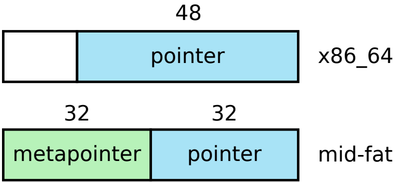
Fast and Generic Metadata Management with Mid-Fat Pointers

Taddeüs Kroes* Koen Koning*
Cristiano Giuffrida Herbert Bos
Erik van der Kouwe

April 23, 2017

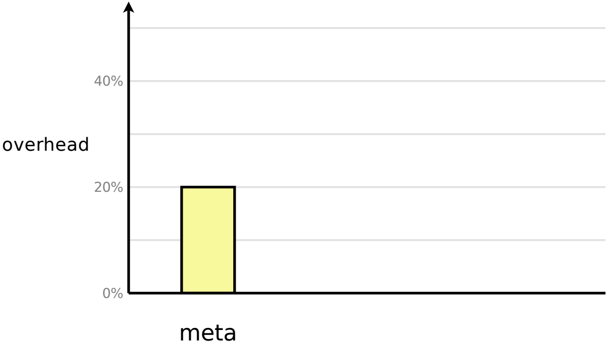
VRIJE UNIVERSITEIT AMSTERDAM

Summary

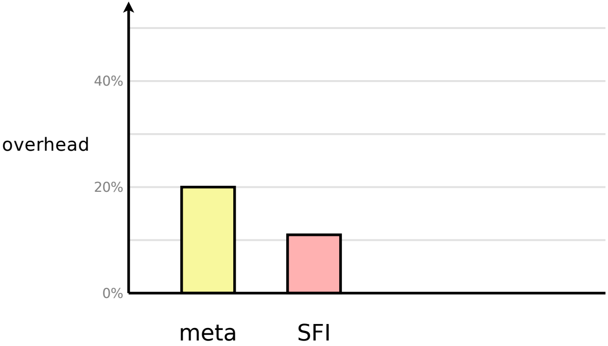


- Cache metapointer in upper bits
- Reduce metadata lookup cost
- 45% ⇒ 20% overhead on load+store lookups

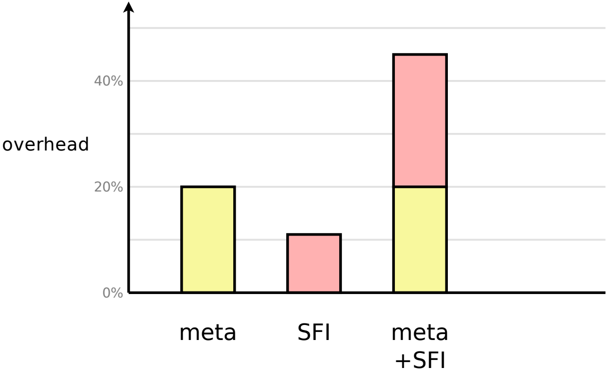
Motivation



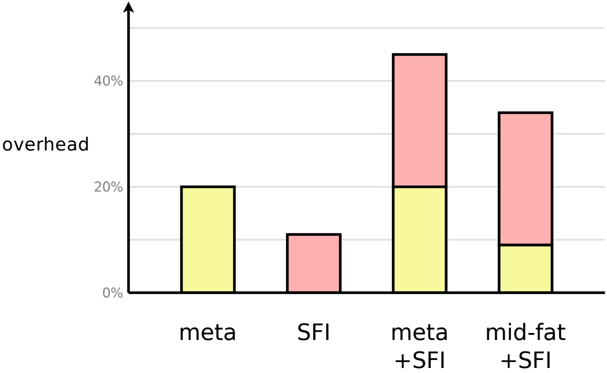
Motivation



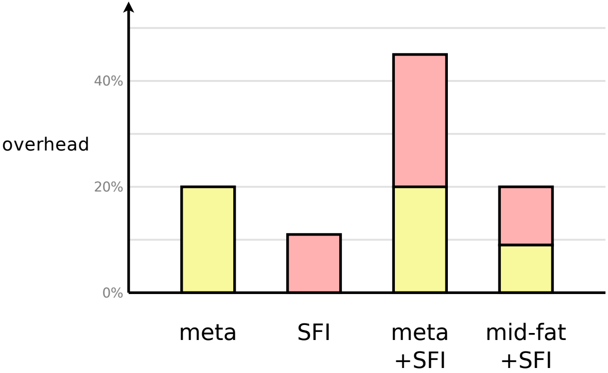
Motivation



Motivation



Motivation

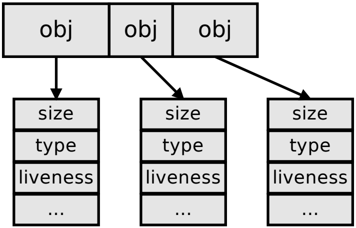


Per-object metadata

- Basis for memory safety defenses
- Associate metadata to each allocated object
- Look up metadata when pointer is used
  - Bounds checking
  - Type confusion detection
  - Use-after-free detection
  - ...

Per-object metadata

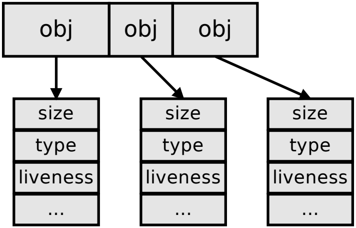
- Basis for memory safety defenses
- Associate metadata to each allocated object



- Look up metadata when pointer is used
  - Bounds checking
  - Type confusion detection
  - Use-after-free detection
  - ...

Per-object metadata

- Basis for memory safety defenses
- Associate metadata to each allocated object



- Look up metadata when pointer is used
  - Bounds checking
  - Type confusion detection
  - Use-after-free detection
  - ...

Software Fault Isolation (SFI)

- Protect sensitive program data
- E.g., metadata

- Mask dereferenced pointer

- *Key insight:* we can encode information in unused pointer bits
  - Decoding is free!

Software Fault Isolation (SFI)

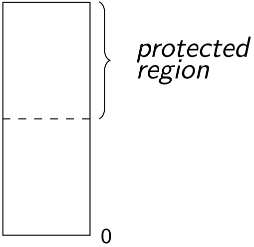
- Protect sensitive program data
- E.g., metadata

- Mask dereferenced pointer

- *Key insight:* we can encode information in unused pointer bits
  - Decoding is free!

Software Fault Isolation (SFI)

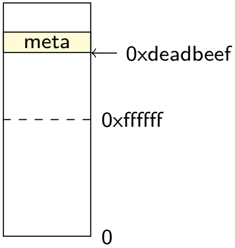
- Protect sensitive program data
- E.g., metadata
- Mask dereferenced pointer



- *Key insight:* we can encode information in unused pointer bits
  - Decoding is free!

Software Fault Isolation (SFI)

- Protect sensitive program data
- E.g., metadata
- Mask dereferenced pointer

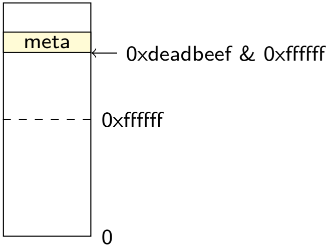


- *Key insight:* we can encode information in unused pointer bits
  - Decoding is free!

Software Fault Isolation (SFI)

- Protect sensitive program data
- E.g., metadata

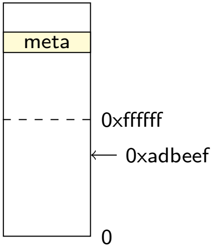
- Mask dereferenced pointer



- *Key insight:* we can encode information in unused pointer bits
  - Decoding is free!

Software Fault Isolation (SFI)

- Protect sensitive program data
- E.g., metadata

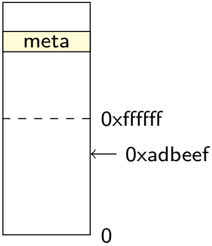


- Mask dereferenced pointer

- *Key insight:* we can encode information in unused pointer bits
  - Decoding is free!

Software Fault Isolation (SFI)

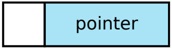
- Protect sensitive program data
- E.g., metadata
- Mask dereferenced pointer



- *Key insight:* we can encode information in unused pointer bits
  - Decoding is free!

Tagged pointers

Regular pointer



- 48-bit address space

Fat pointers

- increase pointer size
- Generic, but slow and incompatible

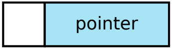
Low-fat pointers

- Small metadata tag in pointer
- Modify memory layout
- Fast and compatible, but not generic

Our approach: *Mid-fat pointers*

Tagged pointers

Regular pointer



- 48-bit address space

Fat pointers

- increase pointer size
- Generic, but slow and incompatible

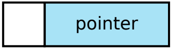
Low-fat pointers

- Small metadata tag in pointer
- Modify memory layout
- Fast and compatible, but not generic

Our approach: *Mid-fat pointers*

Tagged pointers

Regular pointer

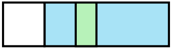


- 48-bit address space

Fat pointers

- increase pointer size
- Generic, but slow and incompatible

Low-fat pointers

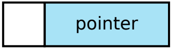


- Small metadata tag in pointer
- Modify memory layout
- Fast and compatible, but not generic

Our approach: *Mid-fat pointers*

Tagged pointers

Regular pointer

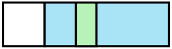


- 48-bit address space

Fat pointers

- increase pointer size
- Generic, but slow and incompatible

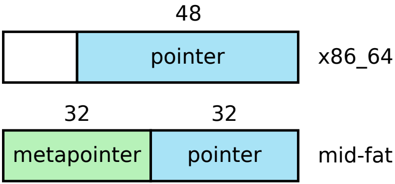
Low-fat pointers



- Small metadata tag in pointer
- Modify memory layout
- Fast and compatible, but not generic

Our approach: *Mid-fat pointers*

Mid-fat pointers



- Cache metapointer in unused pointer bits
  - Reduce address space
- Fast and generic, but still compatible
- For SFI-backed defenses

Compatibility

- Libraries
  - Cannot dereference tagged pointer
  - Need to mask at callsites

```
strdup(ptr)
⇓
strdup(MASK(ptr))
```

- Pointer comparisons

```
p1 == p2
⇓
MASK(p1) == MASK(p2)
```

- Unions

```
union {
  char buf[8];
  uint64_t flags;
}
```

- Pointers from memory

```
uint64_t *p;
char *c = (char*)(*p);
```

- Can be solved with static analysis

Compatibility

- Libraries
  - Cannot dereference tagged pointer
  - Need to mask at callsites

```
strdup(ptr)
⇓
strdup(MASK(ptr))
```

- Pointer comparisons

```
p1 == p2
⇓
MASK(p1) == MASK(p2)
```

- Unions

```
union {
  char buf[8];
  uint64_t flags;
}
```

- Pointers from memory

```
uint64_t *p;
char *c = (char*)(*p);
```

- Can be solved with static analysis

Compatibility

- Libraries
  - Cannot dereference tagged pointer
  - Need to mask at callsites

```
strdup(ptr)
⇓
strdup(MASK(ptr))
```

- Pointer comparisons

```
p1 == p2
⇓
MASK(p1) == MASK(p2)
```

- Unions

```
union {
  char buf[8];
  uint64_t flags;
}
```

- Pointers from memory

```
uint64_t *p;
char *c = (char*)(*p);
```

- Can be solved with static analysis

Compatibility

- Libraries
  - Cannot dereference tagged pointer
  - Need to mask at callsites

```
strdup(ptr)
⇓
strdup(MASK(ptr))
```

- Pointer comparisons

```
p1 == p2
⇓
MASK(p1) == MASK(p2)
```

- Unions

```
union {
  char buf[8];
  uint64_t flags;
}
```

- Pointers from memory

```
uint64_t *p;
char *c = (char*)(*p);
```

- Can be solved with static analysis

Compatibility

- Libraries
  - Cannot dereference tagged pointer
  - Need to mask at callsites

```
strdup(ptr)
⇓
strdup(MASK(ptr))
```

- Pointer comparisons

```
p1 == p2
⇓
MASK(p1) == MASK(p2)
```

- Unions

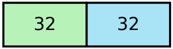
```
union {
  char buf[8];
  uint64_t flags;
}
```

- Pointers from memory

```
uint64_t *p;
char *c = (char*)(*p);
```

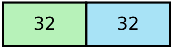
- Can be solved with static analysis

Implementation

- Track metadata for C/C++ heap
- 32-bit address space 

- LLVM for instrumentation
- METAlloc for metadata management
- Shrink address space from user space

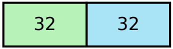
Implementation

- Track metadata for C/C++ heap
- 32-bit address space 

- LLVM for instrumentation
- METAlloc for metadata management

- Shrink address space from user space

Implementation

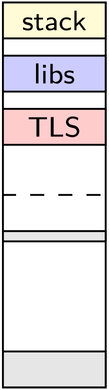
- Track metadata for C/C++ heap
- 32-bit address space 

- LLVM for instrumentation
- METAlloc for metadata management

- Shrink address space from user space

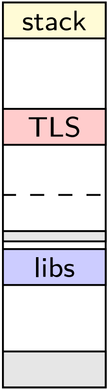
Shrinking the address space

1. Prelink shared libraries
2. Move stack and TLS
3. Reserve upper virtual memory area



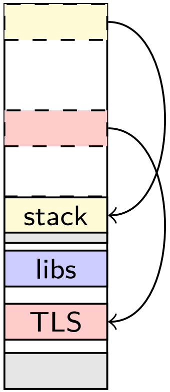
Shrinking the address space

1. Prelink shared libraries
2. Move stack and TLS
3. Reserve upper virtual memory area



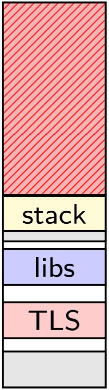
Shrinking the address space

1. Prelink shared libraries
2. Move stack and TLS
3. Reserve upper virtual memory area



Shrinking the address space

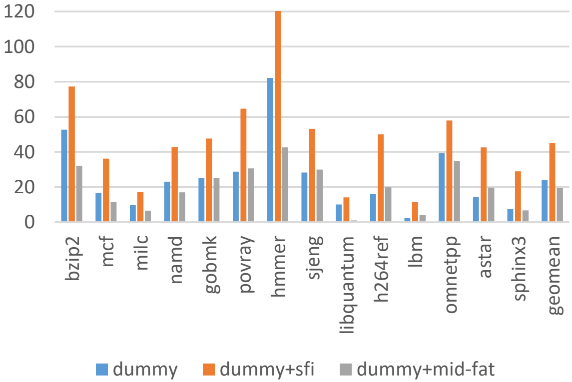
1. Prelink shared libraries
2. Move stack and TLS
3. Reserve upper virtual memory area



Experiment

- Representative “dummy” defense
- Expected overhead between METAlloc and METAlloc+SFI

Runtime overhead



Limitations

- Only 4GB memory
- Reduces entropy for ASLR
- Conflicts with custom pointer tagging
- Implementation does not protect metadata memory

Conclusion

- We piggy-back on SFI to cache metapointers as pointer tags
- Performance is practical
    - Can even reduce overhead!
- Source will be at `https://github.com/vusec/midfat`

VRIJE UNIVERSITEIT AMSTERDAM