Dynamic Hooks

Hiding Control Flow Changes within Non-Control Data

Sebastian Vogl[*], Robert Gawlik[†], Behrad Garmany[†],
Thomas Kittel[*], Jonas Pfoh[*], Claudia Eckert[*], Thorsten Holz[†]

[*]Chair for IT-Security
Technische Universität München
Munich, Germany

[†]Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum
Bochum, Germany

- In general, malware needs to intercept events within the system

- In general, malware needs to intercept events within the system
- Event interception requires us to divert the control flow at runtime

- In general, malware needs to intercept events within the system
- Event interception requires us to divert the control flow at runtime
- This is accomplished by installing **hooks** into the control flow

- Types
  - Change code (Code Hooks)

- Types
  - Change code (Code Hooks)
  - Change function pointer (Data Hooks)

- Types
  - Change code (Code Hooks)
  - Change function pointer (Data Hooks)
- Researchers have presented effective detection mechanisms for both types

- Types
  - Change code (Code Hooks)
  - Change function pointer (Data Hooks)
- Researchers have presented effective detection mechanisms for both types

$\Rightarrow$ **How can we evade existing detection mechanisms?**

Assumption

- Hooks must target **persistent** control data

Assumption

- Hooks must target **persistent** control data

Dynamic Hooks: Evade existing mechanisms by targeting transient control data

Outline

Dynamic Hooks

▸ Idea

- Apply **exploitation techniques** to the problem of hooking

Dynamic Hooks

- Idea

- Apply **exploitation techniques** to the problem of hooking
- Modify non-control data to trigger **vulnerabilities**

Dynamic Hooks

▸ Idea

- Apply **exploitation techniques** to the problem of hooking
- Modify non-control data to trigger **vulnerabilities**
- Change control flow dynamically at **runtime**

- Apply **exploitation techniques** to the problem of hooking
- Modify non-control data to trigger **vulnerabilities**
- Change control flow dynamically at **runtime**

⇒ **Target transient control data**

Dynamic Hooks

- Apply **exploitation techniques** to the problem of hooking
- Modify non-control data to trigger **vulnerabilities**
- Change control flow dynamically at **runtime**

⇒ **Target transient control data**
⇒ **No evident connection between hook and control flow change**

We already **control** the target application

We already **control** the target application

- We are not affected by most **protection mechanisms**

We already **control** the target application

- We are not affected by most **protection mechanisms**
- We can modify **internal** data structures and attack **internal** functions

We already **control** the target application

- We are not affected by most **protection mechanisms**
- We can modify **internal** data structures and attack **internal** functions
- We can **prepare** our shellcode in advance

We already **control** the target application

- We are not affected by most **protection mechanisms**
- We can modify **internal** data structures and attack **internal** functions
- We can **prepare** our shellcode in advance

$\Rightarrow$ **Much stronger attacker model**

```c
struct list_head
{
        struct list_head *next;
        struct list_head *prev;
};

static void list_del(struct list_head *entry)
{
        entry->next->prev = entry->prev;
        entry->prev->next = entry->next;
}
```

Dynamic Hooks

- Example: Linux

```
struct list_head
{
        struct list_head *next;
        struct list_head *prev;
};

static void list_del(struct list_head *entry)
{
        entry->next->prev = entry->prev;
        entry->prev->next = entry->next;
}
```

write-where-what

- $[next + 8] = prev$

Dynamic Hooks

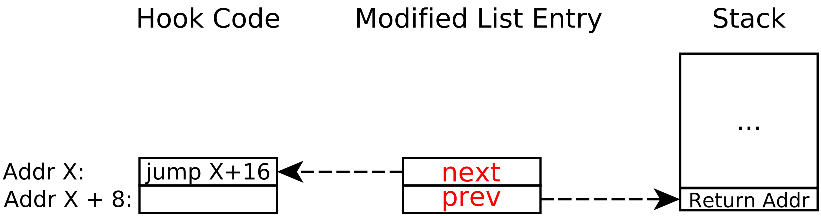
Example: Linux

```c
struct list_head
{
        struct list_head *next;
        struct list_head *prev;
};

static void list_del(struct list_head *entry)
{
        entry->next->prev = entry->prev;
        entry->prev->next = entry->next;
}
```

write-where-what

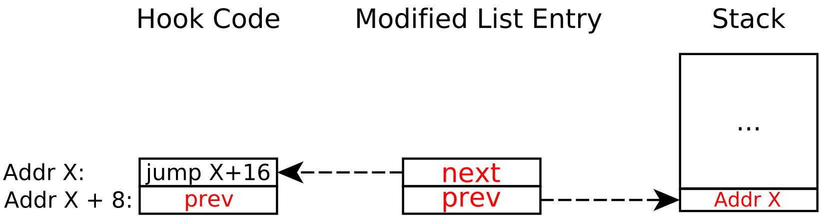
- $[next + 8] = prev$
- $[prev] = next$



Hook Code
Modified List Entry
Stack
Addr X:
Addr X + 8:
jump X+16
next
prev
...
Return Addr

Dynamic Hooks

- Example: Linux



- **Any** vulnerability can be used to implement a dynamic hook.

- **Any** vulnerability can be used to implement a dynamic hook.
- We focus on **8-byte** writes

Dynamic Hooks

▸ Suited Vulnerabilities

- **Any** vulnerability can be used to implement a dynamic hook.
- We focus on **8-byte** writes

```
mov [rax], rbx
```

- Dynamic **control** hooks

Dynamic Hooks

Types

- Dynamic **control** hooks
- Dynamic **data** hooks

- **Program Slicing**
- Symbolic Execution

- **Program Slicing**
  - mov [<destination>], <source>

- Symbolic Execution

- **Program Slicing**
  - mov [<destination>], <source>
  - backwards breadth-first search on the assembly-level
- Symbolic Execution

- **Program Slicing**
  - mov [<destination>], <source>
  - backwards breadth-first search on the assembly-level
  - extract path if destination and source originate from a global variable
- Symbolic Execution

- **Program Slicing**
  - mov [<destination>], <source>
  - backwards breadth-first search on the assembly-level
  - extract path if destination and source originate from a global variable
  - Implementation: Based on IDA Pro
- Symbolic Execution

- Program Slicing
- **Symbolic Execution**

- Program Slicing
- **Symbolic Execution**
  - transform extracted path into VEX IR (pyvex)

- Program Slicing
- **Symbolic Execution**
  - transform extracted path into VEX IR (pyvex)
  - map VEX statements into Z3 expressions

- Program Slicing
- **Symbolic Execution**
  - transform extracted path into VEX IR (pyvex)
  - map VEX statements into Z3 expressions
  - check satisfiability of conditional branches

- Program Slicing
- **Symbolic Execution**
  - transform extracted path into VEX IR (pyvex)
  - map VEX statements into Z3 expressions
  - check satisfiability of conditional branches
  - generate detailed information about controlled registers

Outline

OS	*Instructions*	*8-byte moves*	*Slices*	*Paths*
Linux	1,976,441	42,130	1753	**566**
Windows	1,330,791	26,694	5450	**379**

OS	*Instructions*	*8-byte moves*	*Slices*	*Paths*
Linux	1,976,441	42,130	1753	**566**
Windows	1,330,791	26,694	5450	**379**

Prototype Limitations

- Program Slicing: no memory model
  $\Rightarrow$ **79,853** paths ignored

OS	*Instructions*	*8-byte moves*	*Slices*	*Paths*
Linux	1,976,441	42,130	1753	**566**
Windows	1,330,791	26,694	5450	**379**

Prototype Limitations

- Program Slicing: no memory model
  $\Rightarrow$ **79,853** paths ignored
- Symbol Execution: supports only a subset of x86 instruction set
  $\Rightarrow$ **5857** slices ignored

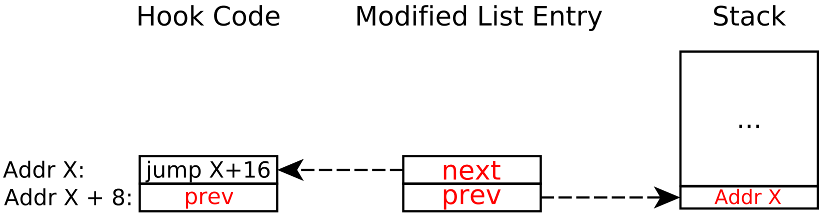
Implemented three prototypes of dynamic hooks

1. Control Hook: Interception of system calls (Linux)
2. Data Hook: Backdoor (Linux)
3. Control Hook: Interception of process termination (Windows)

Outline

Limitations

- Vulnerability may place restrictions on the hook



Hook Code
Modified List Entry
Stack
Addr X:
Addr X + 8:
jump X+16
prev
next
prev
...
Addr X

Limitations

- Vulnerability may place restrictions on the hook
- Coverage?

Limitations

- Vulnerability may place restrictions on the hook
- Coverage?
- Side effects?

Outline

Conclusion

Dynamic Hooks

Dynamic Hooks

Pros

Conclusion

Dynamic Hooks

Pros

- evade existing detection mechanisms

Dynamic Hooks

Pros

- evade existing detection mechanisms
- are more powerful than existing hooking mechanisms

Dynamic Hooks

Pros

- evade existing detection mechanisms
- are more powerful than existing hooking mechanisms
- are more difficult to detect

Dynamic Hooks

Pros

- evade existing detection mechanisms
- are more powerful than existing hooking mechanisms
- are more difficult to detect

Cons

Conclusion

Dynamic Hooks

Pros

- evade existing detection mechanisms
- are more powerful than existing hooking mechanisms
- are more difficult to detect

Cons

- are more complex than traditional hooks

Dynamic Hooks

Pros

- evade existing detection mechanisms
- are more powerful than existing hooking mechanisms
- are more difficult to detect

Cons

- are more complex than traditional hooks
- are more fragile than traditional hooks