Symnet: scalable symbolic execution for modern networks

University Politehnica of Bucharest

Radu Stoinescu, Matei Popovici, Lorina Negreanu and Costin Raiciu
Networks are increasingly complex
Understand the network

Reachability
Packet modifications
Security policy violations
Static verification to the rescue

Data plane snapshot

Symbolic Execution Friendly Language (SEFL) - Network model

Symnet - Verification engine
Choosing a modeling language

C code

• Expressive, well understood
• Symbolic execution captures many properties
• Very expensive to verify

Header Space Analysis

• Cheap, scalable
• No arbitrary protocol layering
• Only captures reachability

Middle ground
Symbol execution of firewall - C code

1: packet* filter(packet* p){
2:    if (p->dst_port==80)
4:        return p;
5:    else {
6:        free p;
7:        return NULL;
8:    }
9:}
Symbol execution of firewall - C code

1: packet* filter(packet* p){
2:    if (p->dst_port==80)
4:        return p;
5:    else {
6:        free p;
7:        return NULL;
8:    }
9:}

Path 1
p=*
Symbol execution of firewall - C code

1:   packet* filter(packet* p){
2:     if (p->dst_port==80)
4:       return p;
5:     else {
6:       free p;
7:       return NULL;
8:     }
9:   }
Symbol execution of firewall - C code

1: packet* filter(packet* p) {
2:   if (p->dst_port == 80) { 
4:     return p;
5:   } else { 
6:     free p;
7:     return NULL;
8:   }
9: }

Path 1

Path 2

\( p->\text{dst}\_\text{port}=80 \quad p->\text{dst}\_\text{port}!\neq 80 \)
Symbol execution of firewall - C code

1: packet* filter(packet* p){
2:     if (p->dst_port==80)  
3:         return p;        
4:     else {
5:         free p;         
6:         return NULL;  
7:     }
8: }

Path 1

Path 2

p->dst_port=80  p->dst_port!=80
Symbol execution of firewall - C code

1: packet* filter(packet* p){
2:     if (p->dst_port==80)
3:         return p;
4:     else {
5:         free p;
6:         return NULL;
7:     }
8: }

Path 1

Path 2
Symbol execution of firewall - C code

```
1: packet* filter(packet* p){
2:   if (p->dst_port==80)
4:     return p;
5:   else {
6:     free p;
7:     return NULL;
8:   }
9: }
```
Symbol execution of firewall - C code

1: packet* filter(packet* p){
2:    if (p->dst_port==80)
4:        return p;
5:    else {
6:        free p;
7:        return NULL;
8:    }
9:}

Path 1

Path 2

p->dst_port!=80
p->dst_port=80
filter = p
Symbol execution of firewall - C code

1: packet* filter(packet* p){
2:     if (p->dst_port==80) {
4:         return p;
5:     }
6: else {
7:         free p;
8:     return NULL;
8: }
9: }

Path 1

Path 2

1. If the destination port is 80, return the packet.
2. Else, free the packet and return NULL.
Symbol execution of firewall - C code

1: packet* filter(packet* p){
2:   if (p->dst_port==80)
4:     return p;
5:   else {
6:     free p;
7:     return NULL;
8:   }
9:}

Path 1

Path 2

p->dst_port=80
filter = p

p=NULL
Symbol execution of firewall - C code

1: packet* filter(packet* p){
2:   if (p->dst_port==80)
4:     return p;
5:   else {
6:     free p;
7:     return NULL;
8:   }
9:}

Path 1

Path 2

p->dst_port=80
filter = p

p= NULL
filter = NULL
Symbol execution of firewall - C code

1: packet* filter(packet* p){
2:   if (p->dst_port==80)
4:     return p;
5:   else {
6:     free p;
7:     return NULL;
8:   }
9:}

Two symbolic paths vs. one viable in the network
Non-packet processing being executed
Symbol execution of firewalls - C code

Firewall #1

Firewall #2

Firewall #3

N-1 unnecessary symbolic paths
Symbolic execution of network data plane implementations does not scale

• A core IP router results in hundreds of thousands of paths

• For a TCP options-parsing middlebox, runtime depends on option length (<40):
  – 6B ~ 1 hour, 7B ~ 3 hours
Principles for scalable data plane symbolic execution

Fundamental tradeoff between fast symbolic execution and runtime efficiency [Wagner‘13]
=> Use models of networks instead of real code

Only analyze relevant code
=> 1 execution path == 1 network packet

Complex data structures kill symbolic execution
=> Use symbolic-execution friendly data structures

Loops + conditionals are dangerous
=> Careful looping semantics with low branching factor
Our solution

**SEFL** symbolic execution friendly language

**Symnet** symbolic execution tool

**Memory safety by design**
- The memory space is the packet
- No pointers
- Memory access via concrete offsets; validated

**Symbolic execution constructs part of the language**
- Explicit forking of new execution paths
- Explicit stating of path constraints

**No arbitrary data structures**
- Only a map data structure
Variables are packet headers or metadata

- **Packet headers** allocated at specific addresses in the packet header
- **Metadata** are key/value pairs in a map data structure
The packet header in SEFL

CreateTag("L3",0)

Allocate(IpSrc,96)
Assign(IpSrc,"192.168.1.1")

Allocate(IpDst,32)
Assign(IpDst,Symbolic)

CreateTag("L2",Tag("L3")-112)
Assign(DstMac,Symbolic)  ERROR
Firewall

C

1: packet* filter(packet* p) {
2: if (p->dst_port == 80) {
3: return p;
4: } else {
5: free p;
6: return NULL;
7: }
8: }
9:}

SEFL

1: filter() {
2: constrain(IpDst==80);
3: }

Only relevant paths explored
Concise
Symnet symbolic execution tool

- 10K LOC of Scala; Z3 for constraint solving

Input: SEFL network model
  - SEFL models of individual network elements
  - Connections between elements

Output: all feasible symbolic paths
  - Values of header and metadata fields
  - Path constraints
SEFL Network Models

Element A

Element B
Symbolic execution of filter + DNAT

**Element A model**

InputPort(0):
- Constrain(IPDst==1.1.1.1),
- If (Constrain(TcpDst==20),
  InstructionBlock(
    Assign(IPDst,192.168.0.1),
    Assign(TcpDst,30),
    Forward(OutputPort(0))
  ),
  Forward(OutputPort(1))
),
- Forward(OutputPort(1)),

Packet 1
- IpDst=*  TcpDst=*  
- IpDst=1.1... TcpDst=*  
- IpDst=192... TcpDst=20  
- IpDst=192... TcpDst=30 
- CrtPort = 0

Packet 2
- IpDst=1.1,  
  TcpDst != 20  
- CrtPort = 1

- Reachability
- Loop detection
- Invariant header fields
- Header memory safety
Ready-made network models

Modeling network boxes is fairly difficult
We have developed parsers that output SEFL code from:

- Router/switch forwarding table snapshots
- CISCO ASA firewall configuration
- Click modular router configurations
- Openstack Neutron network configurations
Evaluation

Model correctness

Functionality

Scalability
## Verifiable properties

<table>
<thead>
<tr>
<th>Property</th>
<th>HSA</th>
<th>NoD</th>
<th>SymNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Loop Detection</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Header Field Invariance</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Arbitrary Packet Layout</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Tunneling</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Stateful Data Plane Processing</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Payload-sensitive Processing</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Properties Across Multiple Flows</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Does Symnet scale?

![Graph showing verification time vs. MAC entries in switch table (K). The Basic curve indicates 'out of memory'.]
Does Symnet scale?

![Graph showing verification time vs. MAC entries in switch table (K)]

- Basic
- Ingress

out of memory
Does Symnet scale?

![Graph showing verification time vs. MAC entries in switch table](image)

- **Basic**
- **Ingress**
- **Egress**

Out of memory
Analyzing bigger networks

- Stanford university backbone network
- Switches, routers and VLANs
  - Two-layer topology
  - Core routers have 180,000 entries in their FIBs

<table>
<thead>
<tr>
<th></th>
<th>HSA</th>
<th>Symnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Generation Time</td>
<td>3.2 min</td>
<td>8.1 min</td>
</tr>
<tr>
<td>Runtime</td>
<td>24s</td>
<td>37s</td>
</tr>
</tbody>
</table>
Conclusions

SEFL + Symnet offers a deeper understanding of modern data planes at a low price.

Symnet is open-source
Check demo session tomorrow
Backup slides
int crt = 0;
while (crt>=0 && crt<length &&
          options[crt]){
    switch(options[crt]){
    case 1:
        crt++; break;
    case 2://MSS
    case 3://WINDOW SCALE
    case 4://SACK PERMITTED
    case 8://TIMESTAMP
        crt += options[crt+1]; break;
    default:
        //unknown options, scrub
        int len = options[crt+1];
        for (i=crt;i<crt+len;i++)
            options[i] = 1;
        crt += len; break;
    }
}
TCP options parsing

```c
int crt = 0;
while (crt>=0 && crt<length &&
       options[crt]){
    switch(options[crt]){
    case 1:
      crt++; break;
    case 2://MSS
    case 3://WINDOW SCALE
    case 4://SACK PERMITTED
    case 8://TIMESTAMP
      crt += options[crt+1]; break;
    default:
      //unknown options, scrub
      int len = options[crt+1];
      for (i=crt;i<crt+len;i++)
        options[i] = 1;
      crt += len; break;
    }
  }
```
TCP Options parsing

Leave the TCP options header outside of symbolic execution

Model TCP options as metadata instead

“OPT-x” models the presence of option x
“SZ-x” size of the option in bytes
“DATA-x” value of the option

Assign("OPT30",ConstantValue(0)),
If (Constrain(TcpDst==80),Assign("OPT4",0),NoOp),
Assign("OPT2",1),
Assign("SIZE2",4),
If (Constrain("VAL2">1380),Assign("VAL2",1380),NoOp)
Does Symnet scale?

**Symbolic execution of a core router**

<table>
<thead>
<tr>
<th>Prefixes</th>
<th>Basic</th>
<th>Ingress</th>
<th>Egress</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>25s</td>
<td>2.1s</td>
<td>0.4s</td>
</tr>
<tr>
<td>62500</td>
<td>DNF</td>
<td>23.1s</td>
<td>5.6s</td>
</tr>
<tr>
<td>188500</td>
<td>DNF</td>
<td>DNF</td>
<td>18s</td>
</tr>
</tbody>
</table>
Running Klee for options parsing

<table>
<thead>
<tr>
<th>Length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of paths</td>
<td>4</td>
<td>67</td>
<td>140</td>
<td>464</td>
<td>1095</td>
<td>3081</td>
</tr>
<tr>
<td>Runtime (s)</td>
<td>0.3</td>
<td>8</td>
<td>20</td>
<td>420</td>
<td>1500</td>
<td>9120</td>
</tr>
</tbody>
</table>