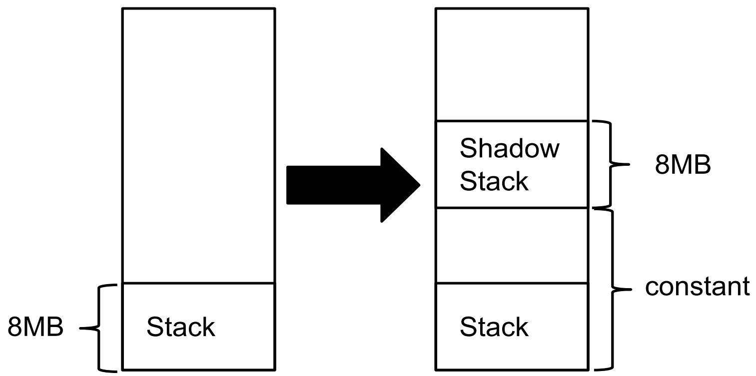
Advantages of Shadow Stacks

- Know at runtime what function you were called from
- Dynamic defense – does **NOT** rely on static analysis
- Separates code and data planes for backward edges

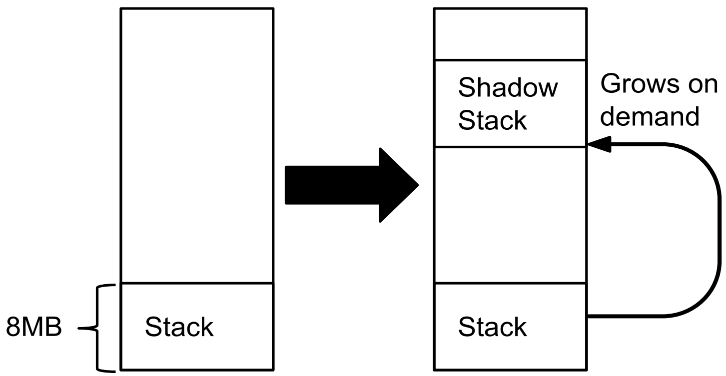
Fully precise backward edge protection!

Shadow Stack Design Space

Direct Mapping [1]



Indirect Mapping [2],[3]



[1] T. H. Dang, P. Maniatis, and D. Wagner, "The performance cost of shadow stacks and stack canaries," in AsiaCCS '15
[2] T.-c. Chiueh and F.-H. Hsu, "Rad: A compile-time solution to buffer overflow attacks," in ICDCS '01
[3] L. Davi, A.-R. Sadeghi, and M. Winandy, "Ropdefender: A detection tool to defend against return-oriented programming attacks," in AsiaCCS'11

Recommended Shadow Stack

- Indirect mapping
- Use a general purpose register for shadow stack pointer

Optimal performance and high compatibility!

Recommended Mapping

- Indirect Mapping
- As performant as direct mapping
- Minimizes memory overhead

Fastest mapping has lowest memory overhead!

Recommended Encoding

- Use general purpose (GP) register for shadow stack pointer
- Does not increase register pressure
- Significant optimization for shadow stacks

Dedicating a register to the shadow stack pointer is an effective optimization!

Compatibility of Recommended Shadow Stack

- Threading: fully supported. GP registers are thread local
- Stack Unwinding: provide instrumented setjmp / longjmp
- Unprotected Code: save and restore shadow stack pointer

Support all applications and incremental deployment!

Intra-Process Memory Isolation

- Shadow Stack splits code and data planes
- Enables integrity enforcement by isolating return addresses

Shadow Stacks enable code pointer integrity for return addresses!

Intra-Process Memory Isolation

- Software based randomization defense are defeasible
- Intel MPX uses bounds checks for isolation, moderate performance
- Intel MPK changes permissions of pages, slow performance

None of these are fully satisfactory. Tagged architectures are a promising new approach.

SPEC CPU2006 Performance Evaluation

Shadow Stack	Geometric Mean	Max	Min
Direct	5.78%	38.68%	0.00%
Recommended	3.65%	9.70%	0.00%

SPEC CPU2006 Performance Evaluation

Shadow Stack	Geometric Mean	Max	Min
Direct	5.78%	38.68%	0.00%
Recommended	3.65%	9.70%	0.00%

SPEC CPU2006 – Integrity Enforcement

Integrity Scheme	**Geometric Mean**	**Max**	**Min**
Randomization	4.31%	13.68%	0.00%
MPX	12.12%	33.02%	2.47%
MPK	61.18%	419.81%	0.00%

Conclusion

- Stack remains vulnerable to code reuse attacks
- Need to separate return addresses from data plane
- Recommend a compact, register based shadow stack for deployment

Shadow Stacks + CFI = practical CFH mitigation

https://github.com/HexHive/ShadowStack