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.
Background & Motivation

- Hooks

- 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.
Background & Motivation

- Achilles Heel: Hooks

- Types
  - Change code (Code Hooks)
Background & Motivation

- Achilles Heel: Hooks

- Types
  - Change code (Code Hooks)
  - Change function pointer (Data Hooks)
Background & Motivation

- Achilles Heel: Hooks

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

- Researchers have presented effective detection mechanisms for both types
Background & Motivation

- Achilles Heel: Hooks

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

- Researchers have presented effective detection mechanisms for both types

⇒ How can we evade existing detection mechanisms?
Background & Motivation

- Hook Detection

Assumption

- Hooks must target **persistent** control data
Background & Motivation

- Hook Detection

Assumption

- Hooks must target **persistent** control data

Dynamic Hooks: Evade existing mechanisms by targeting **transient** control data
Outline

1. Background & Motivation
2. Dynamic Hooks
3. Experiments
4. Limitations
5. Conclusion
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**
Dynamic Hooks

- Idea

- 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

- Idea

- 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**
Dynamic Hooks

- Comparison to Traditional Exploits

We already **control** the target application
Dynamic Hooks

- Comparison to Traditional Exploits

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
Dynamic Hooks

- Comparison to Traditional Exploits

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
Dynamic Hooks

- Comparison to Traditional Exploits

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

⇒ **Much stronger attacker model**
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;
}
```
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`
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`
Dynamic Hooks

- Example: Linux

```
Addr X:       jump X+16
Addr X + 8:   next
              prev
```

Stack

```
...  
Return Addr
```
Dynamic Hooks

Example: Linux

Hook Code   Modified List Entry    Stack

Addr X: jump X+16
Addr X + 8: prev

next
prev

Addr X

...
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 Hooks

- Types

- Dynamic **control** hooks
Dynamic Hooks

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

- Finding Dynamic Hooks

- Program Slicing
- Symbolic Execution
Dynamic Hooks

- Finding Dynamic Hooks

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

- Symbolic Execution
Dynamic Hooks

- Finding Dynamic Hooks

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

- **Symbolic Execution**
Dynamic Hooks

- Finding Dynamic Hooks

- 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
Dynamic Hooks

- Finding Dynamic Hooks

- 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
Dynamic Hooks

- Finding Dynamic Hooks

- Program Slicing

- Symbolic Execution
Dynamic Hooks

- Finding Dynamic Hooks

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

- Finding Dynamic Hooks

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

- Finding Dynamic Hooks

- Program Slicing

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

- Finding Dynamic Hooks

- 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

1. Background & Motivation
2. Dynamic Hooks
3. Experiments
4. Limitations
5. Conclusion
## Experiments
- Finding Dynamic Hooks

<table>
<thead>
<tr>
<th>OS</th>
<th>Instructions</th>
<th>8-byte moves</th>
<th>Slices</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1,976,441</td>
<td>42,130</td>
<td>1753</td>
<td>566</td>
</tr>
<tr>
<td>Windows</td>
<td>1,330,791</td>
<td>26,694</td>
<td>5450</td>
<td>379</td>
</tr>
</tbody>
</table>
## Experiments

### Finding Dynamic Hooks

<table>
<thead>
<tr>
<th>OS</th>
<th>Instructions</th>
<th>8-byte moves</th>
<th>Slices</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1,976,441</td>
<td>42,130</td>
<td>1753</td>
<td>566</td>
</tr>
<tr>
<td>Windows</td>
<td>1,330,791</td>
<td>26,694</td>
<td>5450</td>
<td>379</td>
</tr>
</tbody>
</table>

### Prototype Limitations

- Program Slicing: no memory model
  - 79,853 paths ignored
## Experiments

- Finding Dynamic Hooks

<table>
<thead>
<tr>
<th>OS</th>
<th>Instructions</th>
<th>8-byte moves</th>
<th>Slices</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1,976,441</td>
<td>42,130</td>
<td>1,753</td>
<td>566</td>
</tr>
<tr>
<td>Windows</td>
<td>1,330,791</td>
<td>26,694</td>
<td>5,450</td>
<td>379</td>
</tr>
</tbody>
</table>

### Prototype Limitations

- Program Slicing: no memory model  
  ⇒ **79,853** paths ignored
- Symbol Execution: supports only a subset of x86 instruction set  
  ⇒ **5,857** slices ignored
Experiments

▶ Automated Path Extraction

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)
Limitations

- Vulnerability may place restrictions on the hook
Limitations

- Automated Path Extraction

Dynamic Hooks

Hook Code | Modified List Entry | Stack
---|---|---
Addr X: | jump X+16 | next
Addr X + 8: | prev | prev
| Addr X |
Limitations

- Vulnerability may place restrictions on the hook
- Coverage?
Limitations

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

1. Background & Motivation
2. Dynamic Hooks
3. Experiments
4. Limitations
5. Conclusion
Conclusion

Dynamic Hooks

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

Cons
- They are more complex than traditional hooks
- They are more fragile than traditional hooks
Dynamic Hooks

**Pros**

1. Can evade existing detection mechanisms.
2. Are more powerful than existing hooking mechanisms.
3. Are more difficult to detect.
4. Are more complex than traditional hooks.
5. Are more fragile than traditional hooks.
Conclusion

Dynamic Hooks

Pros

- evade existing detection mechanisms
Conclusion

Dynamic Hooks

**Pros**

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

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