ATTAIN: An Attack Injection Framework for Software-Defined Networking

Benjamin E. Ujcich, Uttam Thakore, and William H. Sanders
University of Illinois at Urbana–Champaign
Why Software-Defined Networks?

- **Decoupled** control and data planes
- **Logically centralized** decision-making
- **Programmable and extendable** applications

SDN Popularity: The Good

SDN and NFV Market Ecosystem: $18B Revenue by 2020 Forecast With Investments Growing at 46% CAGR

By
Published: May 9, 2016 8:30 a.m. ET

PUNE, India, May 09, 2016 (PR Newswire Europe via COMTEX) -- PUNE, India, May 9, 2016 /PRNewswire/ --

The market report "SDN, NFV & Network Virtualization Ecosystem: 2016 - 2030 - Opportunities, Challenges, Strategies & Forecasts" presents an in-depth assessment of the SDN, NFV and network virtualization ecosystem including

SDN poised to grow significantly over next few years
SDN Popularity: The Bad (i.e., Opportunities)

We still need to understand the security ramifications
Selected SDN Security Issues

1. Network state queries or changes dependent on OpenFlow
2. Controller behavior not well-defined
3. Lack of systematic testing for controller variations in implementation
“If we assume an attacker is attacking my network, how will each one respond?”

Simple learning switch implementations vary widely!
Prior Work and Limitations

- “DELTA: A Security Assessment Framework for Software-Defined Networks” by Lee et al. in NDSS ‘17
  - Fuzzing approach for penetration testing
  - Assumes well-defined protocol state transitions
  - Requires instrumenting agents on controllers

- OFTest project by Big Switch Networks
  - Tests switch validation for OpenFlow compliance

- “AJECT…” by Neves et al. in DSN ’06 and Antunes et al. in IEEE Trans. on Software Engineering ‘10
Our Contribution: ATTAIN

- **ATTACK Injection in Software–Defined Networks**
- Common framework for writing **reusable attack descriptions** and running attacks (for OpenFlow SDNs)
Part 1: Attack Model

ATTAIN Framework

- Practitioner (e.g., system administrator, developer, operator)
  - System model
  - Attacker capabilities
  - Attack descriptions
  - Network infrastructure

- Results
  - Log files, analysis algorithms, etc.

- Attack Model
  - Constrains

- Attack Language
  - Implemented by

- Attack Injector
Part 1: Attack Model

- Capture practitioner’s system attributes and attacker assumptions
  - System model
    - Data plane entities
    - Control plane connections
  - Attacker capabilities
    - Assumes attacker already has capabilities, not how they got them

<table>
<thead>
<tr>
<th>Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROPMESSAGE(msg)</td>
</tr>
<tr>
<td>PASSMESSAGE(msg)</td>
</tr>
<tr>
<td>DELAYMESSAGE(msg)</td>
</tr>
<tr>
<td>DUPLICATEMESSAGE(msg)</td>
</tr>
<tr>
<td>READMESSAGEMETADATA(msg)</td>
</tr>
<tr>
<td>MODIFYMESSAGEMETADATA(msg)</td>
</tr>
<tr>
<td>FUZZMESSAGE(msg)</td>
</tr>
<tr>
<td>READMESSAGE(msg)</td>
</tr>
<tr>
<td>MODIFYMESSAGE(msg)</td>
</tr>
<tr>
<td>INJECTNEWMESSAGE(msg)</td>
</tr>
</tbody>
</table>
Part 2: Attack Language

ATTAIN Framework

Attacker capabilities

Network infrastructure

Attack descriptions

System model

Practitioner
(e.g., system administrator, developer, operator)

Results
Log files, analysis algorithms, etc.

implemented by

Attack Injector

ATTAIN Framework

constraints

Attack Language

Attack Model
Part 2: Attack Language

- Description language for attacks
- Set of states ($\sigma$)
  - Start, absorbing, end states
- Set of rules ($\varphi$)
  - Conditions ($\lambda$)
  - Actions ($\alpha$)
- Represented as an attack state graph

$\sigma_1 : \sigma_1 = \{\phi_1\}$ ($\sigma_{\text{start}} = \sigma_1$
\[
\begin{align*}
\phi_1 &= (n_1, \gamma_1, \lambda_1, \alpha_1) \\
n_1 &= (c_1, s_1) \\
\gamma_1 &= \Gamma_{\text{unencrypted}} \\
\lambda_1 &= \text{READMESSAGE}(\text{msg}, \text{MESSAGETYPE} = \text{PACKET}_{-\text{IN}}) \\
\alpha_1 &= \{\alpha_{11}, \alpha_{12}\} \\
\alpha_{12} &= \text{GOToSTATE}(\sigma_2)
\end{align*}
\]

$\sigma_2 : \sigma_2 = \{\phi_2\}$
\[
\begin{align*}
\phi_2 &= (n_2, \gamma_2, \lambda_2, \alpha_2) \\
n_2 &= (c_1, s_1) \\
\gamma_2 &= \Gamma_{\text{unencrypted}} \\
\lambda_2 &= \text{READMESSAGE}(\text{msg}, \text{MESSAGETYPE} = \text{PACKET}_{-\text{OUT}}) \\
\alpha_2 &= \{\alpha_{21}, \alpha_{22}\} \\
\alpha_{22} &= \text{PASSMESSAGE}(\text{msg})
\end{align*}
\]

$\sigma_3 : \sigma_3 = \{\phi_3\}$
\[
\begin{align*}
\phi_3 &= (n_3, \gamma_3, \lambda_3, \alpha_3) \\
n_3 &= (c_1, s_1) \\
\gamma_3 &= \Gamma_{\text{unencrypted}} \\
\lambda_3 &= \text{READMESSAGE}(\text{msg}, \text{MESSAGETYPE} = \text{FLOW}_{-\text{MOD}}) \\
\alpha_3 &= \{\alpha_{31}, \ldots\}
\end{align*}
\]

(a) Attack states that model the memory of previously seen \text{PACKET}_{-\text{IN}} and \text{PACKET}_{-\text{OUT}} messages.

(b) Attack state graph representation that models the memory of previously seen \text{PACKET}_{-\text{IN}} and \text{PACKET}_{-\text{OUT}} messages.
Part 2: Attack Language Example

State $\sigma_1$

Rule $\phi_1$

Control plane connection $n_1$

Attacker capabilities $\gamma_1$

Condition $\lambda_1$

Actions $\alpha_1$

$\sigma_1 \rightarrow \sigma_1 = \{\phi_1\} \quad (\sigma_{\text{start}} = \sigma_1)$

$\phi_1 = (n_1, \gamma_1, \lambda_1, \alpha_1)$

$n_1 = (c_1, s_1)$

$\gamma_1 = \Gamma_{\text{unencrypted}}$

$\lambda_1 = \text{READMESSAGE}(msg, \text{MESSAGETYPE} = \text{PACKET_IN})$

$\alpha_1 = \{\alpha_{11}, \alpha_{12}\}$

$\alpha_{11} = \text{PASSMESSAGE}(msg)$

$\alpha_{12} = \text{GOTOSTATE}(\sigma_2)$
\begin{align*}
\sigma_2 : \sigma_2 &= \{ \phi_2 \} \\
\phi_2 &= (n_2, \gamma_2, \lambda_2, \alpha_2) \\
n_2 &= (c_1, s_1) \\
\gamma_2 &= \Gamma_{unencrypted} \\
\lambda_2 &= \text{READMESSAGE}(msg, \text{MESSAGETYPE} = \text{PACKET\_OUT}) \\
\alpha_2 &= \{ \alpha_{21}, \alpha_{22} \} \\
\alpha_{21} &= \text{PASSTMESSAGE}(msg) \\
\alpha_{22} &= \text{GOTOSTATE}(\sigma_3)
\end{align*}
Part 2: Attack Language Example

\[ \sigma_3 : \sigma_3 = \{ \phi_3 \} \]
\[ \phi_3 = (n_3, \gamma_3, \lambda_3, \alpha_3) \]
\[ n_3 = (c_1, s_1) \]
\[ \gamma_3 = \Gamma_{unencrypted} \]
\[ \lambda_3 = \text{READMESSAGE}(msg, \text{MESSAGETYPE} = \text{FLOW_MOD}) \]
\[ \alpha_3 = \{ \alpha_{31}, \ldots \} \]
\[ \alpha_{31} = \ldots \]
Part 3: Attack Injector

Practitioner
(e.g., system administrator, developer, operator)

System model
Attacker capabilities
Attack descriptions
Network infrastructure

Results
Log files, analysis algorithms, etc.

ATTAIN Framework

Attack Model

Attack Language

implemented by

Attack Injector
Part 3: Attack Injector

- Implement attacks during runtime
- Proxy control plane connections
- Attack description can call monitors to perform actions or collect data during testing
Evaluation Setup

- **External-facing network switch**
- **DMZ firewall switch**
- **Local internal network switches**
- **Internet**
- **SDN controller**
- **Internal networks**
- **User workstations**

**Control plane connection**: Blue lines
**Data plane network link**: Gray lines
**Logical network boundary**: Dashed lines

**Network Components**
- **h1**
- **h2**
- **h3**
- **h4**
- **h5**
- **h6**
Evaluation: Flow Modification Suppression

Evaluation: Flow Modification Suppression

What are the effects in the control and data planes? (Throughput, latency, etc.)

Evaluation: Flow Modification Suppression

\[ \sigma_1 : \sigma_1 = \{\phi_1, \phi_2, \phi_3, \phi_4\} \text{ (} \sigma_{\text{start}} = \sigma_1; \sigma_{\text{absorbing}} = \{\sigma_1\}; \sigma_{\text{end}} = \emptyset \text{)} \]

\[ \phi_1 = (n_1, \gamma_1, \lambda_1, \alpha_1) \]
\[ n_1 = (c_1, s_1) \]
\[ \gamma_1 = \Gamma_{\text{NoTLS}} \]
\[ \lambda_1 = \text{READMESSAGEMETADATA}(msg, \text{MESSAGESOURCE} = c_1) \]
\[ \wedge \text{READMESSAGEMETADATA}(msg, \text{MESSAGEDESTINATION} = s_1) \]
\[ \wedge \text{READMESSAGE}(msg, \text{MESSAGETYPE} = \text{FLOW.MOD}) \]
\[ \wedge \text{READMESSAGE}(msg, \text{MESSAGETYPEOPTIONS.command} = \text{ADD}) \]
\[ \alpha_1 = \{\alpha_{11}\} \]
\[ \alpha_{11} = \text{DROPMESSAGE}(msg) \]
\[ \phi_2 = (n_2, \gamma_2, \lambda_2, \alpha_2) \]
\[ n_2 = (c_1, s_2) \]
\[ \gamma_2 = \Gamma_{\text{NoTLS}} \]
\[ \lambda_2 = \text{READMESSAGEMETADATA}(msg, \text{MESSAGESOURCE} = c_1) \]
\[ \wedge \text{READMESSAGEMETADATA}(msg, \text{MESSAGEDESTINATION} = s_2) \]

Flow modification suppression attack description (abbreviated)
1 state, 4 rules
Results: End-to-End Throughput

* No connection (denial of service)
Results: End-to-End Latency

* No connection (denial of service)
Results: Control Plane Messages
Conclusion

- Attack injection framework for testing SDN implementations
- Evaluated Floodlight, Ryu, POX controllers in virtualized network environment

Practical lessons:
- Even simple functionality can be implemented very differently!
- Data plane activity can affect control plane
Questions?

- Thanks for listening!
- For more information:
  - **M.S. Thesis:** "An Attack Model, Language, and Injector for the Control Plane of Software-defined Networks"
    University of Illinois at Urbana-Champaign, July 2016.
  - **Contact:** Ben Ujcich, ujcich2@illinois.edu