Paving the Way for NFV: Simplifying Middlebox Modifications with StateAlyzr

Junaid Khalid, Aaron Gember-Jacobson, Roney Michael, Archie Abhashkumar, Aditya Akella
Middleboxes
Middleboxes

Perform sophisticated operations on network traffic
Middleboxes

Perform sophisticated operations on network traffic

Firewall
Middleboxes

Perform sophisticated operations on network traffic

*Firewall*

*Intrusion detection system (IDS)*
Middleboxes

Perform sophisticated operations on network traffic

- **Firewall**
- **Intrusion detection system (IDS)**
- **Caching proxy**
Middleboxes

Perform sophisticated operations on network traffic

- **Firewall**
- **Intrusion detection system (IDS)**
- **Caching proxy**

Maintain state about connections and hosts
Network Function Virtualization (NFV)
Network Function Virtualization (NFV)

NFV enables *elastic scaling* and *high availability*
Network Function Virtualization (NFV)

NFV enables *elastic scaling* and *high availability*
Network Function Virtualization (NFV)

NFV enables *elastic scaling* and *high availability*
Network Function Virtualization (NFV)

NFV enables *elastic scaling* and *high availability*
Network Function Virtualization (NFV)

NFV enables *elastic scaling* and *high availability*
Network Function Virtualization (NFV)

NFV enables *elastic scaling* and *high availability*

Reroute new connections
Network Function Virtualization (NFV)

NFV enables *elastic scaling* and *high availability*
State taxonomy

State created or updated by a middlebox applies to either a single connection or a set of connections
State taxonomy

State created or updated by a middlebox applies to either a single connection or a set of connections.
State taxonomy

State created or updated by a middlebox applies to either a single connection or a set of connections

Per-conn state

Connection

TcpAnalyzer

HttpAnalyzer

Connection

TcpAnalyzer

HttpAnalyzer
State created or updated by a middlebox applies to either a **single connection** or a **set of connections**
State taxonomy

State created or updated by a middlebox applies to either a **single connection** or a **set of connections**.
State taxonomy

State created or updated by a middlebox applies to either a single connection or a set of connections.

Per-conn state

Connection

TcpAnalyzer

HttpAnalyzer

Cross-conn state

ConnCount

Input state

Config + Sig

All-conns state

Statistics
NFV state management -> middlebox modification
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

• *Require modifications* or *annotation* to middlebox code
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- Require modifications or annotation to middlebox code

Split/Merge [NSDI 2013]
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

• Require modifications or annotation to middlebox code

Required modifications:
1. State allocation
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

• *Require modifications* or *annotation* to middlebox code

Required modifications:
1. State allocation
2. State access
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- **Require modifications** or **annotation** to middlebox code

Required modifications:
1. State allocation
2. State access
3. State merge
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- *Require modifications* or *annotation* to middlebox code

<table>
<thead>
<tr>
<th>Framework</th>
<th>State Allocation</th>
<th>State Access</th>
<th>Serialization</th>
<th>Merge State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split/Merge [NSDI 2013]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state
- *Require modifications* or *annotation* to middlebox code

<table>
<thead>
<tr>
<th>Framework</th>
<th>State Allocation</th>
<th>State Access</th>
<th>Serialization</th>
<th>Merge State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split/Merge [NSDI 2013]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>OpenNF [SIGCOMM 2014]</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

*Require modifications or annotation* to middlebox code

<table>
<thead>
<tr>
<th>Framework</th>
<th>State Allocation</th>
<th>State Access</th>
<th>Serialization</th>
<th>Merge State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split/Merge [NSDI 2013]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>OpenNF [SIGCOMM 2014]</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FTMB [SIGCOMM 2015]</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- **Require modifications** or **annotation** to middlebox code

<table>
<thead>
<tr>
<th>Framework</th>
<th>State Allocation</th>
<th>State Access</th>
<th>Serialization</th>
<th>Merge State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split/Merge [NSDI 2013]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>OpenNF [SIGCOMM 2014]</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FTMB [SIGCOMM 2015]</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pico Rep. [SoCC 2013]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

*Require modifications or annotation* to middlebox code

<table>
<thead>
<tr>
<th>Framework</th>
<th>State Allocation</th>
<th>State Access</th>
<th>Serialization</th>
<th>Merge State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split/Merge [NSDI 2013]</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>OpenNF [SIGCOMM 2014]</td>
<td></td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>FTMB [SIGCOMM 2015]</td>
<td></td>
<td></td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Pico Rep. [SoCC 2013]</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stateless NF [HotMiddlebox 2015]</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why is modifying a middlebox hard?
Why is modifying a middlebox hard?

Middleboxes are complex

<table>
<thead>
<tr>
<th>MB</th>
<th>LOC (C/C++)</th>
<th>Classes/Structs</th>
<th>Level of pointers</th>
<th>Number of Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRADS</td>
<td>10K</td>
<td>40</td>
<td>4</td>
<td>297</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>62K</td>
<td>194</td>
<td>2</td>
<td>2023</td>
</tr>
<tr>
<td>HAProxy</td>
<td>63K</td>
<td>191</td>
<td>8</td>
<td>2560</td>
</tr>
<tr>
<td>Bro IDS</td>
<td>97K</td>
<td>1798</td>
<td>-</td>
<td>3034</td>
</tr>
<tr>
<td>Squid</td>
<td>166K</td>
<td>875</td>
<td>-</td>
<td>2133</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>275K</td>
<td>898</td>
<td>10</td>
<td>4617</td>
</tr>
</tbody>
</table>
Why is modifying a middlebox hard?

Middleboxes are complex

<table>
<thead>
<tr>
<th>MB</th>
<th>LOC (C/C++)</th>
<th>Classes/Structs</th>
<th>Level of pointers</th>
<th>Number of Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRADS</td>
<td>10K</td>
<td>40</td>
<td>4</td>
<td>297</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>62K</td>
<td>194</td>
<td>2</td>
<td>2023</td>
</tr>
<tr>
<td>HAProxy</td>
<td>63K</td>
<td>191</td>
<td>8</td>
<td>2560</td>
</tr>
<tr>
<td>Bro IDS</td>
<td>97K</td>
<td>1798</td>
<td>-</td>
<td>3034</td>
</tr>
<tr>
<td>Squid</td>
<td>166K</td>
<td>875</td>
<td>-</td>
<td>2133</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>275K</td>
<td>898</td>
<td>10</td>
<td>4617</td>
</tr>
</tbody>
</table>
Why is modifying a middlebox hard?

Middleboxes are complex

<table>
<thead>
<tr>
<th>MB</th>
<th>LOC (C/C++)</th>
<th>Classes/Structs</th>
<th>Level of pointers</th>
<th>Number of Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRADS</td>
<td>10K</td>
<td>40</td>
<td>4</td>
<td>297</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>62K</td>
<td>194</td>
<td>2</td>
<td>2023</td>
</tr>
<tr>
<td>HAProxy</td>
<td>63K</td>
<td>191</td>
<td>8</td>
<td>2560</td>
</tr>
<tr>
<td>Bro IDS</td>
<td>97K</td>
<td>1798</td>
<td>-</td>
<td>3034</td>
</tr>
<tr>
<td>Squid</td>
<td>166K</td>
<td>875</td>
<td>-</td>
<td>2133</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>275K</td>
<td>898</td>
<td>10</td>
<td>4617</td>
</tr>
</tbody>
</table>
Why is modifying a middlebox hard?

Middleboxes are **complex**, **diverse** and have a **variety of state**

<table>
<thead>
<tr>
<th>MB</th>
<th>LOC (C/C++)</th>
<th>Classes/Structs</th>
<th>Level of Pointers</th>
<th>Number of Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRADS</td>
<td>10K</td>
<td>40</td>
<td>4</td>
<td>297</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>62K</td>
<td>194</td>
<td>2</td>
<td>2023</td>
</tr>
<tr>
<td>HAProxy</td>
<td>63K</td>
<td>191</td>
<td>8</td>
<td>2560</td>
</tr>
<tr>
<td>Bro IDS</td>
<td>97K</td>
<td>1798</td>
<td>-</td>
<td>3034</td>
</tr>
<tr>
<td>Squid</td>
<td>166K</td>
<td>875</td>
<td>-</td>
<td>2133</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>275K</td>
<td>898</td>
<td>10</td>
<td>4617</td>
</tr>
</tbody>
</table>
Middleboxes are **complex**, **diverse** and have a **variety of state**.

<table>
<thead>
<tr>
<th>MB</th>
<th>LOC (C/C++)</th>
<th>Classes/Structs</th>
<th>Level of pointers</th>
<th>Number of Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRADS</td>
<td>10K</td>
<td>40</td>
<td>4</td>
<td>297</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>62K</td>
<td>194</td>
<td>2</td>
<td>2023</td>
</tr>
<tr>
<td>HAProxy</td>
<td>63K</td>
<td>191</td>
<td>8</td>
<td>2560</td>
</tr>
<tr>
<td>Bro IDS</td>
<td>97K</td>
<td>1798</td>
<td>-</td>
<td>3034</td>
</tr>
<tr>
<td>Squid</td>
<td>166K</td>
<td>875</td>
<td>-</td>
<td>2133</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>275K</td>
<td>898</td>
<td>10</td>
<td>4617</td>
</tr>
</tbody>
</table>
Why is modifying a middlebox hard?

Middleboxes are complex, diverse and have a variety of state

<table>
<thead>
<tr>
<th>MB</th>
<th>LOC (C/C++)</th>
<th>Classes/Structs</th>
<th>Level of pointers</th>
<th>Number of Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRADS</td>
<td>10K</td>
<td>40</td>
<td>4</td>
<td>297</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>62K</td>
<td>194</td>
<td>2</td>
<td>2023</td>
</tr>
<tr>
<td>HAProxy</td>
<td>63K</td>
<td>191</td>
<td>8</td>
<td>2560</td>
</tr>
<tr>
<td>Bro IDS</td>
<td>97K</td>
<td>1798</td>
<td>-</td>
<td>3034</td>
</tr>
<tr>
<td>Squid</td>
<td>166K</td>
<td>875</td>
<td>-</td>
<td>2133</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>275K</td>
<td>898</td>
<td>10</td>
<td>4617</td>
</tr>
</tbody>
</table>

Missing a change to some structure, class or function, may violate output equivalence.
Middleboxes are complex, diverse, and have a variety of pointers and procedures. Modifying a middlebox is hard because of the complexity and diversity of these instances.

### Output equivalence

Output equivalence: for any input, the aggregate output of a dynamic set of instances should be equivalent to the output produced by single instance.

<table>
<thead>
<tr>
<th>Middlebox</th>
<th>Lines of Code</th>
<th>Procedures</th>
<th>Level of Pointers</th>
<th>Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRADS</td>
<td>123</td>
<td>40</td>
<td>4</td>
<td>2560</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>63K</td>
<td>191</td>
<td>8</td>
<td>3034</td>
</tr>
<tr>
<td>HAProxy</td>
<td>69K</td>
<td>1798</td>
<td>-</td>
<td>2560</td>
</tr>
<tr>
<td>Bro IDS</td>
<td>97K</td>
<td>1798</td>
<td>-</td>
<td>3034</td>
</tr>
<tr>
<td>Squid</td>
<td>166K</td>
<td>875</td>
<td>-</td>
<td>2133</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>275K</td>
<td>898</td>
<td>10</td>
<td>4617</td>
</tr>
</tbody>
</table>

Missing a change to some structure, class or function may violate output equivalence.
Why is modifying a middlebox hard?

**Output equivalence:** for any input the aggregate output of a dynamic set of instances should be equivalent to the output produced by single instance.

<table>
<thead>
<tr>
<th>Middlebox</th>
<th>Size (LOC)</th>
<th>Level of pointers</th>
<th>Number of procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRADS</td>
<td>40</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>191</td>
<td></td>
<td>2560</td>
</tr>
<tr>
<td>HAProxy</td>
<td>63K</td>
<td>191</td>
<td>-</td>
</tr>
<tr>
<td>Bro IDS</td>
<td>97K</td>
<td>1798</td>
<td>3034</td>
</tr>
<tr>
<td>Squid</td>
<td>166K</td>
<td>875</td>
<td>-</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>275K</td>
<td>898</td>
<td>2133</td>
</tr>
</tbody>
</table>
Why is modifying a middlebox hard?

**Output equivalence:** for any input the aggregate output of a dynamic set of instances should be equivalent to the output produced by single instance.

<table>
<thead>
<tr>
<th>Middlebox</th>
<th>LOC (C/C++)</th>
<th>MB classes/structs</th>
<th>LOC (C/C++)</th>
<th>MB level of pointers</th>
<th>MB number of procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRADS</td>
<td>40</td>
<td>10K</td>
<td>8</td>
<td>2560</td>
<td></td>
</tr>
<tr>
<td>OpenVPN</td>
<td>121</td>
<td>2</td>
<td>194</td>
<td>3034</td>
<td></td>
</tr>
<tr>
<td>HAProxy</td>
<td>63K</td>
<td>8</td>
<td>2</td>
<td>2133</td>
<td></td>
</tr>
<tr>
<td>Bro IDS</td>
<td>97K</td>
<td>8</td>
<td>2</td>
<td>3034</td>
<td></td>
</tr>
<tr>
<td>Squid</td>
<td>166K</td>
<td>8</td>
<td>2</td>
<td>2133</td>
<td></td>
</tr>
<tr>
<td>Snort IDS</td>
<td>275K</td>
<td>10</td>
<td>4</td>
<td>4617</td>
<td></td>
</tr>
</tbody>
</table>
StateAlyzr: program analysis to the rescue
StateAlyzr: program analysis to the rescue

A system that relies on data and control-flow analysis to automatically identify state objects that need explicit handling.
StateAlyzr: program analysis to the rescue

A system that relies on *data* and *control-flow analysis* to automatically identify state objects that need explicit handling.

Leverage middlebox code structure to improve precision without compromising soundness.
StateAlyzr: program analysis to the rescue

A system that relies on data and control-flow analysis to automatically identify state objects that need explicit handling.

**Soundness** means that the system **must not miss any critical** types, storage locations, allocations, or uses of state.

Leverage middlebox code structure to **improve precision without compromising soundness**.
StateAlyzr: program analysis to the rescue

A system that relies on data and control-flow analysis can automatically identify state objects that need explicit handling.

**Soundness** means that the system *must not miss any critical* types, storage locations, allocations, or uses of state.

**Precision** means that the system identifies *the minimal set of state* that requires special handling.
StateAlyzr: program analysis to the rescue

A system that relies on data and control-flow analysis to automatically identify state objects that need explicit handling.

**Soundness** means that the system **must not miss any critical** types, storage locations, allocations, or uses of state.

**Precision** means that the system identifies **the minimal set of state** that requires special handling.
StateAlyzr: program analysis to the rescue

A system that relies on data and control-flow analysis can automatically identify state objects that need explicit handling. Soundness means that the system must not miss any critical types, storage locations, allocations, or uses of state.

Precision means that the system identifies the minimal set of state that requires special handling.
Fault tolerance IDS
Fault tolerance IDS

Primary

Hot standby
Fault tolerance IDS

Config state

Primary

Hot standby
Fault tolerance IDS

- Per flow state
- Multi flow state
- All state
- Config state

Primary

Hot standby
Fault tolerance IDS

- Per flow state
- Multi flow state
- All state
- Config state

The primary sends a copy of the state to the hot standby after each packet
Fault tolerance IDS

The **primary** sends a copy of the state to the **hot standby** after each packet.

- Per flow state
- Multi flow state
- All state
- Config state

- Primary
- Hot standby
StateAlyzr

All State

Per-/Cross-Flow State

Output-Impacting State
Logical structure of middlebox code
Logical structure of middlebox code

Main

init()
Logical structure of middlebox code

Main

init() loopProcedure()
Logical structure of middlebox code

Main

while (!done)

packet = receive()

process(packet)

send(packet) write(log)

Packet processing loop

Packet processing procedures

init() loopProcedure()

raiseEvent() process(packet)

foo()
1. Per-/cross-flow state identification
1. Per-/cross-flow state identification

*Variables* corresponding to per-/cross-flow state must be *persistent*
1. Per-/cross-flow state identification

*Variables* corresponding to per-/cross-flow state must be *persistent*

Persistent state can be stored in

1. Global variables
1. Per-/cross-flow state identification

*Variables* corresponding to per-/cross-flow state must be *persistent*

Persistent state can be stored in

1. Global variables
2. Static variables
1. Per-/cross-flow state identification

**Variables** corresponding to per-/cross-flow state must be *persistent*

Persistent state can be stored in
1. Global variables
2. Static variables
3. Local variables declared in loop proc.

```c
int loopProcedure(int *threshold) {
    int count = 0;
    while(1) {
        struct pcap_pkthdr pcapHdr;
        char *pkt = pcap_next(extPcap, &pcapHdr);
        ...
    }
}
```
1. Per-/cross-flow state identification

**Variables** corresponding to per-/cross-flow state must be *persistent*

Persistent state can be stored in

1. Global variables
2. Static variables
3. Local variables declared in loop proc.

```c
int loopProcedure(int *threshold) {
    int count = 0;
    while(1) {
        struct pcap_pkthdr pcapHdr;
        char *pkt = pcap_next(extPcap, &pcapHdr);
        ...
    }
}
```
1. Per-/cross-flow state identification

**Variables** corresponding to per-/cross-flow state must be **persistent**

Persistent state can be stored in:
1. Global variables
2. Static variables
3. Local variables declared in loop proc.

```c
int loopProcedure(int *threshold) {
    int count = 0;
    while(1) {
        struct pcap_pkthdr pcapHdr;
        char *pkt = pcap_next(extPcap, &pcapHdr);
        ...
    }
}
```
1. Per-/cross-flow state identification

**Variables** corresponding to per-/cross-flow state must be *persistent*

Persistent state can be stored in

1. Global variables
2. Static variables
3. Local variables declared in loop proc.

```c
int loopProcedure(int *threshold) {
    int count = 0;
    while(1) {
        struct pcap_pkthdr pcapHdr;
        char *pkt = pcap_next(extPcap, &pcapHdr);
    }
}
```
1. Per-/cross-flow state identification

**Variables** corresponding to per-/cross-flow state must be *persistent*

Persistent state can be stored in

1. Global variables
2. Static variables
3. Local variables declared in loop proc.

```c
int loopProcedure(int *threshold) {
    int count = 0;
    while(1) {
        struct pcap_pkthdr pcapHdr;
        char *pkt = pcap_next(extPcap, &pcapHdr);
        .
    }
}
```
1. Per-/cross-flow state identification
1. Per-/cross-flow state identification
1. Per-/cross-flow state identification
1. Per-/cross-flow state identification

Improve precision by considering variables which are used in packet processing code
1. Per-/cross-flow state identification

How to identify packet processing code?

Improve precision by considering variables which are used in packet processing code
while (!done)
packet = receive()
process(packet)
send(packet) write(log)

Packet processing loop

Packet processing procedures

1. Per-/cross-flow state identification

How to identify packet processing code?
while (!done)
    packet = receive()
    process(packet)
    send(packet)
    write(log)

Packet processing loop

init()
loopProcedure()

How to identify packet processing code?

1. Per-/cross-flow state identification

Packet processing procedures

process(packet) → foo()
1. Per-/cross-flow state identification

How to identify packet processing code?
1. Per-/cross-flow state identification

How to identify packet processing code?

```c
struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
```
How to identify packet processing code?

1. Per-/cross-flow state identification

```
struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
```
while (!done)
packet = receive()
process(packet)
send(packet)
write(log)

Packet processing loop

while (event = dequeue())

Event thread

processIndirect(event)

Packet processing procedures

raiseEvent()

process(packet)
foo()

How to identify packet processing code?

struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
1. Per-/cross-flow state identification

How to identify packet processing code?

Computes a **forward slice** from packet recv function. Any procedure appearing **in the slice** is considered as **packet processing procedure**.
2. Identify updateable state
2. Identify updateable state
2. Identify updateable state
2. Identify updateable state

Whether the state is updated while processing the packet?
2. Identify updateable state

Whether the state is updated while processing the packet?

• Strawman approach
  • Identify top-level variable on the left-hand-side (LHS) of assignment statement

```plaintext
in_port = pkt.src_port;
```

<table>
<thead>
<tr>
<th>Read-only</th>
<th>Updateable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conf</td>
<td>Per</td>
</tr>
<tr>
<td></td>
<td>Multi</td>
</tr>
<tr>
<td></td>
<td>All</td>
</tr>
</tbody>
</table>
2. Identify updateable state

Whether the state is updated while processing the packet?

- Strawman approach
  - Identify top-level variable on the left-hand-side (LHS) of assignment statement

Falls short due to *aliasing*
2. Identify updateable state

Whether the state is updated while processing the packet?

• Strawman approach

  • Identify top-level variable on the left-hand-side (LHS) of assignment statement

Falls short due to *aliasing*

```c
int *index = &tail;
*index = (*index + 1) % 100;
```
2. Identify updateable state

Whether the state is updated while processing the packet?

• Strawman approach
  • Identify top-level variable on the left-hand-side (LHS) of assignment statement

StateAlyzer employs flow-, context-, and field-insensitive pointer analysis to identify updateable variables.
3. Identify states’ flowspace dimensions
3. Identify states’ flowspace dimensions
3. Identify states’ flowspace dimensions
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state.
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state.

Common access patterns
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state

Common access patterns
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state.

Common access patterns

1. Square brackets
   
   ```
   entry = table[index];
   ```
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state

Common access patterns

1. Square brackets
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state

Common access patterns

1. Square brackets
2. Pointer arithmetic
   
   \[
   \text{entry} = \text{head + offset};
   \]
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state

Common access patterns

1. Square brackets
2. Pointer arithmetic
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state

Common access patterns

1. Square brackets
2. Pointer arithmetic
3. Iteration
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state

Common access patterns

1. Square brackets
2. Pointer arithmetic
3. Iteration

```c
struct host *lookup(uint ip) {
    struct host *curr = hosts;
    while (curr != NULL) {
        if (curr->ip == ip)
            return curr;
        curr = curr->next;
    }
}
```
3. Identify states’ flowspace dimensions

Identify a set of \textit{packet header fields} that delineate the subset of traffic that relates to the state

entry = host_map[index]
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state

Program *chopping* to determine relevant *header fields*

```
struct pktHdr *pkt = recv(extPcap);
entry = host_map[index]
```
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state

Program *chopping* to determine relevant *header fields*

```c
struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
entry = host_map[index]
```
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state

Program *chopping* to determine relevant *header fields*

```c
struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
entry = host_map[index]
```
3. Identify states’ flowspace dimensions

Identify a set of *packet header fields* that delineate the subset of traffic that relates to the state.

Program *chopping* to determine relevant *header fields*.
StateAlyzr steps
1. Identify Per-/Cross-flow state
StateAlyzr steps

1. Identify Per-/Cross-flow state
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State
3. Identify States’ Flowspace Dimensions
1. Identify Per-/Cross-flow state
2. Identify Updateable State
3. Identify States’ Flowspace Dimensions
4. Output Impacting State
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State
3. Identify States’ Flowspace Dimensions
4. Output Impacting State
   - Identify the type of output (log or packet) that updateable state affects
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State
3. Identify States’ Flowspace Dimensions
4. Output Impacting State
   • Identify the type of output (log or packet) that updateable state affects
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State
3. Identify States’ Flowspace Dimensions
4. Output Impacting State
   • Identify the type of output (log or packet) that updateable state affects
5. Tracking Run-time Update
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State
3. Identify States’ Flowspace Dimensions
4. Output Impacting State
   • Identify the type of output (log or packet) that updateable state affects
5. Tracking Run-time Update
   • Insert statements to do run time monitoring to track whether a variable is updated
Implementation
Implementation

Used CodeSurfer to implement StateAlyzr
Used CodeSurfer to implement StateAlyzr
  • CodeSurfer has built-in support for
Implementation

Used CodeSurfer to implement StateAlyzr
  • CodeSurfer has built-in support for
    • Control flow graph construction
Used CodeSurfer to implement StateAlyzr
  • CodeSurfer has built-in support for
    • Control flow graph construction
    • Flow and context-insensitive pointer analysis
Implementation

Used CodeSurfer to implement StateAlyzr
  • CodeSurfer has built-in support for
    • Control flow graph construction
    • Flow and context-insensitive pointer analysis
    • Forward/backward slice and chop computation
Used CodeSurfer to implement StateAlyzr
  • CodeSurfer has built-in support for
    • Control flow graph construction
    • Flow and context-insensitive pointer analysis
    • Forward/backward slice and chop computation

Analyzed four open-source middleboxes
Implementation

Used CodeSurfer to implement StateAlyzr

• CodeSurfer has built-in support for
  • Control flow graph construction
  • Flow and context-insensitive pointer analysis
  • Forward/backward slice and chop computation

Analyzed four open-source middleboxes

1. PRADS – a monitoring middlebox
Implementation

Used CodeSurfer to implement StateAlyzr
  • CodeSurfer has built-in support for
    • Control flow graph construction
    • Flow and context-insensitive pointer analysis
    • Forward/backward slice and chop computation

Analyzed four open-source middleboxes
  1. PRADS – a monitoring middlebox
  2. Snort – an IDS
Used CodeSurfer to implement StateAlyzr
  • CodeSurfer has built-in support for
    • Control flow graph construction
    • Flow and context-insensitive pointer analysis
    • Forward/backward slice and chop computation

Analyzed four open-source middleboxes
  1. PRADS – a monitoring middlebox
  2. Snort – an IDS
  3. HAProxy – a load balancing proxy
Used CodeSurfer to implement StateAlyzer

- CodeSurfer has built-in support for
  - Control flow graph construction
  - Flow and context-insensitive pointer analysis
  - Forward/backward slice and chop computation

Analyzed four open-source middleboxes

1. PRADS – a monitoring middlebox
2. Snort – an IDS
3. HAProxy – a load balancing proxy
4. OpenVPN – a VPN gateway
Evaluation
Evaluation

• Precision
Evaluation

• Precision
• Performance benefits at run time
## Evaluation: effectiveness

<table>
<thead>
<tr>
<th>MB</th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All variables</td>
<td>Persistent variables</td>
<td>per-/cross-flow variables</td>
</tr>
<tr>
<td>PRADS</td>
<td>1529</td>
<td>61</td>
<td>29</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>18393</td>
<td>507</td>
<td>333</td>
</tr>
<tr>
<td>HAproxy</td>
<td>7876</td>
<td>272</td>
<td>176</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>8704</td>
<td>156</td>
<td>131</td>
</tr>
</tbody>
</table>
## Evaluation: effectiveness

<table>
<thead>
<tr>
<th></th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All variables</td>
<td>Persistent variables</td>
<td>per-/cross-flow variables</td>
</tr>
<tr>
<td>PRADS</td>
<td>1529</td>
<td>61</td>
<td>29</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>18393</td>
<td>507</td>
<td>333</td>
</tr>
<tr>
<td>HAproxy</td>
<td>7876</td>
<td>272</td>
<td>176</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>8704</td>
<td>156</td>
<td>131</td>
</tr>
</tbody>
</table>
Evaluation: effectiveness

<table>
<thead>
<tr>
<th></th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All variables</td>
<td>Persistent variables</td>
<td>per-/cross-flow variables</td>
</tr>
<tr>
<td>MB</td>
<td>1529</td>
<td>61</td>
<td>29</td>
</tr>
<tr>
<td>PRADS</td>
<td>18393</td>
<td>507</td>
<td>333</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>7876</td>
<td>272</td>
<td>176</td>
</tr>
<tr>
<td>HAproxy</td>
<td>8704</td>
<td>156</td>
<td>131</td>
</tr>
<tr>
<td>OpenVPN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

StateAlyzr offers useful *improvements* in *precision*
Evaluation: effectiveness

<table>
<thead>
<tr>
<th></th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All variables</td>
<td>Persistent variables</td>
<td>per-/cross-flow variables</td>
</tr>
<tr>
<td>MB</td>
<td>1529</td>
<td>61</td>
<td>29</td>
</tr>
<tr>
<td>PRADS</td>
<td>18393</td>
<td>507</td>
<td>333</td>
</tr>
<tr>
<td>Snort IDS</td>
<td>7876</td>
<td>272</td>
<td>176</td>
</tr>
<tr>
<td>HAproxy</td>
<td>8704</td>
<td>156</td>
<td>131</td>
</tr>
<tr>
<td>OpenVPN</td>
<td>8704</td>
<td>156</td>
<td>131</td>
</tr>
</tbody>
</table>

**StateAlyzr** offers useful *improvements* in *precision*

Theoretically *proved* the *soundness* of our algorithms.
Highly available PRADS
Highly available PRADS
Highly available PRADS

State transfer after each packet

Primary

Hot standby
Highly available PRADS

![Graph showing state transfer per packet number](image)

- State transfer after each packet
- Primary
- Hot standby

- All persistent state
- All updatable state
- Flowspace

<table>
<thead>
<tr>
<th>packet number</th>
<th>per pkt state transfer (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5 k</td>
<td>10</td>
</tr>
<tr>
<td>10 k</td>
<td>100</td>
</tr>
<tr>
<td>15 k</td>
<td>1000</td>
</tr>
<tr>
<td>20 k</td>
<td>10000</td>
</tr>
<tr>
<td>25 k</td>
<td>10000</td>
</tr>
<tr>
<td>30 k</td>
<td>10000</td>
</tr>
<tr>
<td>35 k</td>
<td>10000</td>
</tr>
</tbody>
</table>
Highly available PRADS

- State transfer after each packet

Graph:
- Y-axis: per pkt state transfer (KB)
- X-axis: packet number
- Lines:
  - Green: All persistent state
  - Purple: All updatable state
  - Red: Flowspace

Diagram:
- Primary
- Hot standby
- State transfer after each packet
Highly available PRADS

State transfer after each packet

- Primary
- Hot standby

- All persistant state
- All updatable state
- Flowspace

Graph showing per pkt state transfer (KB) vs. packet number.
Highly available PRADS

State transfer after each packet

Reduction in the state transfer by 305x
Highly available PRADS

Reduction in the state transfer by 305x
Highly available PRADS

- **StateAlyzr** reduced the manual effort of modifying PRADS from **120hrs** to **6 hrs**

Reduction in the state transfer by **305x**
Highly available PRADS

- **StateAlyzr** reduced the manual effort of modifying PRADS from *120hrs* to *6 hrs*
- **StateAlyzr** found a compound variable which we *missed* in our prior modification.

Reduction in the state transfer by *305x*
Summary
Summary

• Goal is to aid middlebox developers to identify state objects that need explicit handling
Summary

• Goal is to aid middlebox developers to identify state objects that need explicit handling

• Novel state characterization algorithms that adapt standard program analysis tools
Summary

• Goal is to aid middlebox developers to identify state objects that need explicit handling
• Novel state characterization algorithms that adapt standard program analysis tools
• Ensure soundness and high precision
Summary

• Goal is to aid middlebox developers to identify state objects that need explicit handling
• Novel state characterization algorithms that adapt standard program analysis tools
• Ensure soundness and high precision
• Ultimate goal is to fully automate the process