

STKTOKENS: Enforcing Well-Bracketed Control Flow and Stack Encapsulation Using Linear Capabilities

Lau Skorstengaard¹ Dominique Devriese² Lars Birkedal¹

¹Aarhus University

²Vrije Universiteit Brussel

POPL, January 19, 2019

Overview

STKTOKENS-paper in the big picture

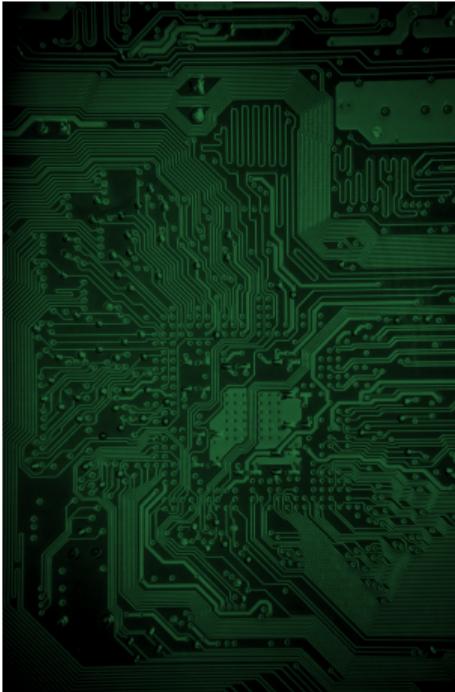
Defining well-bracketed control flow and local state encapsulation

Overview

STKTOKENS-paper in the big picture

Defining well-bracketed control flow and local state encapsulation

Abstractions all the way down



Abstractions all the way down

```
main:  
    .cfi_startproc  
# BB#0:  
    pushq %rbp  
.Ltmp0:  
    .cfi_offset %rbp, -16  
.Ltmp1:  
    .cfi_offset %rbp, -16  
    movq %rsp, %rbp  
.Ltmp2:  
    .cfi_offset %rbp, -16  
    subq $16, %rsp  
    movabsq $.L.str, %rdi  
    movl $0, -4(%rbp)  
    movb $0, %al  
    callq printf  
    xorl %ecx, %ecx  
    movl %eax, -8(%rbp)  
    movl %ecx, %eax  
    addq $16, %rsp  
    popq %rbp  
    retq  
.Lfunc_end0:  
    .size main, .Lfunc_end0-main  
    .cfi_endproc
```



Abstractions all the way down

```
#include <stdio.h>
int main()
{
    printf("Hello, World!");
    return 0;
}

main:
.cfi_startproc
# BB#0:
    pushq %rbp
.Ltmp0:
    .cfi_offset %rbp, -16
    movq %rsp, %rbp
.Ltmp1:
    .cfi_offset %rbp, -16
    movq %rbp, %rsp
.Ltmp2:
    .cfi_offset %rbp, -16
    subq $16, %rsp
    movabsq $.L.str, %rdi
    movl $0, -4(%rbp)
    movb $0, %al
    callq printf
    xorl %ecx, %ecx
    movl %eax, -8(%rbp)
    movl %ecx, %eax
    addq $16, %rsp
    popq %rbp
    retq
.Lfunc_end0:
.size main, .Lfunc_end0-main
.cfi_endproc
```



Abstractions all the way down

compilation

```
#include <stdio.h>
int main()
{
    printf("Hello, World!");
    return 0;
}
```

```
main:
    .cfi_startproc
# BB#0:
    pushq %rbp
.Ltmp0:
    .cfi_offset %rbp, -16
.Ltmp1:
    .cfi_offset %rbp, -16
    movq %rsp, %rbp
.Ltmp2:
    .cfi_offset %rbp, -16
    subq $16, %rsp
    movabsq $.L.str, %rdi
    movl $0, -4(%rbp)
    movb $0, %al
    callq printf
    xorl %ecx, %ecx
    movl %eax, -8(%rbp)
    movl %ecx, %eax
    addq $16, %rsp
    popq %rbp
    retq
.Lfunc_end0:
    .size main, .Lfunc_end0-main
    .cfi_endproc
```



Abstractions all the way down

secure
compilation

```
#include <stdio.h>
int main()
{
    printf("Hello, World!");
    return 0;
}
```

```
main:
    .cfi_startproc
# BB#0:
    pushq %rbp
.Ltmp0:
    .cfi_offset %rbp, -16
.Ltmp1:
    .cfi_offset %rbp, -16
    movq %rsp, %rbp
.Ltmp2:
    .cfi_offset %rbp, -16
    subq $16, %rsp
    movabsq $.L.str, %rdi
    movl $0, -4(%rbp)
    movb $0, %al
    callq printf
    xorl %ecx, %ecx
    movl %eax, -8(%rbp)
    movl %ecx, %eax
    addq $16, %rsp
    popq %rbp
    retq
.Lfunc_end0:
    .size main, .Lfunc_end0-main
    .cfi_endproc
```

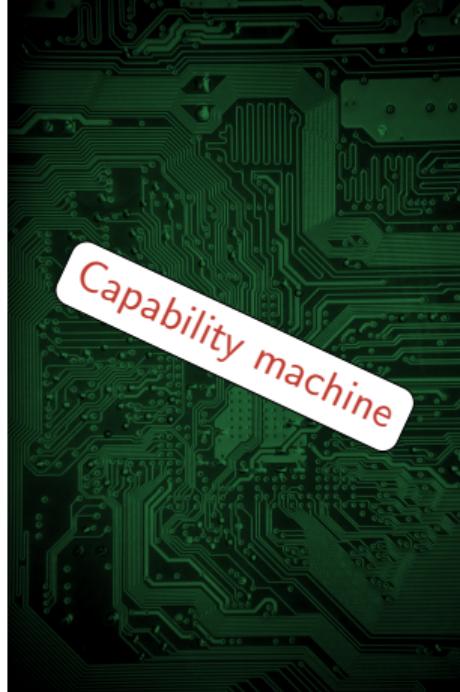


Abstractions all the way down

secure
compilation

```
#include <stdio.h>
int main()
{
    printf("Hello, World!");
    return 0;
}
```

```
main:
    .cfi_startproc
# BB#0:
    pushq %rbp
.Ltmp0:
    .cfi_offset %rbp, -16
    movq %rsp, %rbp
.Ltmp1:
    .cfi_offset %rbp, -16
    movq %rbp, %rsp
    movabsq $.L.str, %rdi
    movl $0, -4(%rbp)
    movb $0, %al
    callq printf
    xorl %ecx, %ecx
    movl %eax, -8(%rbp)
    movl %ecx, %eax
    addq $16, %rsp
    popq %rbp
    retq
.Lfunc_end0:
    .size main, .Lfunc_end0-main
    .cfi_endproc
```



Capability machine

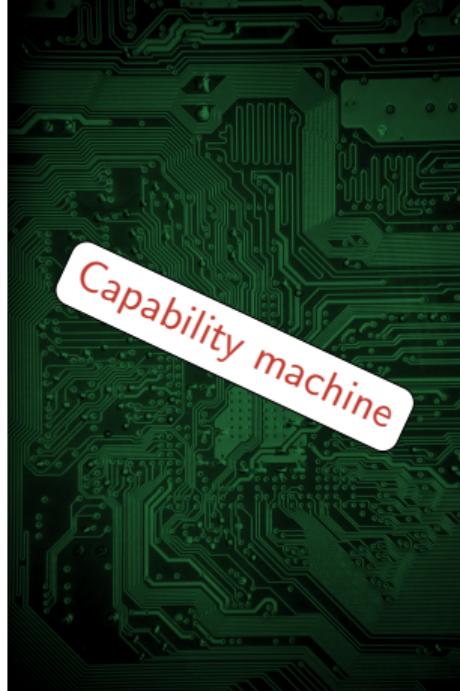
Abstractions all the way down

secure compilation

ir → ir'

```
#include <stdio.h>
int main()
{
    printf("Hello, World!");
    return 0;
}
```

```
main:
.cfi_startproc
# BB#0:
    pushq %rbp
.Ltmp0:
    .cfi_offset %rbp, -16
.Ltmp1:
    .cfi_offset %rbp, -16
    movq %rsp, %rbp
.Ltmp2:
    .cfi_offset %rbp, -16
    subq $16, %rsp
    movabsq $.L.str, %rdi
    movl $0, -4(%rbp)
    movb $0, %al
    callq printf
    xorl %ecx, %ecx
    movl %eax, -8(%rbp)
    movl %ecx, %eax
    addq $16, %rsp
    popq %rbp
    retq
.Lfunc_end0:
.size main, .Lfunc_end0-main
.cfi_endproc
```



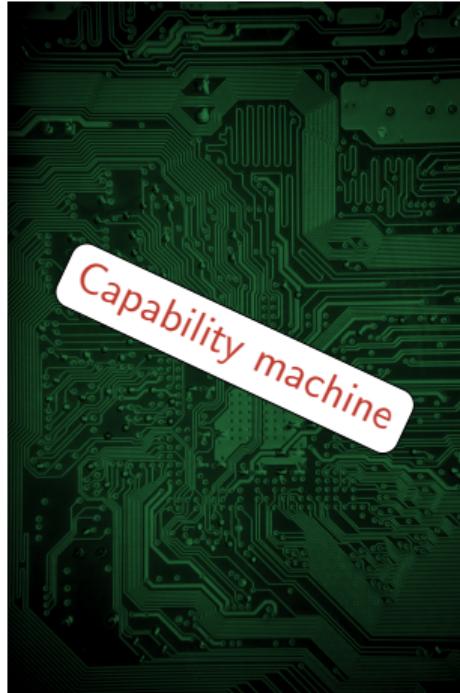
Abstractions all the way down

secure compilation

Diagram showing the flow from source code to intermediate representation (ir) and then to modified intermediate representation (ir'). A red box highlights the transition from ir to ir'.

```
#include <stdio.h>
int main()
{
    printf("Hello, World!");
    return 0;
}
```

```
main:
.cfi_startproc
# BB#0:
pushq %rbp
.Ltmp0:
.cfi_offset %rbp, -16
.Ltmp1:
.cfi_offset %rbp, -16
movq %rsp, %rbp
.Ltmp2:
.cfi_offset %rbp, -16
subq $16, %rsp
movabsq $.L.str, %rdi
movl $0, -4(%rbp)
movb $0, %al
callq printf
xorl %ecx, %ecx
movl %eax, -8(%rbp)
movl %ecx, %eax
addq $16, %rsp
popq %rbp
retq
.Lfunc_end0:
.size main, .Lfunc_end0-main
.cfi_endproc
```



Paper Contents Overview

Formalization of CHERI-like capability machine with linear capabilities

`STKTOKENS` a calling convention that provably guarantees local-state encapsulation (LSE) and well-bracketed control-flow (WBCF)

Fully-abstract overlay semantics a novel way to prove LSE and WBCF claims

Overview

STKTOKENS-paper in the big picture

Defining well-bracketed control flow and local state encapsulation

Defining well-bracketed control flow and local state encapsulation

```
void a()
{
    int y = 40+2;
    return;
}
```

```
void b()
{
    int x = 5;
    a();
    x = 2;
    a();
    return;
}
```

Defining well-bracketed control flow and local state encapsulation

```
void a()  
{  
    int y = 40+2;  
    return;  
}
```

Function a cannot
access variable x

```
void b()  
{  
    int x = 5;  
    a();  
    x = 2;  
    a();  
    return;  
}
```

Local-state encapsulation

Defining well-bracketed control flow and local state encapsulation

```
void a()
{
    int y = 40+2;
    return;
}
```

```
void b()
{
    → int x = 5;
    a();
    x = 2;
    a();
    return;
}
```

Well-bracketed control flow

Desired properties of the WBCF and LSE definition

1. *Intuitive*
2. *Useful for reasoning*
3. *Reusable in secure compiler chains*
4. *Arguably "complete"*

Overlay Semantics

```
move rtmp1 42
store rstk rtmp1
cca rstk -1
geta rtmp1 rstk
cca rretc 5
move rtmp1 pc
cca rtmp1 -20
```

call

```
load rtmp1 rtmp1
cca rtmp1 -21
cseal rretd rtmp1
move rretc pc
return
cseal rretc rtmp1
move rtmp1 0
```

return

Overlay Semantics (oLCM)

```
move rtmp1 42
store rstk rtmp1
cca rstk -1
geta rtmp1 rstk
cca rretc 5
move rtmp1 pc
cca rtmp1 -20
```

```
load rtmp1 rtmp1
cca rtmp1 -21
cseal rretd rtmp1
move rretc pc
xjmp r1 r2
cseal rretc rtmp1
move rtmp1 0
```

Linear Capability Machine (LCM)

Overlay Semantics

```
move rtmp1 42
store rstk rtmp1
cca rstk -1
geta rtmp1 rstk
cca rretc 5
move rtmp1 pc
cca rtmp1 -20
```

```
load rtmp1 rtmp1
cca rtmp1 -21
cseal rretd rtmp1
move rretc pc
return
cseal rretc rtmp1
move rtmp1 0
```

Overlay Semantics
(oLCM)

```
move rtmp1 42
store rstk rtmp1
cca rstk -1
geta rtmp1 rstk
cca rretc 5
move rtmp1 pc
cca rtmp1 -20
```

```
load rtmp1 rtmp1
cca rtmp1 -21
cseal rretd rtmp1
move rretc pc
xjmp r1 r2
cseal rretc rtmp1
move rtmp1 0
```

Linear Capability
Machine (LCM)

Evaluating the definition

1. *Intuitive*
2. *Useful for reasoning*
3. *Reusable in secure compiler chains*
4. *Arguably "complete"*

Evaluating the definition

1. *Intuitive*
Yes, call stack corresponds exactly to our intuition.
2. *Useful for reasoning*
3. *Reusable in secure compiler chains*
4. *Arguably "complete"*

Evaluating the definition

1. *Intuitive*
Yes, call stack corresponds exactly to our intuition.
2. *Useful for reasoning*
Yes, ...
3. *Reusable in secure compiler chains*
4. *Arguably "complete"*

Evaluating the definition

1. *Intuitive*

Yes, call stack corresponds exactly to our intuition.

2. *Useful for reasoning*

Yes, ...

3. *Reusable in secure compiler chains*

Yes, fully-abstract compilations compose vertically, so oLCM can be used as a target for other compilation phases.

4. *Arguably "complete"*

Evaluating the definition

1. *Intuitive*

Yes, call stack corresponds exactly to our intuition.

2. *Useful for reasoning*

Yes, ...

3. *Reusable in secure compiler chains*

Yes, fully-abstract compilations compose vertically, so oLCM can be used as a target for other compilation phases.

4. *Arguably "complete"*

Yes, ...

Thank you!