

Korset: Code-based Intrusion Detection for Linux

Ohad Ben-Cohen Avishai Wool

Tel Aviv University

Table of Contents

why what how

demo!

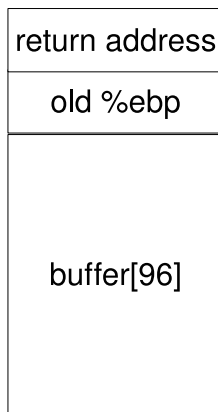
evaluate

Section 1: The Problem

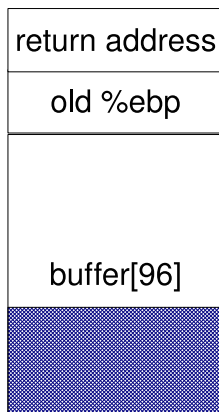
Exploit this

```
void sayhi(char *param)
{
    char buf[96];
    gets(buf);
    printf("Hi %s, please don't hurt me!\n", buf);
}
```

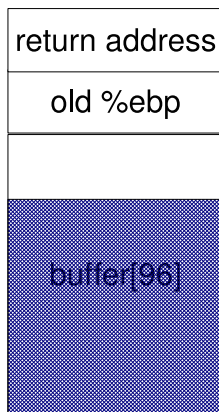
Buffer Overflow



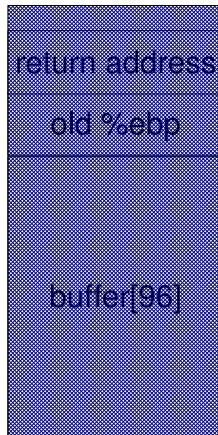
Buffer Overflow



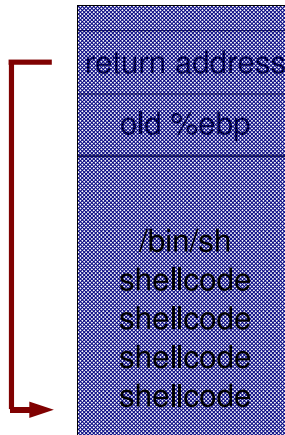
Buffer Overflow



Buffer Overflow



Buffer Overflow



Code Injection

Defense

Host-based Intrusion Detection Systems (HIDS's)

To Identify Malicious Activities

- Pre-construct a model of normal behavior
- Monitor running processes
- Compare data to model
- Alarm when deviates

Host-based Intrusion Detection Systems (HIDS's)

To Identify Malicious Activities

- Pre-construct a model of normal behavior
- Monitor running processes
- Compare data to model
- Alarm when deviates

Terms

- False Positives (\Rightarrow usability)
- False Negatives (\Rightarrow precision)

Models of normal behavior

Models of normal behavior

1. Machine Learning

- Automated
- Capable of detecting a wide range of attacks
- Statistical \Rightarrow Have False Alarms

Models of normal behavior

1. Machine Learning

- Automated
- Capable of detecting a wide range of attacks
- Statistical \Rightarrow Have False Alarms

False Alarms are inherent and inevitable

```
if(time() < YEAR2009)
    read(...);
else
    write(...);
```


Models of normal behavior

1. Machine Learning

- Automated
- Capable of detecting a wide range of attacks
- Statistical \Rightarrow Have False Alarms

False Alarms are inherent and inevitable

```
if(time() < YEAR2009)
    read(...);
else
    write(...);
```

2. Program Policies

- Can be very accurate \Rightarrow Eliminate False Alarms
- Tedious and demanding

Models of normal behavior

1. Machine Learning

- Automated
- Capable of detecting a wide range of attacks
- Statistical \Rightarrow Have False Alarms

False Alarms are inherent and inevitable

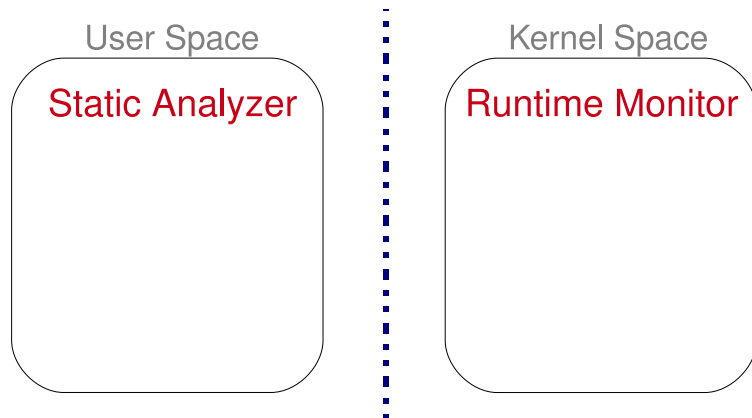
```
if(time() < YEAR2009)
    read(...);
else
    write(...);
```

2. Program Policies

- Can be very accurate \Rightarrow Eliminate False Alarms
- Tedious and demanding

Section 2: Korset

General Architecture



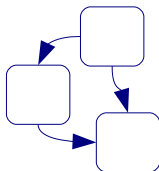
Model of Normal Behavior

Control Flow Graph (CFG)

General Architecture

User Space

Static Analyzer



Kernel Space

Runtime Monitor

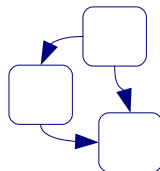
General Architecture

User Space

Static Analyzer

Kernel Space

Runtime Monitor



Stage #1: Model Preconstruction

Protect me

```
if (num < 2)
    num++;
fd = open("idata", O_RDONLY);
i = argc - 1;
if (2 == i) {
    for ( ; num < 5; num++)
        n += read(fd, buf, 50);
} else {
    n = write(fd, buf, 59);
}
n++;
close(fd);
```

Assumption:

System calls are the only way to inflict damage

(Not entirely true...)

Protect me

```
if (num < 2)
    num++;
fd = open("idata", O_RDONLY);
i = argc - 1;
if (2 == i) {
    for ( ; num < 5; num++)
        n += read(fd, buf, 50);
} else {
    n = write(fd, buf, 59);
}
n++;
close(fd);
```

Protect me

```
if (num < 2)
    num++;
fd = open("idata", O_RDONLY);
i = argc - 1;
if (2 == i) {
    for ( ; num < 5; num++)
        n += read(fd, buf, 50);
} else {
    n = write(fd, buf, 59);
}
n++;
close(fd);
```

Protect me

```
fd = open("idata", O_RDONLY);

if (2 == i) {
    for ( ; num < 5; num++)
        n += read(fd, buf, 50);
} else {
    n = write(fd, buf, 59);
}

close(fd);
```

Protect me

```
fd = open("idata", O_RDONLY);

if (2 == i) {
    for ( ; num < 5; num++)
        n += read(fd, buf, 50);
} else {
    n = write(fd, buf, 59);
}

close(fd);
```

Protect me

```
open(...);  
  
if (...) {  
    for (...)  
        read(...);  
} else {  
    write(...);  
}  
  
close(...);
```

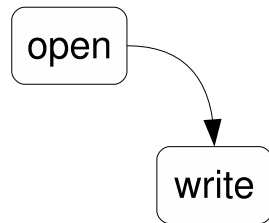
Protect me

```
open(...);  
if (...) {  
    for (...)  
        read(...);  
} else {  
    write(...);  
}  
close(...);
```

open

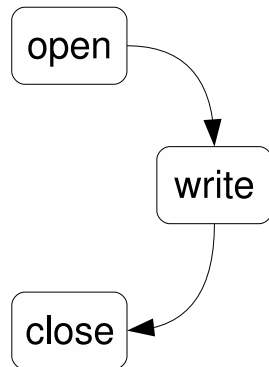
Protect me

```
open(...);  
if (...) {  
    for (...)  
        read(...);  
} else {  
    write(...);  
}  
close(...);
```



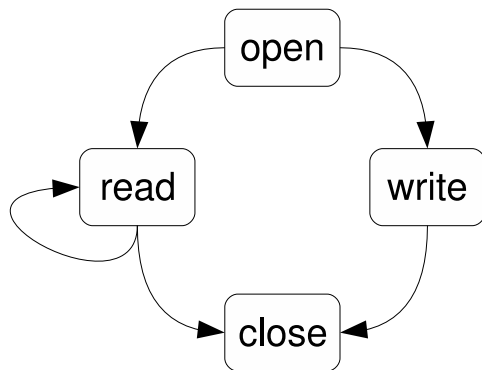
Protect me

```
open(...);  
if (...) {  
    for (...)  
        read(...);  
} else {  
    write(...);  
}  
close(...);
```



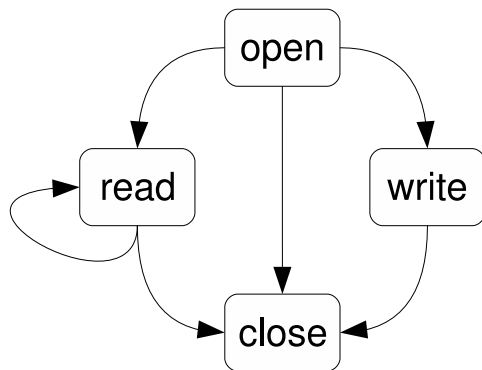
Protect me

```
open(...);  
if (...) {  
    for (...)  
        read(...);  
} else {  
    write(...);  
}  
close(...);
```

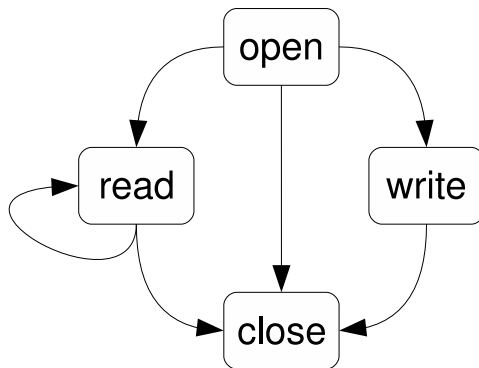


Protect me

```
open(...);  
if (...) {  
    for (...)  
        read(...);  
} else {  
    write(...);  
}  
close(...);
```

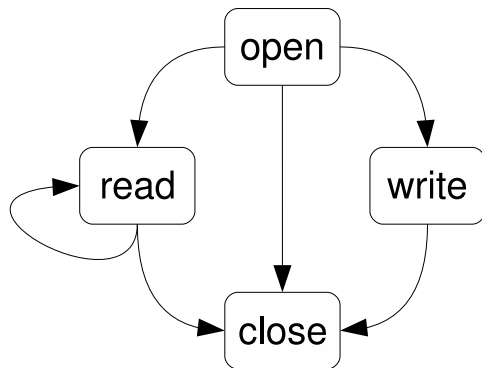


Model of Normal Behavior



System call sequences \Rightarrow Paths in the graph

Model of Normal Behavior



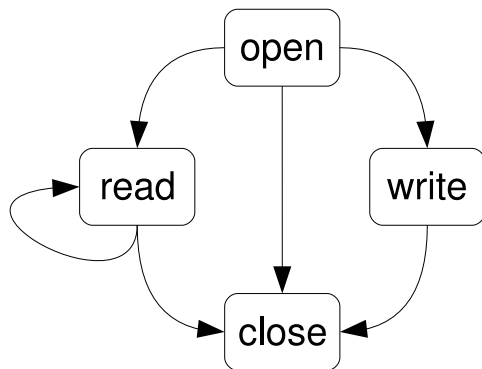
System call sequences \Rightarrow Paths in the graph

No path in the graph \Rightarrow Invalid system call sequence

Stage #2: Runtime Monitoring

Protecting

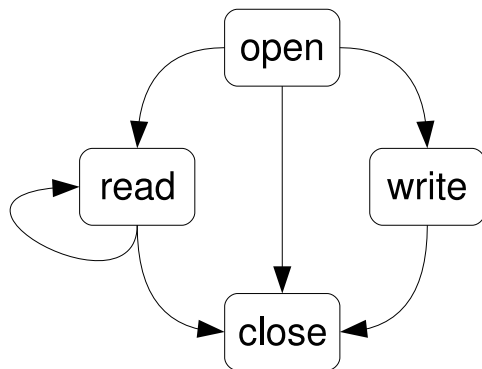
Userland



Protecting

Userland

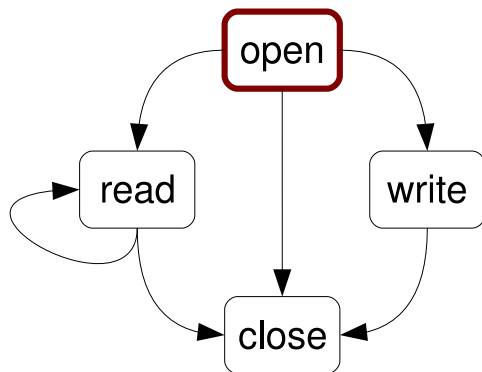
open →



Protecting

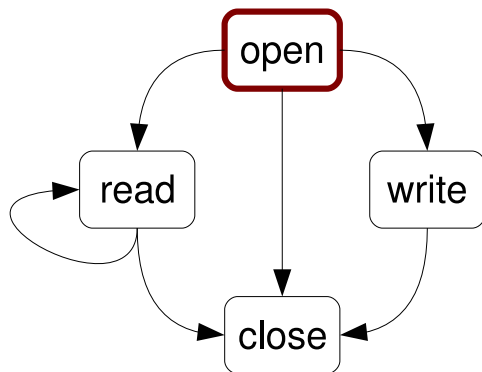
Userland

open



Protecting

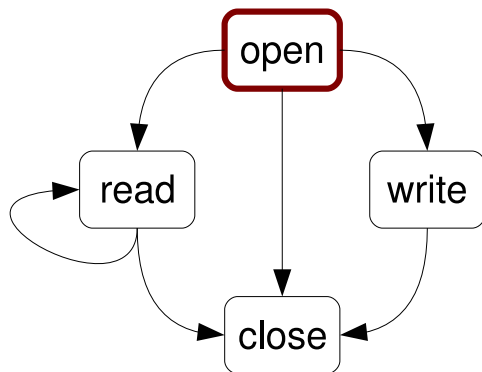
Userland



Protecting

Userland

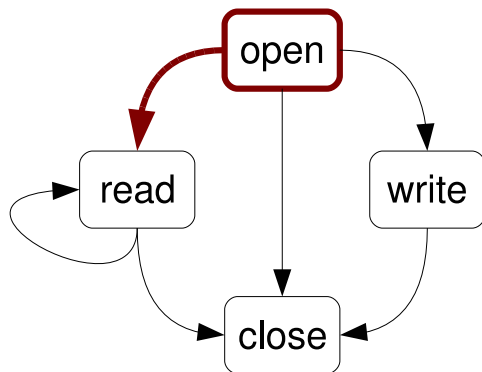
read



Protecting

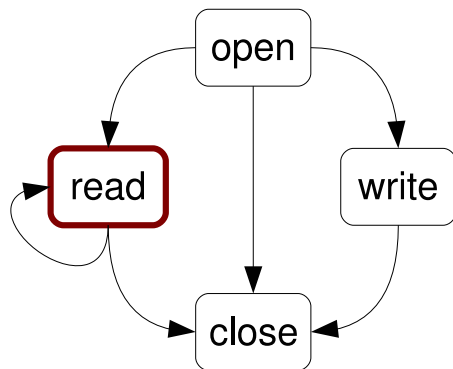
Userland

read →



Protecting

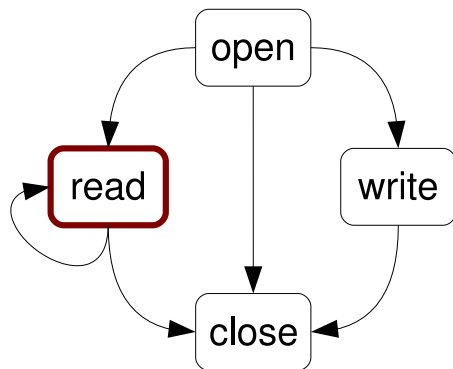
Userland



Protecting

Userland

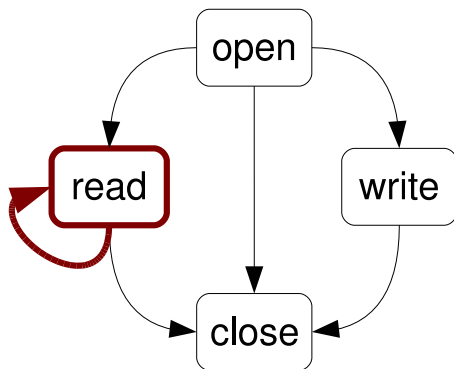
read



Protecting

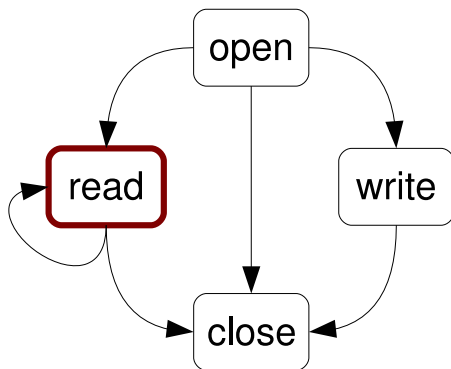
Userland

read



Protecting

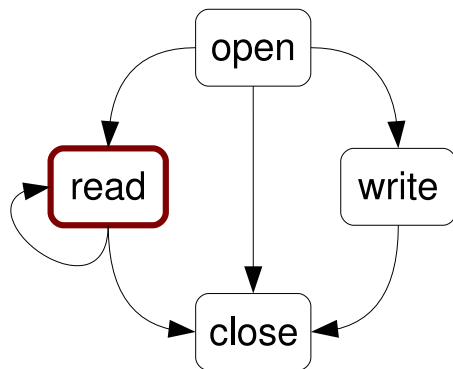
Userland



Protecting

Userland

execve →



Protecting

Kill!

In words

Model of Normal Behavior

- Control Flow Graphs (CFG)
- Only System Calls
- Statically Preconstructed
- Once for every app

Runtime Monitoring

- Monitor system calls emitted in run-time
- Simulate observed system calls on automata
- Always maintain a current node
- Terminate diverging processes

Code-based Intrusion Detection

Code-based Intrusion Detection

First work by David Wagner and Drew Dean, 2001

Intrusion Detection via Static Analysis

Pros

- Automated
- Provable zero false positives
(assuming that code isn't self modifying)

Intrusion Detection via Static Analysis

Pros

- Automated
- Provable zero false positives
(assuming that code isn't self modifying)

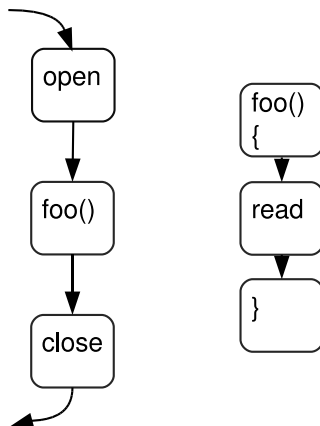
Cons

- Limited to code injection attacks
- High precision comes with a cost

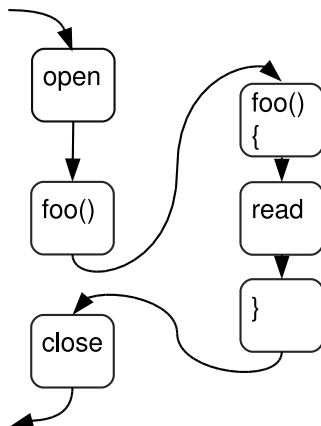
Action !

Section 3: Not so simple

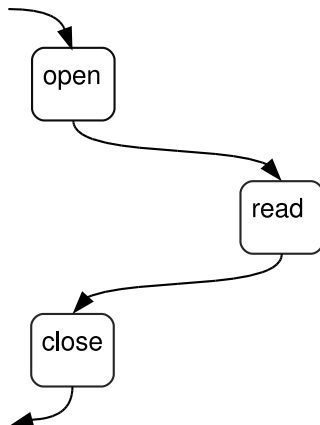
Functions



Functions - Link CFGs

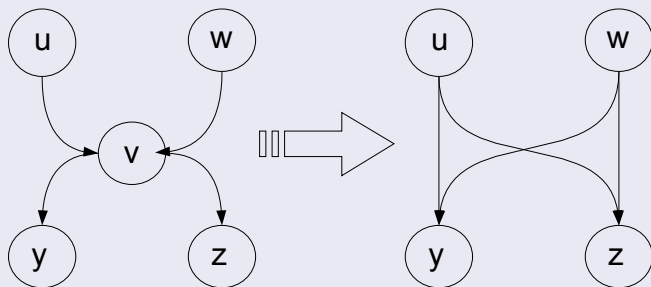


And... Simplify



Simplification Process

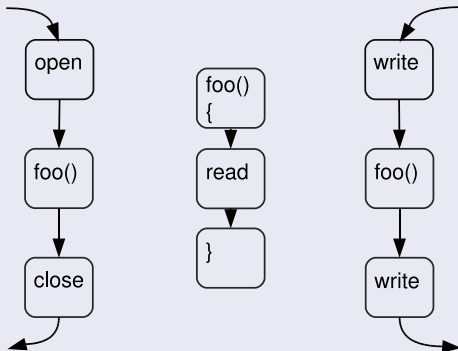
Simple and Smooth



Challenge #1

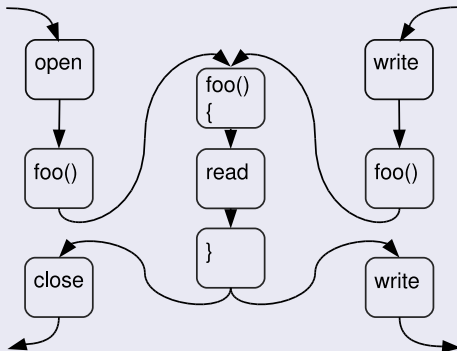
Functions Redux - Context Insensitivity

Before linking



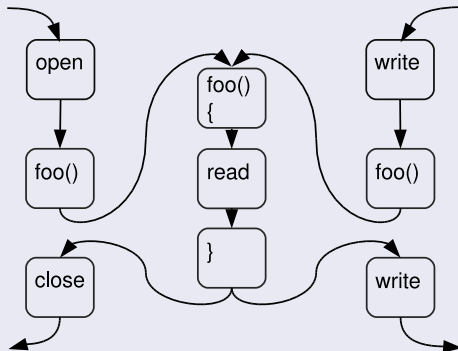
Functions Redux - Context Insensitivity

After linking



Functions Redux - Context Insensitivity

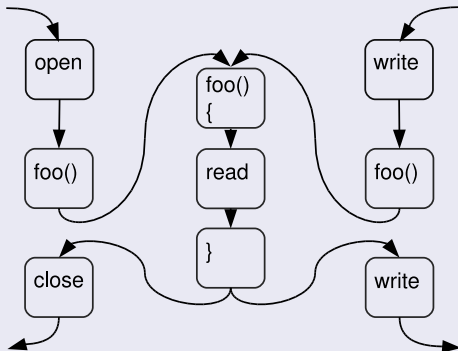
After linking



... So ?

Functions Redux - Context Insensitivity

After linking

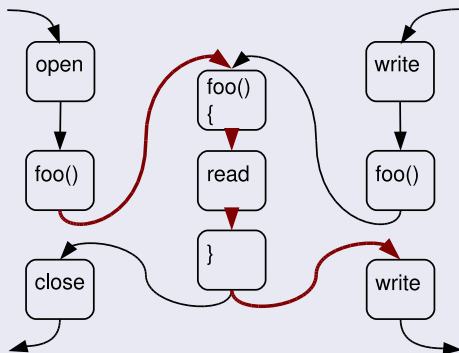


... So ?

- Impossible execution paths are allowed
- E.g.: open-read-write

Context Insensitivity

A Function after linking

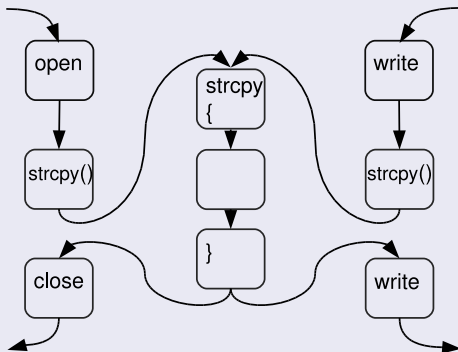


... So ?

- Impossible execution paths are allowed
- E.g.: open-read-write

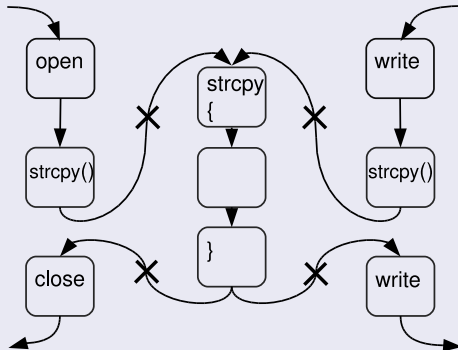
Hey before you link

Not all functions emit/lead to system calls



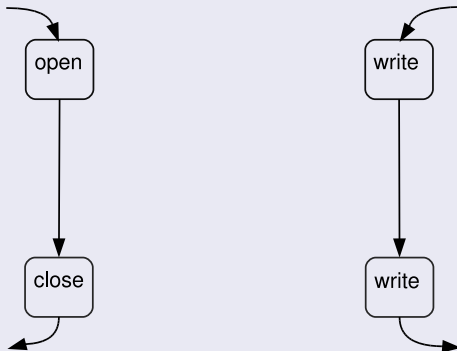
Graph Unlinking

Do not link them



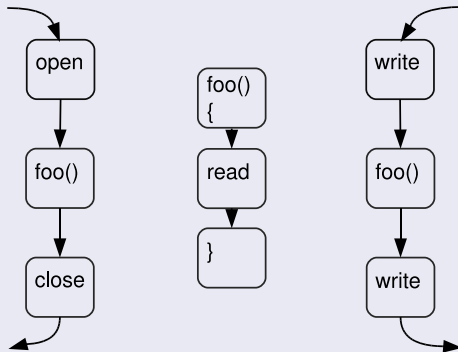
Graph Unlinking

Just ditch their calling nodes...



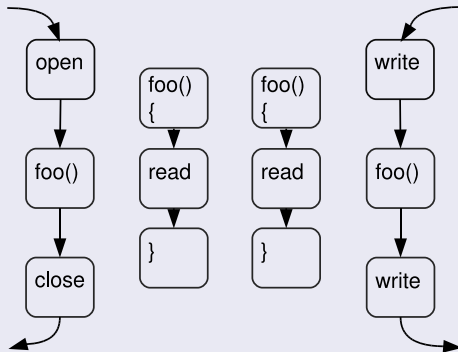
Graph Inlining

Inline CFGs of functions that issue system calls



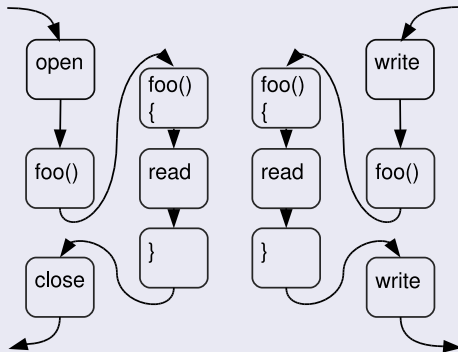
Graph Inlining

Create Private Copies



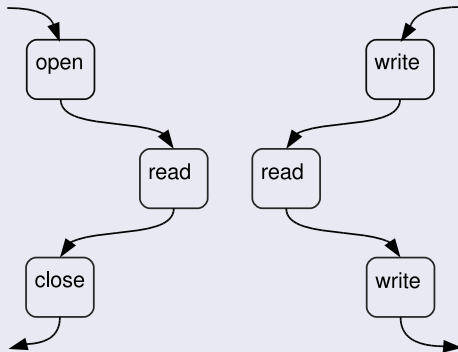
Graph Inlining

Link Private Copies



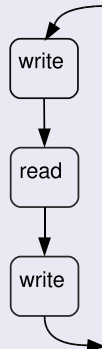
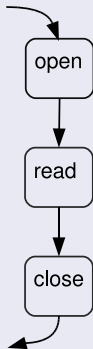
Graph Inlining

Simplify Result



Graph Inlining

After Simplifying



Graph Inlining

Inlining Depth ?

Graph Inlining

Inlining Depth ?

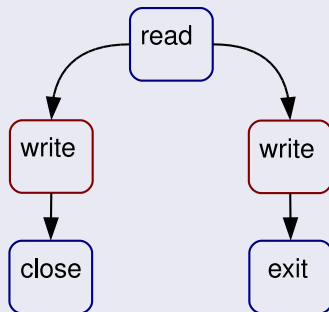
(currently - depth 1)

Challenge #2

Non Determinism

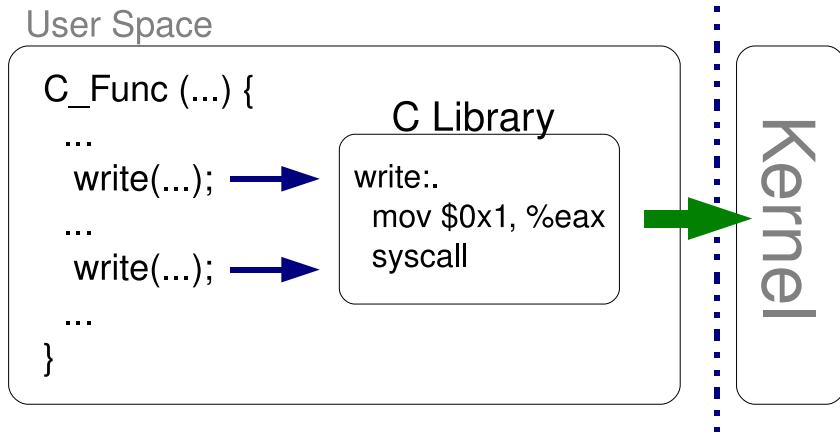
Which write is it ?

```
read(...);  
if (...) {  
    write(...);  
    close(...);  
} else {  
    write(...);  
    exit(...);  
}
```



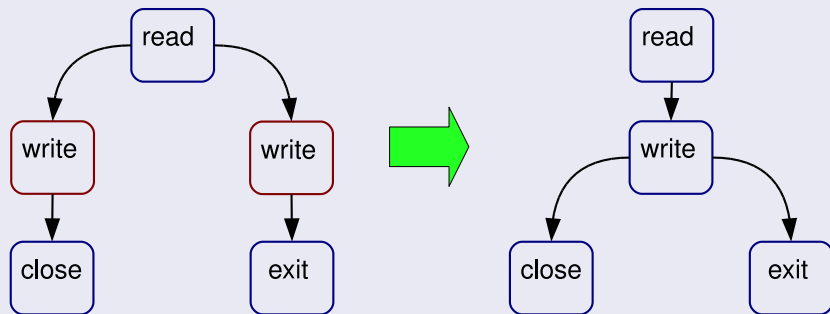
%EIP ?

%EIP does not help



Solution: Merge Nodes

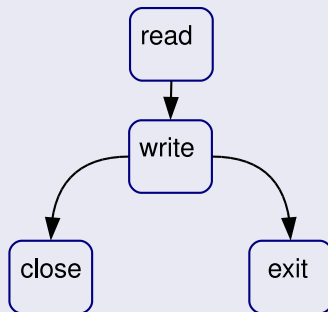
Solution: Merge nodes



Non Determinism

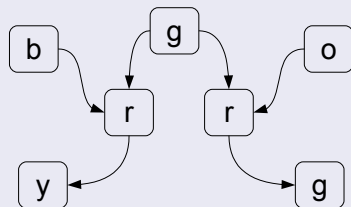
Solution: Merge nodes

```
read(...);  
if (...) {  
    write(...);  
    close(...);  
} else {  
    write(...);  
    exit(...);  
}
```

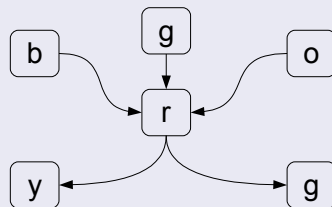


Merging cost

Graph now allows impossible paths!



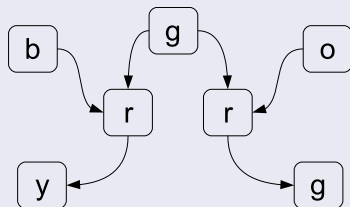
accepting: gry, grg, bry, org



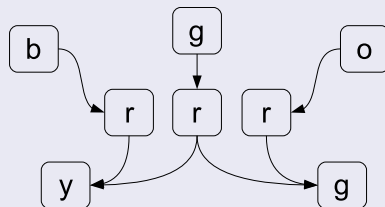
accepting: gry, grg, bry,
org, brg, ory

Minimizing Merging cost

Don't merge, add

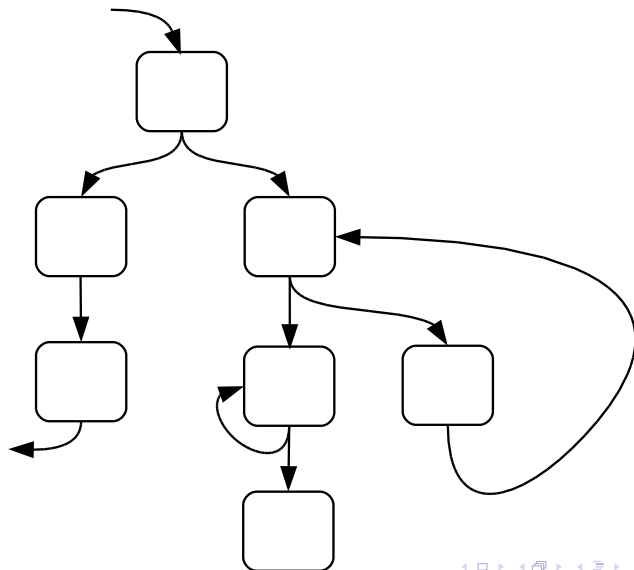


accepting: gry, grg, bry, org



accepting: gry, grg, bry, org

the Deterministic Callgraph Automaton (DCA)



the Deterministic Callgraph Automaton (DCA)

Only system call nodes

- There are no ϵ -edges
- \Rightarrow Need to check only direct descendants

No control flow ambiguity

- No more than a single match
- \Rightarrow Current state is always a single node

Complexity

- Time: $O(|\Sigma|)$ (Σ - set of system calls)
- Space: $O(1)$

Section 4: Implementation

General Architecture

User Space

Static Analyzer

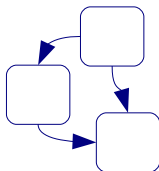
Kernel Space

Runtime Monitor

General Architecture

User Space

Static Analyzer



Kernel Space

Runtime Monitor

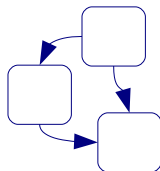
General Architecture

User Space

Static Analyzer

Kernel Space

Runtime Monitor



Kernel guts

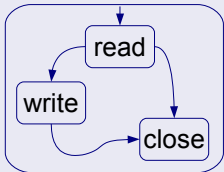
The Monitoring Agent

User Space

example

ELF executable

example.korset

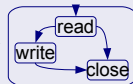


System Calls

Kernel Space

Korset
Monitoring
Agent

Kernel
System Call
Handler



Per process state

sched.h

```
struct task_struct {  
    ...  
    char *korset_graph;  
    u32 korset_node;  
    ...  
};
```

Monitoring Agent - via a new LSM hook

```
entry.S
```

```
ENTRY(system_call)

...
GET_THREAD_INFO(%rcx)
SAVE_ARGS
movq %rax,%rsi
movq %rcx,%rdi
call security_system_call
cmpl $0, %eax
jnz syscall_noperm
RESTORE_ARGS

...
call *sys_call_table(,%rax,8)

...
```

Monitoring Agent - via a new LSM hook

entry.S

ENTRY(system_call)

...

GET_THREAD_INFO(%rcx)

SAVE_ARGS

movq %rax,%rsi

movq %rcx,%rdi

call security_system_call

cmpl \$0, %eax

jnz syscall_noperm

RESTORE_ARGS

...

call *sys_call_table(,%rax,8)

...

Monitoring Agent - via a new LSM hook

entry.S

ENTRY(system_call)

...

GET_THREAD_INFO(%rcx)

SAVE_ARGS

movq %rax,%rsi

movq %rcx,%rdi

call security_system_call

cmpl \$0, %eax

jnz syscall_noperm

RESTORE_ARGS

...

call *sys_call_table(,%rax,8)

...

Monitoring Agent - via a new LSM hook

entry.S

```
ENTRY(system_call)
...
GET_THREAD_INFO(%rcx)
SAVE_ARGS
movq %rax,%rsi
movq %rcx,%rdi
call security_system_call
cmpl $0, %eax
jnz syscall_noperm
RESTORE_ARGS
...
call *sys_call_table(,%rax,8)
...
```

```
$ korset_runtime_monitor  
start
```

```
$ korset_runtime_monitor  
stop
```

Monitoring Agent

Sum up

- Integrated into the Kernel's system call handler
- Uses and extends the Linux Security Module (LSM) interface
- Simulate automaton on observed system calls
- Terminate subverted applications
- Can dynamically update in-memory DCA
- Can dump updated DCA back to disk

Userland

The Static Analyzer

User Space

example.c

```
i = read(fd, buf, n);
if (i == n) {
    write(fd, buf, n);
}
close(fd);
```

gcc, ld, ...

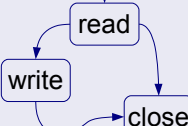
example

ELF executable

System Calls

Kernel Space

example.korset



Korset
Static Analyzer

```
$ korset_static_analyzer  
start
```

```
$ gcc -c foo.c -o foo.o
```



```
$ gcc -c bar.S -o bar.o
```

```
$ ar c foobar.a foo.o  
bar.o
```

```
$ gcc foo.o bar.o -o  
    foobar
```

foo.o.kvcg

bar.o.kvcg

foobar.a.kvcg

foobar.korset

```
$ korset_static_analyzer  
stop
```

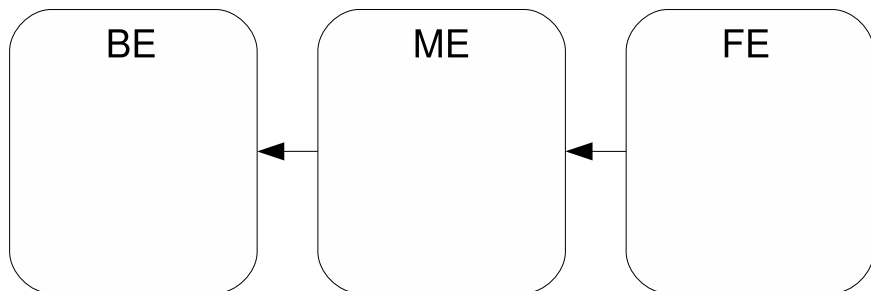

Static Analyzer

Sum up

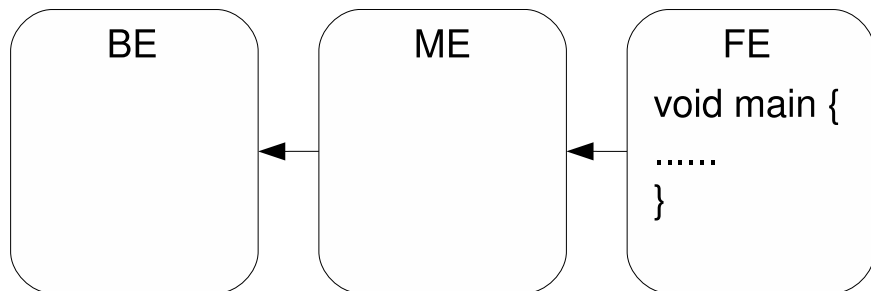
- Wraps the Linux build tools
- Transparently runs whenever user compiles, links or ar(chives)
- Creates DCAs for objects, libraries and executables

Constructing the Graphs

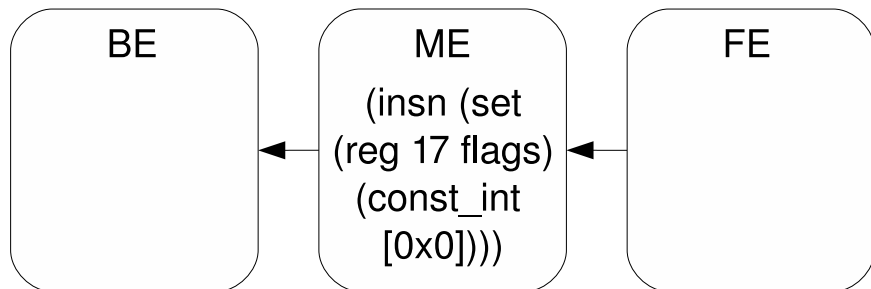
GCC saves the day



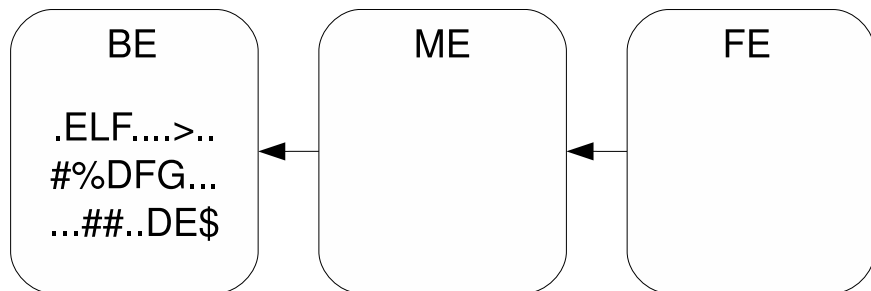
GCC saves the day



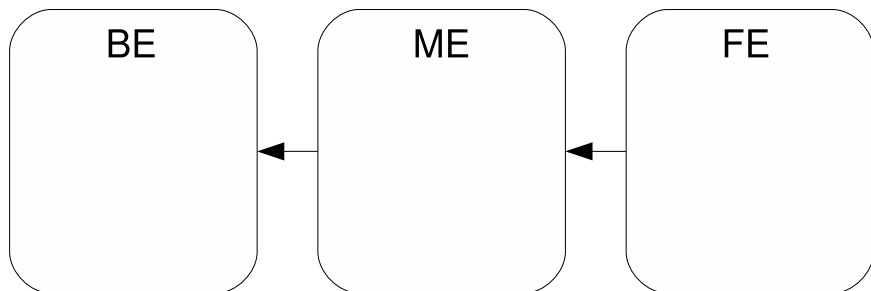
GCC saves the day



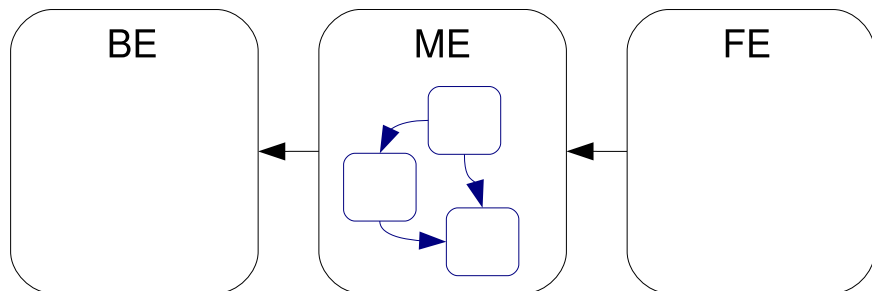
GCC saves the day



GCC saves the day



GCC saves the day



GCC saves the day

GCC Plugins ?

GCC saves the day

```
$ gcc -dv -fdump-rtl-pass
```

Visualization of Compiler Graphs (VCG)

Just parse and the CFG is yours

```
graph: { title: "hack_digit"
...
node: { title: "hack_digit.0" }
...
edge: { sourcename: "hack_digit.0" targetname:
"hack_digit.7" color: blue }

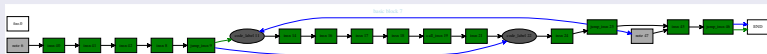
node: {
  title: "hack_digit.7"
  label: "note 7"
}
...
```

Creating CFGs for C files

Use gcc's VCG output

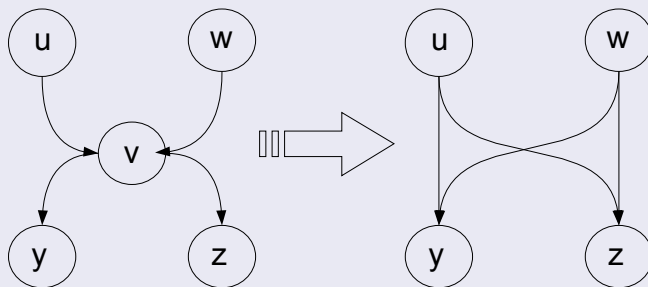
```
$ gcc -dv -fdump-rtl-pass -c foo.c
```

```
void foo(void)
{
    int i;
    for (i = 0; i < 10; i++)
        fwrite("Hello!\n", 7, 1, stdout);
}
```



Simplification Process

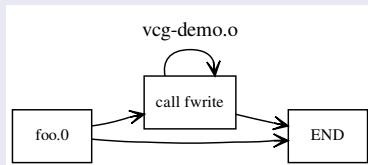
Simple and Smooth



Creating CFGs for C files

After simplifying VCG output

```
void foo(void)  
{  
    int i;  
    for (i = 0; i < 10; i++)  
        fwrite("Hello!\n", 7, 1, stdout);  
}
```



VCG Summary

Neat.

VCG Summary

Neat.

**Does not apply for Assembly
files...**

Creating CFGs for Assembly files

Lots of Macros...

```
#include <sysdep-cancel.h>
```

```
PSEUDO (__libc_read, read, 3)  
      ret
```

```
PSEUDO_END(__libc_read)
```

```
libc_hidden_def (__libc_read)  
weak_alias (__libc_read, __read)  
libc_hidden_weak (__read)  
weak_alias (__libc_read, read)  
libc_hidden_weak (read)
```

Creating CFGs for Assembly files

Disassemble corresponding object file:

```
mov    %rdx,0x18(%rsp)
callq  35 <__write_nocancel+0x2c>
       R_X86_64_PC32    __libc_enable_asynccancel
mov    0x8(%rsp),%rdi
mov    0x10(%rsp),%rsi
mov    0x18(%rsp),%rdx
mov    %rax,(%rsp)
mov    $0x1,%eax
syscall
```

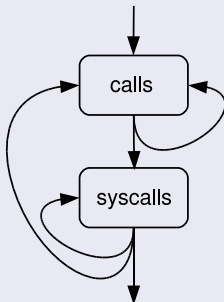
Creating CFGs for Assembly files

Look for system and function calls:

```
mov    %rdx,0x18(%rsp)
callq  35 <__write_nocancel+0x2c>
      R_X86_64_PC32    __libc_enable_asynccancel
mov    0x8(%rsp),%rdi
mov    0x10(%rsp),%rsi
mov    0x18(%rsp),%rdx
mov    %rax,(%rsp)
mov    $0x1,%eax
syscall
```

Creating CFGs for Assembly files

Create a simplified matching graph



- Crude, ok for simple files
- Sound solution
- Requires a better flow analysis

Creating CFGs for stdin files

something like this:

```
$ gcc -x c++ -o output.o -
```

redundant ?

Creating CFGs for stdin files

common glibc build:

```
(echo '#include <sysdep-cancel.h>'; \  
    echo 'PSEUDO (__libc_read, read, 3)'; \  
    echo 'ret'; \  
    echo 'PSEUDO_END(__libc_read)'; \  
    echo 'libc_hidden_def (__libc_read)'; \  
    echo 'weak_alias (__libc_read, __read)'; \  
    echo 'libc_hidden_weak (__read)'; \  
    echo 'weak_alias (__libc_read, read)'; \  
    echo 'libc_hidden_weak (read)'; \  
) | gcc -c -x assembler-with-cpp -o read.o -
```

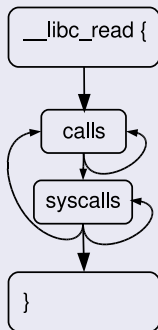
Creating CFGs for stdin files

Disassemble output file and build graph:

```
(echo '#include <sysdep-cancel.h>'; \  
    echo 'PSEUDO (__libc_read, read, 3)'; \  
    echo 'ret'; \  
    echo 'PSEUDO_END(__libc_read)'; \  
    echo 'libc_hidden_def (__libc_read)'; \  
    echo 'weak_alias (__libc_read, __read)'; \  
    echo 'libc_hidden_weak (__read)'; \  
    echo 'weak_alias (__libc_read, read)'; \  
    echo 'libc_hidden_weak (read)'; \  
) | gcc -c -x assembler-with-cpp -o read.o -
```

Creating CFGs for stdin files

Result: a simplified matching graph



Is it enough ?

common glibc build:

```
(echo '#include <sysdep-cancel.h>'; \  
    echo 'PSEUDO (__libc_read, read, 3)'; \  
    echo 'ret'; \  
    echo 'PSEUDO_END(__libc_read)'; \  
    echo 'libc_hidden_def (__libc_read)'; \  
    echo 'weak_alias (__libc_read, __read)'; \  
    echo 'libc_hidden_weak (__read)'; \  
    echo 'weak_alias (__libc_read, read)'; \  
    echo 'libc_hidden_weak (read)'; \  
) | gcc -c -x assembler-with-cpp -o read.o -
```

Pay attention to symbol aliases

common glibc build:

```
(echo '#include <sysdep-cancel.h>'; \  
    echo 'PSEUDO (__libc_read, read, 3)'; \  
    echo 'ret'; \  
    echo 'PSEUDO_END(__libc_read)'; \  
    echo 'libc_hidden_def (__libc_read)'; \  
    echo 'weak_alias (__libc_read, __read)'; \  
    echo 'libc_hidden_weak (__read)'; \  
    echo 'weak_alias (__libc_read, read)'; \  
    echo 'libc_hidden_weak (read)'; \  
) | gcc -c -x assembler-with-cpp -o read.o -
```

Collect symbol information

```
objdump -syms
```

```
read.o:          file      format      elf64-x86-64
```

SYMBOL TABLE:

00000000	g	F	.text	00000073	__libc_read
00000009	g	F	.text	00000014	__read_nocancel
00000000	w	F	.text	00000073	__read
00000000	w	F	.text	00000073	read

Collect symbol information

```
objdump -syms
```

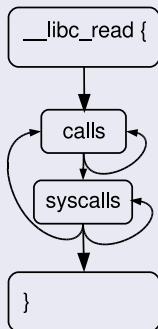
```
read.o:          file      format      elf64-x86-64
```

```
SYMBOL TABLE:
```

00000000	g	F	.text	00000073	__libc_read
00000009	g	F	.text	00000014	__read_nocancel
00000000	w	F	.text	00000073	__read
00000000	w	F	.text	00000073	read

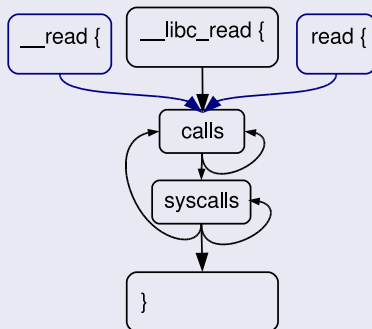
Add symbol aliases

Before



Add symbol aliases

After



Linking issues

Not all functions are equal

malloc.o: file format elf64-x86-64

SYMBOL TABLE:

000032e4	l	F	.text	0000009f	malloc_check
00001c46	l	F	.text	000000f2	free_check
00000000	w		*UND*	00000000	__dso_handle
0000395e	g	F	.text	00000331	__calloc
0000395e	w	F	.text	00000331	calloc
00001b79	g	F	.text	000000cd	__cfree
00001b79	w	F	.text	000000cd	cfree
00003e41	g	F	.text	000001cf	malloc

Linking issues

Not all functions are equal

malloc.o: file format elf64-x86-64

SYMBOL TABLE:

000032e4	I	F	.text	0000009f	malloc_check
00001c46	I	F	.text	000000f2	free_check
00000000	w		*UND*	00000000	__dso_handle
0000395e	g	F	.text	00000331	__calloc
0000395e	w	F	.text	00000331	calloc
00001b79	g	F	.text	000000cd	__cfree
00001b79	w	F	.text	000000cd	cfree
00003e41	g	F	.text	000001cf	malloc

Linking issues

Not all functions are equal

malloc.o: file format elf64-x86-64

SYMBOL TABLE:

000032e4		F	.text	0000009f	malloc_check
00001c46		F	.text	000000f2	free_check
00000000	w		*UND*	00000000	__dso_handle
0000395e	g	F	.text	00000331	__calloc
0000395e	w	F	.text	00000331	calloc
00001b79	g	F	.text	000000cd	__cfree
00001b79	w	F	.text	000000cd	cfree
00003e41	g	F	.text	000001cf	malloc

Linking issues

Not all functions are equal

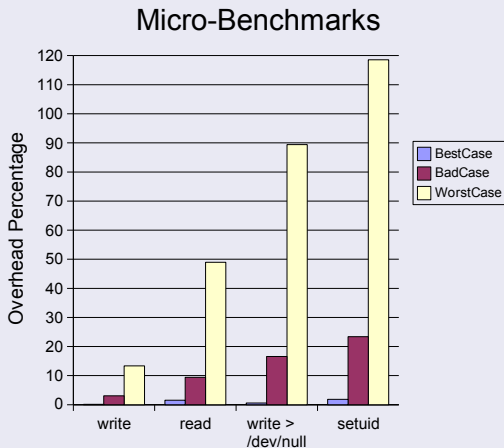
malloc.o: file format elf64-x86-64

SYMBOL TABLE:

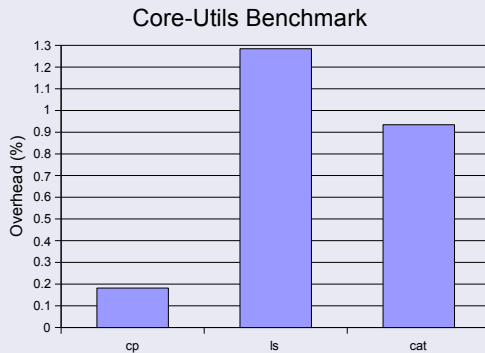
000032e4	l	F	.text	0000009f	malloc_check
00001c46	l	F	.text	000000f2	free_check
00000000	w		*UND*	00000000	__dso_handle
0000395e	g	F	.text	00000331	__calloc
0000395e	w	F	.text	00000331	calloc
00001b79	g	F	.text	000000cd	__cfree
00001b79	w	F	.text	000000cd	cfree
00003e41	g	F	.text	000001cf	malloc

Section 4: Evaluation

Micro-Benchmarks



Core-utils Benchmarks



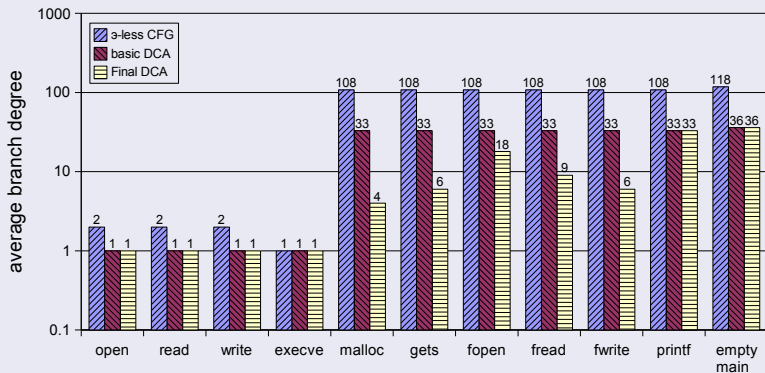
Precision Analysis

The Branching Factor

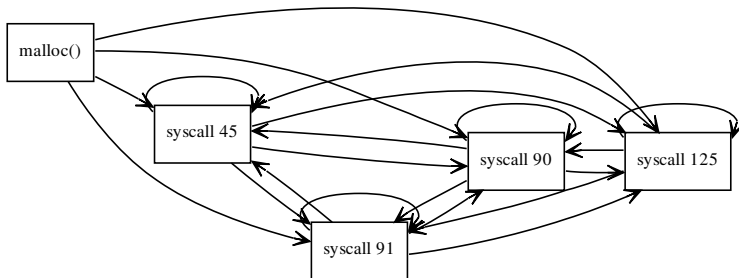
Graphs Analysis

Glibc Graph Branching

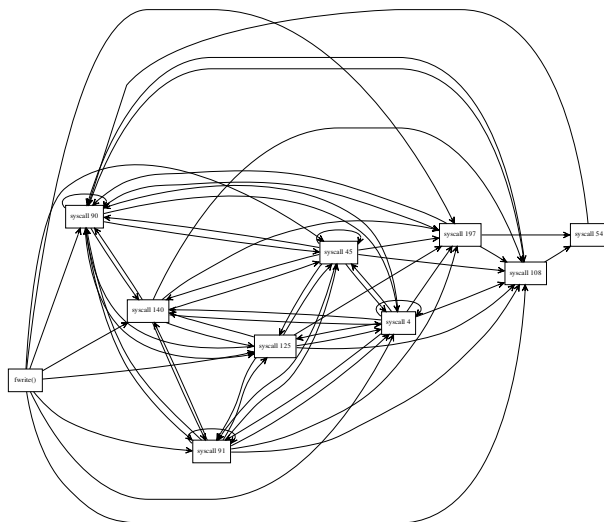
glibc DCA branching



malloc()



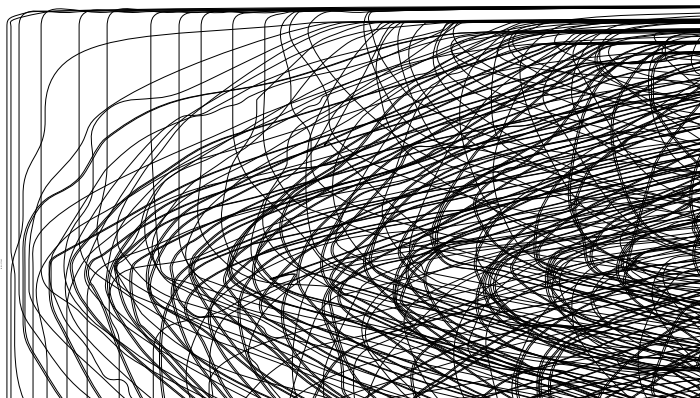
fwrite()



Empty main

```
void main(void)
{
}
```

Empty main



Section 5: Sum up

Sum up

Summary

- Zero False Positives Intrusion Detection
- Negligible (/Bounded) Runtime Overhead
- Linux Kernel Prototype
- Automatic Analysis of the GNU C library
- Free Software (GPL'ed)

Status

- Proof of concept!
- Very limited, e.g.: only static linking

<http://www.korset.org>

THE END

Thank You