Dynamic Data Excavation
or: “Gimme back my symbol table!”

Asia Slowinska
Traian Stancescu
Herbert Bos

VU University Amsterdam
Compilation is pseudo-unbreakable code

irreversibility assumption
Compilation is pseudo-unbreakable code

- Most software available only in binary form
  - malware analysis is difficult
  - forensics is difficult
  - source gets lost
  - we do not know what code is doing
  - we cannot fix it
Goals

Long term: reverse engineer complex software
Goals

Long term: reverse engineer complex software

```
push %ebp
mov %esp,%ebp
sub $0xa8,%esp
mov 0x8(%ebp),%eax
lea -0x98(%ebp),%ecx
mov %eax,%edx
mov $0x8c,%eax
mov %eax,0x8(%esp)
mov %edx,0x4(%esp)
mov %ecx,(%esp)
call 0x29
mov 0x8(%ebp),%eax
leave
ret
nop
nop
```
Goals

Long term: reverse engineer complex software
Goals

Long term: reverse engineer complex software

```c
struct employee {
    char name [128];
    int year;
    int month;
    int day;
};

struct employee *
foo (struct employee* src)
{
    struct employee dst;
    dst = *src;
    return src;
}
```
Goals

**Long term**: reverse engineer complex software

**Short term**: reverse engineer data structures

```c
struct employee {
    char name [128];
    int year;
    int month;
    int day;
};

struct employee*
foo (struct employee* src) {
    struct employee dst;
    dst = *src;
    return src;
}
```
Goals

Long term: reverse engineer complex software

Short term: reverse engineer data structures

```c
struct s1 {
    char f1 [128];
    int f2;
    int f3;
    int f4;
};
struct s1*
foo (struct s1* a1)
{
    struct s1 l1;
}
```
WHY?
Application I: legacy binary protection

• legacy binaries everywhere
• we suspect they are vulnerable

But...

How to protect legacy code from memory corruption?

**Answer:** find the buffers and make sure that all accesses to them do not stray beyond array bounds
Application II: binary analysis

- we found a suspicious binary ➔ is it malware?
- a program crashed ➔ investigate

But...

Without symbols, what can we do?

**Answer**: generate the symbols ourselves!
demo later
Example I: binary analysis

```
# file wget.gdb

# The binary is stripped
wget.gdb: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15
# gdb -q wget.gdb

Reading symbols from /home/ /dynamit_instrumented_binaries/wget/wget.gdb...done.
(gdb) b *0x805ad0
Breakpoint 1 at 0x805ad0
(gdb) run www.google.com
Starting program: /home/ /dynamit_instrumented_binaries/wget/wget.gdb www.google.com
[Thread debugging using libthread_db enabled]
--2010-08-09 16:24:00-- http://www.google.com/
Breakpoint 1, 0x805ad0 in function0 ()
(gdb) info scope function0
Scope for function0:
Symbol variables_function0 is a variable with complex or multiple locations (DWARF2), length 152.
(gdb) print variables_function0

s1 = {field_4_bytes_0 = 0, field_4_bytes_1 = 0, pointer_struct_hostent_0 = 0xbffe0, field_8_bytes_0_unsigned = 5795557898248313309,
     pointer_char_0 = 0x3bb014 '274', field_in_addr_t_0 = -1073744880, pointer_struct_1_0 = 0x0, field_1_bytes_0_unsigned = 0 '000',
      field_1_bytes_1 = 0 '000', field_8_bytes_1_unsigned = 4611705105257579775,
     inetaddr_string_0 = 0x8b0170 "www.google.com", field_4_bytes_2 = 2}
(gdb) watch variables_function0.pointer_struct_1_0
(gdb) c
Continuing.
Resolving www.google.com... Hardware watchdog 2: variables_function0.pointer_struct_1_0

Old value = (struct struct_1 *) 0x0
New value = (struct struct_1 *) 0x0805af5f in function0()
(gdb) print /x variables_function0.pointer_struct_1_0

s2 = {field_4_bytes_0 = 0x3, field_4_bytes_1 = 0x0, field_in_addr_t_0 = 0x0, field_1_bytes_0 = 0x0, field_4_bytes_1 = 0x0}
(gdb) print /x variables_function0.pointer_struct_1_0.pointer_struct_0_0

s3 = {field_4_bytes_0 = 0x2, field_in_addr_t_0 = 0x934f7d4a}
(gdb) print (char*) inet_ntoa(variables_function0.pointer_struct_1_0.pointer_struct_0_0.field_in_addr_t_0)
s4 = 0x7be46a "74.125.77.147"
(gdb) print malloc_usable_size(variables_function0.pointer_struct_1_0.pointer_struct_0_0)
s5 = 3
(gdb) print /x variables_function0.pointer_struct_1_0pointer_struct_0_0[1]

s6 = {field_4_bytes_0 = 0x2, field_in_addr_t_0 = 0x634f7d4a}
(gdb) print /x variables_function0.pointer_struct_1_0.pointer_struct_0_0[1].field_in_addr_t_0

s7 = 0x7be46a "74.125.77.99"
(gdb) print (char*) inet_ntoa(variables_function0.pointer_struct_1_0.pointer struct_0_0[2].field_in_addr_t_0)
s8 = 0x7be46a "74.125.77.104"
(gdb) n
```
Why is it difficult?

```c
1. struct employee {
2.     char name[128];
3.     int year;
4.     int month;
5.     int day
6. };
7. 
8. struct employee e;
9. e.year = 2010;
```
Why is it difficult?

MISSING
- Data structures
- Semantics
Data structures: key insight

Yes, data is “apparently unstructured”
But usage is not!
Data structures: key insight

Yes, data is “apparently unstructured”
But usage is not!
Data structures: key insight

Yes, data is “apparently unstructured”
But usage is not!

Analyse dynamically

test
KLEE
inputs
app
DDE Emu
data structures
**Intuition**

- Observe how memory is used at runtime to detect data structures.
- E.g., if $A$ is a pointer...

1. and $A$ is a function frame pointer, then $\*(A + 8)$ is perhaps a function argument.
2. and $A$ is an address of a structure, then $\*(A + 8)$ is perhaps a field in this structure.
3. and $A$ is an address of an array, then $\*(A + 8)$ is perhaps an element of this array.
Approach

- **Track pointers**
  - find root pointers
  - track how pointers derive from each other
    - for any address \( B = A + 8 \), we need to know \( A \).

- **Challenges:**
  - missing base pointers
    - for instance, a field of a struct on the stack may be updated using EBP rather than a pointer to the struct
  - multiple base pointers
    - e.g., normal access and `memset()`
Arrays are tricky

- Detection:
  - looks for chains of accesses in a loop
Arrays are tricky

- Detection:
  - looks for chains of accesses in a loop
Arrays are tricky

- Detection:
  - looks for chains of accesses in a loop
Arrays are tricky

- **Detection:**
  - looks for chains of accesses in a loop
  - and sets of accesses with same base in linear space
Interesting challenges

• Example:
  – Decide which accesses are relevant
  • Problems caused by e.g., memset-like functions

Reported by memset
Challenges

• Arrays
  – Nested loops
  – Consecutive loops
  – Boundary elements
Final mapping

• map access patterns to data structures
  - static memory : on program exit
  - heap memory : on free
  - stack frames : on return
What about semantics?
Semantics: key insight

Yes, data is “apparently unstructured”
But usage is not!

Usage (again) reveals semantics
Semantics: key insights

Yes, data is “apparently unstructured”
But usage is not!

Usage (again) reveals semantics
Semantics: key insights

Yes, data is “apparently unstructured”
But usage is not!

Usage (again) reveals semantics
Semantics: key insight

Yes, data is “apparently unstructured”
But usage is not!

Propagate types from sources + sinks
Semantics: key insights

Yes, data is “apparently unstructured”
But usage is not!

Propagate types from sources + sinks
Semantics: key insights

Yes, data is “apparently unstructured”
But usage is not!

Propagate types from sources + sinks
Semantics: key insights

Yes, data is “apparently unstructured”
But usage is not!

Propagate types from sources + sinks
Semantics: key insights

Yes, data is “apparently unstructured”
But usage is not!

Propagate types from sources + sinks
Semantics: key insights

Yes, data is “apparently unstructured”
But usage is not!

Propagate types from sources + sinks

open ("Herbert.doc", R_ONLY)
Semantics: key insights

Yes, data is “apparently unstructured”
But usage is not!

Propagate types from sources + sinks

open ("Herbert.doc", R_ONLY)
Results
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>

Variables vs. Bytes

Heap Memory

- unused arrays
- flattened
- unused
- missed
- ok
# Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>

## Heap Memory

<table>
<thead>
<tr>
<th></th>
<th>unused arrays</th>
<th>flattened</th>
<th>unused</th>
<th>missed</th>
<th>ok</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lighttpd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gzip</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fortune</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>

### Heap Memory

- **Unused arrays**
- **Flattened**
- **Unused**
- **Missed**
- **Ok**

![Heap Memory Chart]
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>

Heap Memory

% of total

- unused arrays
- flattened
- unused
- missed
- ok
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>

Heap Memory

% of total
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>

Stack Memory

- Unused arrays
- Flattened
- Unused
- Missed
- Ok

% of total
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>

![Stack Memory Chart]

- unused arrays
- flattened
- unused
- missed
- ok
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>
Results

<table>
<thead>
<tr>
<th>Prog</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>46K</td>
</tr>
<tr>
<td>fortune</td>
<td>2K</td>
</tr>
<tr>
<td>grep</td>
<td>24K</td>
</tr>
<tr>
<td>gzip</td>
<td>21K</td>
</tr>
<tr>
<td>lighttpd</td>
<td>21K</td>
</tr>
</tbody>
</table>

Stack Memory

- Unused arrays
- Flattened
- Unused
- Missed
- Ok

% of total
• consolidate Systems Security research in Europe
• promote cybersecurity education
• identify threats and vulnerabilities of the Current and Future Internet
• create active research roadmap in the area
• develop a joint working plan to conduct State-of-the-Art collaborative research.
Conclusions

- We can recover data structures by tracking memory accesses
- We believe we can protect legacy binaries
- We need to work on data coverage

More details
asia@dolphin:~/vu/dynamit_instrumented_binaries/wget$ file wget.gdb
wget.gdb: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
asia@dolphin:~/vu/dynamit_instrumented_binaries/wget$ gdb -q wget.gdb
Reading symbols from /home/asia/vu/dynamit_instrumented_binaries/wget/wget.gdb...done.
(gdb) b *0x805adb0
Breakpoint 1 at 0x805adb0
(gdb) run www.google.com
[Thread debugging using libthread_db enabled]

Breakpoint 1, 0x0805adb0 in function0 ()
(gdb)
(gdb) info scope function0
Scope for function0:
Symbol variables_function0 is a variable with complex or multiple locations (DWARF2), length 152.

(gdb) print variables_function0
$1 = {field_4_bytes_0 = 0, field_4_bytes_1 = 0, pointer_struct_hostent_0 = 0xbfffeaf0,
    field_8_bytes_0_unused = 57955879824831200, pointer_char_0 = 0x2cfb14 "\274\t",
    field_in_addr_t_0 = -1073745296,
    pointer_struct_1_0 = 0x0, field_1_byte_0_unused = 0 "\000", field_1_byte_0 = 0 "\000",
    field_1_byte_1 = 0 "\000", field_8_bytes_1_unused = -4611706891964220672,
    inetaddr_string_0 = 0x80b0170 "www.google.com", field_4_bytes_2 = 0}

(gdb) watch variables_function0.pointer_struct_1_0
Hardware watchpoint 2: variables_function0.pointer_struct_1_0
(gdb) continue
Resolving www.google.com... Hardware watchpoint 2: variables_function0.pointer_struct_1_0

Old value = (struct struct_1 *) 0x0
New value = (struct struct_1 *) 0x80b2678
0x0805af5f in function0 ()
(gdb)
(gdb) print /x *variables_function0.pointer_struct_1_0
$2 = {field_4_bytes_0 = 0x3, pointer_struct_0_0 = 0x80b2690, field_int_0 = 0x0, field_1_byte_0 = 0x0, field_4_bytes_1 = 0x0}
(gdb) print /x *variables_function0.pointer_struct_1_0.pointer_struct_0_0
$3 = {field_4_bytes_0 = 0x2, field_in_addr_t_0 = 0x634d7d4a}
(gdb) print (char*) inet_ntoa(variables_function0.pointer_struct_1_0.pointer_struct_0_0.field_in_addr_t_0)
$4 = 0xb7fe46a0 "74.125.77.99"
(gdb) print malloc_usable_size(variables_function0.pointer_struct_1_0.pointer_struct_0_0) /sizeof(*variables_function0.pointer_struct_0_0)
$5 = 3
(gdb) print /x variables_function0.pointer_struct_1_0.pointer_struct_0_0[1]
$6 = {field_4_bytes_0 = 0x2, field_in_addr_t_0 = 0x684d7d4a}
(gdb) print (char*) inet_ntoa(variables_function0.pointer_struct_1_0.pointer_struct_0_0[1].field_in_addr_t_0)
$7 = 0xb7fe46a0 "74.125.77.104"
(gdb) print /x variables_function0.pointer_struct_1_0.pointer_struct_0_0[2]
$8 = {field_4_bytes_0 = 0x2, field_in_addr_t_0 = 0x934d7d4a}
(gdb) print (char*) inet_ntoa(variables_function0.pointer_struct_1_0.pointer_struct_0_0[2].field_in_addr_t_0)
$9 = 0xb7fe46a0 "74.125.77.147"
(gdb)
(gdb) print variables_function0
$1 = {field_4_bytes_0 = 0, field_4_bytes_1 = 0, pointer_struct_hostent_0 = 0xbffea0,
field_8_bytes_0_unused = 579558798248313200, pointer_char_0 = 0x2c6b14
"\274\t", field_in_addr_t_0 = -1073745296,
p pointer_struct_1_0 = 0x0, field_1_byte_0_unused = 0 "000", field_1_byte_0 = 0 "000",
field_1_byte_1 = 0 "000", field_8_bytes_1_unused = -4611706891864220572,
inetaddr_string_0 = 0x80b0170 "www.google.com", field_4_bytes_2 = 0}