VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching

ACSAC 2019, San Juan

Andre Pawlowski*, Victor van der Veen†, Dennis Andriessen†, Erik van der Kouwe‡, Thorsten Holz*, Cristiano Giuffrida†, Herbert Bos†

* Ruhr-Universität Bochum † Vrije Universiteit Amsterdam ‡ Leiden University
Introduction
Virtual Callsites

```
[...] new object A
[...] mov rdi, object
call [[rdi] + offset]
[...]
```

Function X

```
[...] new object A
[...] mov rdi, object
call [[rdi] + offset]
[...]
```

Code

High Level

Traditional C++ Virtual Callsite

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Code

Function X

[...]
new object A
[...]
mov rdi, object
call [[rdi] + offset]
[...]

High Level

Object A

vtblptr
var_0
[...]

Class A

function_a1
function_a2
[...]

Traditional C++ Virtual Callsite
Virtual Callsites

Traditional C++ Virtual Callsite

Code

Function X

[...]
new object A
[...]
mov rdi, object
call [[rdi] + offset]
[...]

High Level

Object A

vtblptr
var_0
[...]

Class A

function_a1
function_a2
[...]

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Code

Function X

[...]
new object A
[...]
mov rdi, object
call [[rdi] + offset]
[...]

High Level

Object A
vtblptr
var_0
[...]

Class A
function_a1
function_a2
[...]

Traditional C++ Virtual Callsite

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Traditional C++ Virtual Callsite

Code

Function X

[...]
new object A
[...]
mov rdi, object
call [[rdi] + offset]
[...]

High Level

Class A

Object A

vtblptr
var_0
function_a1
function_a2
[...]

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Code

Function X

[...]
new object A
[...]
mov rdi, object
call [[rdi] + offset]
[...]

High Level

Vtable-Hijacking Attack

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Vtable-Hijacking Attack

Code

Function X

new object A

mov rdi, object
call [[rdi] + offset]

High Level

Object A

vtblptr
var_0

Class A

function_a1
function_a2

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Vtable-Hijacking Attack

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Code

Function X

[...]
new object A
[...]
mov rdi, object
call [[rdi] + offset]
[...]

High Level

Object A

vtblptr
var_0
[...]

Class A

function_a1
function_a2
[...]

Class Malicious

shellcode_0
shellcode_1
[...]

Vtable-Hijacking Attack

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Vtable-Hijacking Attack

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Existing C++ Control-Flow Integrity Solutions

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Existing C++ Control-Flow Integrity Solutions

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

**Code**

```
Function X
[...]
new object A
[...]
mov rdi, object
coarse security check
call [[rdi] + offset]
[...]
```

```
Object A
vtblptr
var_0
[...]
```

```
Class A
function_a1
function_a2
[...]
```

**High Level**

```
Class Malicious
shellcode_0
shellcode_1
[...]
```

Existing C++ Control-Flow Integrity Solutions

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Virtual Callsites

Existing C++ Control-Flow Integrity Solutions
Virtual Callsites

**Code**

<table>
<thead>
<tr>
<th>Function X</th>
<th>Class A</th>
<th>Class Malicious</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>vtblptr</td>
<td>shellcode_0</td>
</tr>
<tr>
<td>new object A</td>
<td>function_a1</td>
<td>shellcode_1</td>
</tr>
<tr>
<td>...</td>
<td>var_0</td>
<td>[...]</td>
</tr>
<tr>
<td>mov rdi, object</td>
<td>function_a2</td>
<td>[...]</td>
</tr>
<tr>
<td>coarse security check</td>
<td>call [[rdi] + offset]</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**High Level**

<table>
<thead>
<tr>
<th>Object A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>vtblptr</td>
<td></td>
</tr>
<tr>
<td>var_0</td>
<td></td>
</tr>
</tbody>
</table>

Existing C++ Control-Flow Integrity Solutions

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Existing C++ Control-Flow Integrity Solutions
Virtual Callsites

- Security checks are coarse grained
  - Based on heuristics (e.g., function pointers in read-only memory)
  - Based on overestimated sets (e.g., class hierarchy)
Virtual Callsites

- Security checks are coarse grained
  - Based on heuristics (e.g., function pointers in read-only memory)
  - Based on overestimated sets (e.g., class hierarchy)
- Binary-only approaches are more coarse grained than source-based approaches
Virtual Callsites

- Security checks are coarse grained
  - Based on heuristics (e.g., function pointers in read-only memory)
  - Based on overestimated sets (e.g., class hierarchy)
- Binary-only approaches are more coarse grained than source-based approaches

⇒ Shown multiple times that adversary has still enough wiggle room for attack
Virtual Callsites

Research Questions:

- Can we build a fine-grained security check?
- Can we use this security check for a binary-only approach?
Approach
Vtable Pointer Separation

- Vtable Pointer Separation
  - Store **type of object** (vtable pointer) in *safe memory* during creation
    - Based on observation that vtable pointer in object only change during creation
Vtable Pointer Separation

- Vtable Pointer Separation
  - Store **type of object** (vtable pointer) in *safe memory* during creation
    - Based on observation that vtable pointer in object only change during creation
  - Check at virtual callsite **type of object** the same as stored in *safe memory*
    - Direct mapping between object and virtual callsite
Vtable Pointer Separation

Code

Function X

[...]
new object A
write safe memory
[...]
mov rdi, object
check safe memory
call [[rdi] + offset]
[...]

High Level

Vtable Pointer Separation
Vtable Pointer Separation

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Vtable Pointer Separation

Code

Function X

[...]
new object A
write safe memory
[...]
mov rdi, object
check safe memory
call [[rdi] + offset]
[...]

High Level

Object A
vtblptr
var_0
[...]
Class A
function_a1
function_a2
[...]
Safe Memory
ObjA->ClassA
[...]
Vtable Pointer Separation

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Vtable Pointer Separation

High Level Code

Function X

mov rdi, object
call [[rdi] + offset]

Function X

new object A
write safe memory

Function X

new object A
check safe memory
call [[rdi] + offset]

Class A

function_a1
function_a2

Class Malicious

shellcode_0
shellcode_1

Safe Memory

ObjA->ClassA

Var_0

ObjA->ClassA

Vtable Pointer Separation

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
### Vtable Pointer Separation

#### Code

<table>
<thead>
<tr>
<th>Function X</th>
<th>Object A</th>
<th>Class A</th>
<th>Safe Memory</th>
<th>Class Malicious</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov rdi, object</td>
<td>vtblptr</td>
<td>function_a1</td>
<td>shellcode_0</td>
<td></td>
</tr>
<tr>
<td>call [[rdi] + offset]</td>
<td>var_0</td>
<td>function_a2</td>
<td>shellcode_1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[...]</td>
<td>[...]</td>
<td></td>
</tr>
</tbody>
</table>

#### High Level

- **Function X**: ...
- **Object A**: vtblptr, var_0, ...
- **Class A**: function_a1, function_a2, ...
- **Safe Memory**: ObjA->ClassA, ...
- **Class Malicious**: shellcode_0, shellcode_1, ...

- **check safe memory**: ...
- **write safe memory**: ...
- **new object A**: ...

---

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Vtable Pointer Separation

Code

Function X

[...]
new object A
write safe memory
[...]
mov rdi, object
check safe memory
call [[rdi] + offset]
[...]

High Level

Object A

vtblptr
var_0
[...]

Class A

function_a1
function_a2
[...]

Safe Memory

ObjA->ClassA
[...]

Class Malicious

shellcode_0
shellcode_1
[...]

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Research Questions:

- Can we build a fine-grained security check?
- Can we use this security check for a binary-only approach?
Research Questions:

- Can we build a fine-grained security check? ⇒ ✓
- Can we use this security check for a binary-only approach?
VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Approach

Analysis

- Identifying vtables in memory
- Identifying vtable pointer writes
Approach

Analysis

- Identifying vtables in memory
- Identifying vtable pointer writes
- Searching virtual callsite patterns
Approach

Analysis

- Identifying vtables in memory
- Identifying vtable pointer writes
- Searching virtual callsite patterns
- Verifying virtual callsites
Approach

Analysis

- Identifying vtables in memory
- Identifying vtable pointer writes
- Searching virtual callsite patterns
- Verifying virtual callsites

Instrumentation

- Instrument vtable pointer writes
Approach

Analysis

• Identifying vtables in memory
• Identifying vtable pointer writes
• Searching virtual callsite patterns
• Verifying virtual callsites

Instrumentation

• Instrument vtable pointer writes
• Instrument virtual callsites
Approach

Analysis

- Identifying vtables in memory
- Identifying vtable pointer writes
- Searching virtual callsite patterns
- Verifying virtual callsites

Instrumentation

- Instrument vtable pointer writes
- Instrument virtual callsites
  - Analysis Check (if not verified)
  - Security Check (if verified)
Research Questions:

- Can we build a fine-grained security check? ⇒ ✓
- Can we use this security check for a binary-only approach?
• Research Questions:
  
  • Can we build a fine-grained security check? ⇒ ✓
  • Can we use this security check for a binary-only approach? ⇒ ✓
Evaluation
Object Creation Accuracy

<table>
<thead>
<tr>
<th>Program</th>
<th>GT</th>
<th>Identified</th>
<th>Missed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>8,054</td>
<td>11,401</td>
<td>0</td>
</tr>
<tr>
<td>MySQL</td>
<td>8,532</td>
<td>11,524</td>
<td>0</td>
</tr>
<tr>
<td>Node.js</td>
<td>7,816</td>
<td>19,204</td>
<td>0</td>
</tr>
</tbody>
</table>

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
## Object Creation Accuracy

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>#Identified</th>
<th>#Missed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>8,054</td>
<td>11,401</td>
<td>0</td>
</tr>
<tr>
<td>MySQL</td>
<td>8,532</td>
<td>11,524</td>
<td>0</td>
</tr>
<tr>
<td>Node.js</td>
<td>7,816</td>
<td>19,204</td>
<td>0</td>
</tr>
</tbody>
</table>
## Object Creation Accuracy

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>#Identified</th>
<th>#Missed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>8,054</td>
<td>11,401</td>
<td>0</td>
</tr>
<tr>
<td>MySQL</td>
<td>8,532</td>
<td>11,524</td>
<td>0</td>
</tr>
<tr>
<td>Node.js</td>
<td>7,816</td>
<td>19,204</td>
<td>0</td>
</tr>
</tbody>
</table>
Virtual Callsite Identification (Ground Truth created with CFIXX)
## Virtual Callsite Identification

Virtual Callsite Identification (Ground Truth created with CFIXX)

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>Recall (%)</th>
<th>Precision (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>20,873</td>
<td>99.3</td>
<td>97.9</td>
</tr>
<tr>
<td>MySQL</td>
<td>13,035</td>
<td>99.1</td>
<td>97.1</td>
</tr>
<tr>
<td>Node.js</td>
<td>13,013</td>
<td>99.8</td>
<td>96.4</td>
</tr>
</tbody>
</table>
### Virtual Callsite Identification (Ground Truth created with CFIXX)

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>VPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Recall (%)</td>
</tr>
<tr>
<td>MongoDB</td>
<td>20,873</td>
<td>99.3</td>
</tr>
<tr>
<td>MySQL</td>
<td>13,035</td>
<td>99.1</td>
</tr>
<tr>
<td>Node.js</td>
<td>13,013</td>
<td>99.8</td>
</tr>
<tr>
<td>Program</td>
<td>#GT</td>
<td>Recall (%)</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>MongoDB</td>
<td>20,873</td>
<td>99.3</td>
</tr>
<tr>
<td>MySQL</td>
<td>13,035</td>
<td>99.1</td>
</tr>
<tr>
<td>Node.js</td>
<td>13,013</td>
<td>99.8</td>
</tr>
</tbody>
</table>
Virtual Callsite Identification (Ground Truth created with VTV)

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
## Virtual Callsite Identification (Ground Truth created with VTV)

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>Recall (%)</th>
<th>Precision (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>17,836</td>
<td>91.8</td>
<td>99.7</td>
</tr>
<tr>
<td>MySQL</td>
<td>11,876</td>
<td>97.6</td>
<td>98.5</td>
</tr>
<tr>
<td>Node.js</td>
<td>12,643</td>
<td>97.5</td>
<td>97.2</td>
</tr>
</tbody>
</table>

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
## Virtual Callsite Identification (Ground Truth created with VTV)

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>Recall (%)</th>
<th>Precision (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>17,836</td>
<td>91.8</td>
<td>99.7</td>
</tr>
<tr>
<td>MySQL</td>
<td>11,876</td>
<td>97.6</td>
<td>98.5</td>
</tr>
<tr>
<td>Node.js</td>
<td>12,643</td>
<td>97.5</td>
<td>97.2</td>
</tr>
</tbody>
</table>
### Virtual Callsite Identification

#### (Ground Truth created with VTV)

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>Recall (%)</th>
<th>Precision (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>17,836</td>
<td>91.8</td>
<td>99.7</td>
</tr>
<tr>
<td>MySQL</td>
<td>11,876</td>
<td>97.6</td>
<td>98.5</td>
</tr>
<tr>
<td>Node.js</td>
<td>12,643</td>
<td>97.5</td>
<td>97.2</td>
</tr>
</tbody>
</table>
## Virtual Callsite Identification

**Virtual Callsite Identification (Ground Truth created with VTV)**

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>VPS Recall (%)</th>
<th>VPS Precision (%)</th>
<th>Marx Recall (%)</th>
<th>Marx Precision (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>17,836</td>
<td>91.8</td>
<td>99.7</td>
<td>69.7</td>
<td>90.9</td>
</tr>
<tr>
<td>MySQL</td>
<td>11,876</td>
<td>97.6</td>
<td>98.5</td>
<td>81.3</td>
<td>88.8</td>
</tr>
<tr>
<td>Node.js</td>
<td>12,643</td>
<td>97.5</td>
<td>97.2</td>
<td>84.2</td>
<td>90.7</td>
</tr>
</tbody>
</table>

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
## Virtual Callsite Identification

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>VPS Recall (%)</th>
<th>VPS Precision (%)</th>
<th>Marx Recall (%)</th>
<th>Marx Precision (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>17,836</td>
<td>91.8</td>
<td>99.7</td>
<td>69.7</td>
<td>90.9</td>
</tr>
<tr>
<td>MySQL</td>
<td>11,876</td>
<td>97.6</td>
<td>98.5</td>
<td>81.3</td>
<td>88.8</td>
</tr>
<tr>
<td>Node.js</td>
<td>12,643</td>
<td>97.5</td>
<td>97.2</td>
<td>84.2</td>
<td>90.7</td>
</tr>
</tbody>
</table>

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
## Virtual Callsite Identification

Virtual Callsite Identification (Ground Truth created with VTV)

<table>
<thead>
<tr>
<th>Program</th>
<th>#GT</th>
<th>VPS</th>
<th>Marx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Recall (%)</td>
<td>Precision (%)</td>
</tr>
<tr>
<td>MongoDB</td>
<td>17,836</td>
<td>91.8</td>
<td>99.7</td>
</tr>
<tr>
<td>MySQL</td>
<td>11,876</td>
<td>97.6</td>
<td>98.5</td>
</tr>
<tr>
<td>Node.js</td>
<td>12,643</td>
<td>97.5</td>
<td>97.2</td>
</tr>
</tbody>
</table>

VPS: Excavating High-Level C++ Constructs from Low-Level Binaries to Protect Dynamic Dispatching
Performance Impact

- SPEC CPU2006: 11% geomean
- SPEC CPU2017: 9% geomean
Conclusion
Conclusion

• VPS provides fine-grained security checks for C++ virtual callsites
  • Binary-only approach that reaches protection level of source-based approaches
Conclusion

- VPS provides fine-grained security checks for C++ virtual callsites
  - Binary-only approach that reaches protection level of source-based approaches
- Safe Memory implementation vulnerable against advanced probing attacks
  - Not a conceptual problem, solvable through more sophisticated techniques
• VPS provides fine-grained security checks for C++ virtual callsites
  • Binary-only approach that reaches protection level of source-based approaches
• Safe Memory implementation vulnerable against advanced probing attacks
  • Not a conceptual problem, solvable through more sophisticated techniques
• Available at https://github.com/RUB-SysSec/VPS
Thank you for your attention.
Conclusion

- VPS provides fine-grained security checks for C++ virtual callsites
  - Binary-only approach that reaches protection level of source-based approaches
- Safe Memory implementation vulnerable against advanced probing attacks
  - Not a conceptual problem, solvable through more sophisticated techniques
- Available at https://github.com/RUB-SysSec/VPS