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

<table>
<thead>
<tr>
<th>return address</th>
</tr>
</thead>
<tbody>
<tr>
<td>old %ebp</td>
</tr>
<tr>
<td>buffer[96]</td>
</tr>
</tbody>
</table>
Buffer Overflow

```
<table>
<thead>
<tr>
<th>return address</th>
</tr>
</thead>
<tbody>
<tr>
<td>old %ebp</td>
</tr>
<tr>
<td>buffer[96]</td>
</tr>
</tbody>
</table>
```
Buffer Overflow

```
return address
old %ebp
buffer[96]
```
Buffer Overflow

```
return address
old %ebp
buffer[96]
```
Buffer Overflow

```
return address
old %ebp
/bin/sh
shellcode
shellcode
shellcode
shellcode
```
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

True if $\text{time}() < \text{YEAR2009}$; otherwise, 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

```cpp
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

```c
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

```c
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

User Space

Static Analyzer

Kernel Space

Runtime Monitor
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
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...
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);
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);
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);
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);
open(...);

if (...) {
    for (...) {
        read(...);
    }
} else {
    write(...);
}

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

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

open(...);
if (...) {
    for (...) {
        read(...);
    }
} else {
    write(...);
}
close(...);
open(...);
if (...) {
    for (...) 
        read(...);
} else {
    write(...);
}
close(...);
open(...);
if (...) {
    for (...) 
        read(...);
} else {
    write(...);
}
close(...);
Model of Normal Behavior

System call sequences $\Rightarrow$ Paths in the graph

- open
- read
- close
- write

No path in the graph $\Rightarrow$ Invalid system call sequence
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

- open
- read
- close
- write
Protecting

Userland

open

open →

read

write

close
Protecting Userland

- open
- read
- close
- write
Protecting

Userland

read

open

read

write

close
Protecting

Userland

read

open

read

write

close
Protecting

Userland

read

read

close

open

write
Protecting

Userland

open

read

close

write
Protecting
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

```
open

foo()

close

foo()

{

read

}
```
Functions - Link CFGs

- open
- foo()
- close
- foo()
- read
- }

And... Simplify

Graph:

- open
- read
- close

Arrows indicate the sequence of operations: open → read → close.
Simplification Process

Simple and Smooth

\[
\begin{array}{c}
\text{u} \quad \text{w} \\
\downarrow \quad \downarrow \\
\text{v} \\
\downarrow \\
\text{y} \\
\end{array}
\quad \quad \quad
\begin{array}{c}
\text{u} \quad \text{w} \\
\downarrow \\
\text{y} \\
\downarrow \\
\text{z} \\
\end{array}
\]

\[
\begin{array}{c}
\text{u} \quad \text{w} \\
\downarrow \\
\text{y} \\
\downarrow \\
\text{z} \\
\end{array}
\]

Ohad Ben-Cohen  Avishai Wool

OWASP IL 2008
Challenge #1
Functions Redux - Context Insensitivityivity

Before linking

- open
  - foo()
  - close

- write
  - foo()
  - read
  - }

- write
After linking

```
open
foo()
write
read
foo()
close
write
```
Functions Redux - Context Insensitivity

After linking

... So?
After linking:

- open
- foo()
- close
- foo()
- read
- close
- write
- foo()
- write
- read
- foo()
- open

... So?

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

A Function after linking

- open
- foo()
- close

foo() {
    read
}

- write
- foo()

Impossible execution paths are allowed

E.g.: open-read-write
Hey before you link

Not all functions emit/lead to system calls

open → strcpy() → close → write

strcpy() → { → strcpy() → }

write → write
Do not link them

```
open

strcpy()

close

{

}

strcpy()

write

}```
Graph Unlinking

Just ditch their calling nodes...

- open
- close

- write

Ohad Ben-Cohen  Avishai Wool

OWASP IL 2008
Graph Inlining

Inline CFGs of functions that issue system calls

- `open`
  - `foo()`
    - `read`
      - `close`
  - `write`
- `write`
  - `foo()`
    - `read`
  - `close`
Graph Inlining

Create Private Copies

```
open
  foo()
    read
      }
    foo()
      read
        }
  close

write
  foo()
    read
      }
    write
```
Graph Inlining

Link Private Copies

- `open`
- `foo()`
- `read`
- `close`
- `write`
- `write`
- `foo()`
- `write`
- `read`
- `read`
- `foo()`
Graph Inlining

Simplify Result

- open
- read
- close
- write
- read
- write
Graph Inlining

After Simplifying

- open
  - read
  - close
- write
  - read
  - write
Graph Inlining

Inlining Depth?
Graph Inlining

Inlining Depth ?

(currently - depth 1)
Challenge #2
Non Determinism

Which write is it?

```cpp
read(...);
if (...) {
    write(...);
    close(...);
} else {
    write(...);
    exit(...);
}
```
%EIP ?
%EIP does not help

User Space

C(Func (...) {
...
write(...);
...
write(...);
...
}

C Library

write::
mov $0x1, %eax
syscall

Kernel
Solution: Merge Nodes

read

write

close

write

exit

read

write

close

exit
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

Ohad Ben-Cohen  Avishai Wool  OWASP IL 2008
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 $\epsilon$-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: $O(|\sum|)$ ($\sum$ - 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

Kernel Space

- Korset Monitoring Agent
- Kernel System Call Handler

System Calls

- read
- write
- close

Example:

- example.korset
Per process state

```c
#include <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(system_call)

... GET_THREAD_INFO(%rcx)
SAVEARGS
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, ...

ELF executable

example

Kernel Space

example.korset

Korset Static Analyzer

read

write

close

System Calls
$ 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.kvccg
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

BE

ME

FE

void main {

……

}

Ohad Ben-Cohen   Avishai Wool   OWASP IL 2008
GCC saves the day

BE

ME

(insn (set (reg 17 flags) (const_int [0x0])))

FE
GCC saves the day

BE
.ElF..>..
%%DFG...
...###..DE$
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

Ohad Ben-Cohen  Avishai Wool

OWASP IL 2008
Creating CFGs for C files

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

After simplifying VCG output

vcg-demo.o
foo.0
call fwrite
END
VCG Summary

Neat.
VCG Summary

Neat.

Does not apply for Assembly files...
Creating CFGs for Assembly files

Lots of Macros...

```c
#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)
syscall
```
Creating CFGs for Assembly files

Look for system and function calls:

- \texttt{mov}  \%rdx,0x18(\%rsp)
- \texttt{callq} 35 \textless\textgreater\texttt{write\_nocancel+0x2c}\textgreater
- \texttt{mov} 0x8(\%rsp),\%rdi
- \texttt{mov} 0x10(\%rsp),\%rsi
- \texttt{mov} 0x18(\%rsp),\%rdx
- \texttt{mov} \%rax,(\%rsp)
- \texttt{mov} $0x1,\%eax
- \texttt{syscall}

\texttt{R\_X86\_64\_PC32} \quad \texttt{__libc\_enable\_asynccancel}
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

```bash
(common glibc build:
(\(\text{echo '\#include <sysdep-cancel.h>'; }\)
 \(\text{echo 'PSEUDO (__libc_read, read, 3)'; }\)
 \(\text{echo 'ret'; }\)
 \(\text{echo 'PSEUDO_END(__libc_read)'; }\)
 \(\text{echo 'libc_hidden_def (__libc_read)'; }\)
 \(\text{echo 'weak_alias (__libc_read, _read)'; }\)
 \(\text{echo 'libc_hidden_weak (_read)'; }\)
 \(\text{echo 'weak_alias (__libc_read, read)'; }\)
 \(\text{echo 'libc_hidden_weak (read)'; }\)
) | 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

```
__libc_read {
  calls
  syscalls
}
```
Is it enough?

**common glibc build:**

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

common glibc build:

```bash
(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:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Section</th>
<th>Address</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>g</td>
<td>.text</td>
<td>00000073</td>
<td>__libc_read</td>
</tr>
<tr>
<td>00000009</td>
<td>g</td>
<td>.text</td>
<td>00000014</td>
<td>__read_nocancel</td>
</tr>
<tr>
<td>00000000</td>
<td>w</td>
<td>.text</td>
<td>00000073</td>
<td>__read</td>
</tr>
<tr>
<td>00000000</td>
<td>w</td>
<td>.text</td>
<td>00000073</td>
<td>read</td>
</tr>
</tbody>
</table>
```
Add symbol aliases

Before

```c
__libc_read {
    calls
    syscalls
}
```
Add symbol aliases

After

```
__read {
    __libc_read {
        read {
            calls
            syscalls
        }
    }
}
```
Linking issues

Not all functions are equal

<table>
<thead>
<tr>
<th>Symbol Table</th>
<th>Address</th>
<th>Type</th>
<th>Section</th>
<th>Offset</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc.o</td>
<td>0000032e4</td>
<td>I</td>
<td>.text</td>
<td>0000009f</td>
<td>malloc_check</td>
</tr>
<tr>
<td></td>
<td>000001c46</td>
<td>I</td>
<td>.text</td>
<td>000000f2</td>
<td>free_check</td>
</tr>
<tr>
<td></td>
<td>000000000</td>
<td>w</td>
<td><em>UND</em></td>
<td>00000000</td>
<td>__dso_handle</td>
</tr>
<tr>
<td></td>
<td>00000395e</td>
<td>g</td>
<td>.text</td>
<td>00000331</td>
<td>__calloc</td>
</tr>
<tr>
<td></td>
<td>00000395e</td>
<td>w</td>
<td>.text</td>
<td>00000331</td>
<td>calloc</td>
</tr>
<tr>
<td></td>
<td>00001b79</td>
<td>g</td>
<td>.text</td>
<td>000000cd</td>
<td>__cfree</td>
</tr>
<tr>
<td></td>
<td>00001b79</td>
<td>w</td>
<td>.text</td>
<td>000000cd</td>
<td>cfree</td>
</tr>
<tr>
<td></td>
<td>00003e41</td>
<td>g</td>
<td>.text</td>
<td>000001cf</td>
<td>malloc</td>
</tr>
</tbody>
</table>
### Linking issues

Not all functions are equal

<table>
<thead>
<tr>
<th>Symbol Table</th>
<th>Address</th>
<th>Type</th>
<th>Section</th>
<th>Offset</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc.o:</td>
<td>000032e4</td>
<td>l</td>
<td>.text</td>
<td>0000009f</td>
<td>malloc_check</td>
</tr>
<tr>
<td>malloc.o:</td>
<td>00001c46</td>
<td>l</td>
<td>.text</td>
<td>000000f2</td>
<td>free_check</td>
</tr>
<tr>
<td>malloc.o:</td>
<td>00000000</td>
<td>w</td>
<td><em>UND</em></td>
<td>00000000</td>
<td>__dso_handle</td>
</tr>
<tr>
<td>malloc.o:</td>
<td>0000395e</td>
<td>g</td>
<td>.text</td>
<td>00000331</td>
<td>__calloc</td>
</tr>
<tr>
<td>malloc.o:</td>
<td>0000395e</td>
<td>w</td>
<td>.text</td>
<td>00000331</td>
<td>calloc</td>
</tr>
<tr>
<td>malloc.o:</td>
<td>00001b79</td>
<td>g</td>
<td>.text</td>
<td>000000cd</td>
<td>__cfree</td>
</tr>
<tr>
<td>malloc.o:</td>
<td>00001b79</td>
<td>w</td>
<td>.text</td>
<td>000000cd</td>
<td>cfree</td>
</tr>
<tr>
<td>malloc.o:</td>
<td>00003e41</td>
<td>g</td>
<td>.text</td>
<td>000001cf</td>
<td>malloc</td>
</tr>
</tbody>
</table>
Linking issues

Not all functions are equal

**malloc.o:** file format elf64–x86–64

SYMBOL TABLE:

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Section</th>
<th>Offset</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>000032e4</td>
<td>l</td>
<td>.text</td>
<td>0000009f</td>
<td>malloc_check</td>
</tr>
<tr>
<td>00001c46</td>
<td>l</td>
<td>.text</td>
<td>000000f2</td>
<td>free_check</td>
</tr>
<tr>
<td>00000000</td>
<td>w</td>
<td><em>UND</em></td>
<td>00000000</td>
<td>_dso_handle</td>
</tr>
<tr>
<td>0000395e</td>
<td>g</td>
<td>.text</td>
<td>00000331</td>
<td>__calloc</td>
</tr>
<tr>
<td>0000395e</td>
<td>w</td>
<td>.text</td>
<td>00000331</td>
<td>calloc</td>
</tr>
<tr>
<td>00001b79</td>
<td>g</td>
<td>.text</td>
<td>000000cd</td>
<td>__cfree</td>
</tr>
<tr>
<td>00001b79</td>
<td>w</td>
<td>.text</td>
<td>000000cd</td>
<td>cfree</td>
</tr>
<tr>
<td>00003e41</td>
<td>g</td>
<td>.text</td>
<td>000001cf</td>
<td>malloc</td>
</tr>
</tbody>
</table>
## Linking issues

### Not all functions are equal

**malloc.o:**

- **file format:** elf64-x86-64

**SYMBOL TABLE:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Section</th>
<th>Offset</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000032e4</td>
<td>l</td>
<td>.text</td>
<td>0000009f</td>
<td>malloc_check</td>
</tr>
<tr>
<td>00001c46</td>
<td>l</td>
<td>.text</td>
<td>000000f2</td>
<td>free_check</td>
</tr>
<tr>
<td>00000000</td>
<td>w</td>
<td><em>UND</em></td>
<td>00000000</td>
<td>__dso_handle</td>
</tr>
<tr>
<td>0000395e</td>
<td>g</td>
<td>.text</td>
<td>00000331</td>
<td>__calloc</td>
</tr>
<tr>
<td>0000395e</td>
<td>w</td>
<td>.text</td>
<td>00000331</td>
<td>calloc</td>
</tr>
<tr>
<td>00001b79</td>
<td>g</td>
<td>.text</td>
<td>000000cd</td>
<td>__cfree</td>
</tr>
<tr>
<td>00001b79</td>
<td>w</td>
<td>.text</td>
<td>000000cd</td>
<td>cfree</td>
</tr>
<tr>
<td>00003e41</td>
<td>g</td>
<td>.text</td>
<td>000001cf</td>
<td>malloc</td>
</tr>
</tbody>
</table>
Section 4: Evaluation
Micro-Benchmarks

Overhead Percentage

write | read | write > /dev/null | setuid

BestCase | BadCase | WorstCase

Ohad Ben-Cohen  Avishai Wool  OWASP IL 2008
Core-utils Benchmarks

Core-Utils Benchmark

<table>
<thead>
<tr>
<th>Command</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cp</td>
<td>0.2</td>
</tr>
<tr>
<td>ls</td>
<td>1.3</td>
</tr>
<tr>
<td>cat</td>
<td>0.8</td>
</tr>
</tbody>
</table>
Precision Analysis

The Branching Factor
Graphs Analysis

Glibc Graph Branching

glibc DCA branching

average branch degree

- less CFG
- basic DCA
- Final DCA

<table>
<thead>
<tr>
<th>Function</th>
<th>open</th>
<th>read</th>
<th>write</th>
<th>execve</th>
<th>malloc</th>
<th>gets</th>
<th>fopen</th>
<th>fread</th>
<th>fwrite</th>
<th>printf</th>
<th>empty main</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>18</td>
<td>9</td>
<td>6</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>18</td>
<td>9</td>
<td>6</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>18</td>
<td>9</td>
<td>6</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>18</td>
<td>9</td>
<td>6</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>18</td>
<td>9</td>
<td>6</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>18</td>
<td>9</td>
<td>6</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Ohad Ben-Cohen  Avishai Wool  OWASP IL 2008
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
Thank You