Exploiting Intent-Flow State Vulnerabilities in Intent-Based Networking

Angela Yan
Purdue University
yan431@purdue.edu

Jiwon Kim
Purdue University
kim1685@purdue.edu

Benjamin E. Ujcich
Georgetown University
bu31@georgetown.edu

Dave (Jing) Tian
Purdue University
daveti@purdue.edu

***Abstract*—Intent-based networking (IBN) is a high-level network configuration concept that allows network operators to specify "what" to do instead of "how". IBN translates high-level intents into low-level flow rules, installs flow rules into network devices, and verifies whether these flow rules satisfy intents. However, during the reconciliation process in IBN, inconsistencies between intent and flow-rule status can occur due to IBN bugs. A malicious attacker can exploit such inconsistencies to violate the integrity of benign intent or cause a *denial of service*. A network operator will be unaware of these attacks occurring due to the intent state failing to update based on its flow rule status. We call the misalignment of intent and flow an *intent-flow* discrepancy. We found that both control and data plane updates can cause intent-flow discrepancies in IBN.**

In this paper, we investigate intent-flow discrepancies as a vulnerability in ONOS, a representative IBN implementation. We propose two attacks called *max-flow* and *flow-tamper*, which we demonstrate in ONOS. We propose three mitigation strategies and future research plans to defend against intent-flow attacks.

***Index Terms*—Intent-based networking, software-defined networking, vulnerability analysis, denial-of-service**

I. Introduction

Software-defined networking (SDN) has changed the network paradigm to program network semantics from the centralized control plane instead of manual configurations on data plane devices. More recently, *intent-based networking (IBN)* has further simplified the process of achieving network semantics by allowing network administrators to specify what they want to accomplish rather than how they implement their goals on the network.

An IBN receives "intents" from network operators as various inputs such as configurations [29], [36] or LLM prompts [17], [34]. IBN realizes the intents through the intent life cycle, where the IBN translates the operator's intent into a set of low-level flow rules which are then installed on the devices and govern forwarding behavior. The IBN then verifies whether the intent is properly implemented by monitoring the network. During the life cycle, the intent's "state" is reported to the operator, which can be changed by topology changes (*e.g.,* a downed link) or by operator input (*e.g.,* withdrawal).

Updates in both the data plane and control plane can affect an IBN. While the data plane has limited resources (*e.g.,* bandwidth, device memory availability, device compute capacity), network operators can ask for more capabilities that can be handled through high-level intents in the control plane. The separation of high-level intents and low-level flow rules in the intent life cycle can result in challenges in maintaining their cohesion. An intent's flow rules can conflict with existing ones managed by the control plane (*e.g.,* other intents' rules or flow rule API). When IBN fails to reconcile the updates in both planes, the intent state and its compiled flow rules' states may show conflicting information. We refer to this as an *intent-flow* discrepancy.

Intent-flow discrepancies can serve as attack vectors for malicious attackers. In the data plane, attackers can reduce the overall network resources by consuming a portion of them (*e.g.,* flooding) or downing a link (*e.g.,* link-flooding attack). In the control plane, attackers can affect the flow rules of existing intents by making a flow-rule API call. When such attacks cause an intent-flow discrepancy, an IBN can mislead network operators into believing that their intents are correctly implemented, potentially leading to a *denial of service*.

Despite standardization efforts [1]–[3], [16], [27], industry uses [11], [12], [14], [15], [18], and open-source projects [29], [34], [36], there is relatively little work on understanding IBN security. Of recent work on IBN security [23], [38], [43], none have identified attacks that exploit intent-flow discrepancies. To the best of our knowledge, no work considers *permanent* discrepancies in intent and flow rule state in IBN.

In this paper, we investigate possible attacks from these discrepancies that exploit data/control plane differences in which the intent's owner (*e.g.,* network operator) is blind to these attacks. We propose two categories of attacks that use intent-flow discrepancies that can be leveraged by an adversary looking to attack the integrity of benign intents, referred to as *max-flow* and *flow-tamper*. In *max-flow*, an attacker exploits a switch's physical limit to drop flow rules. In *flow-tamper*, an attacker can remove a flow to cause intent failure. In both of these intent-flow attacks, the network operator is unaware that the attack is taking place, due to the intent state failing to reflect the current state of its underlying flow rules.

We demonstrate the exploitive nature and security impacts of these attacks on the Open Network Operating System (ONOS) [29] as a representative case study of intent-flow discrepancies on both the data and control planes. We propose several mitigation strategies against intent-flow attacks, including a monitoring process that continuously checks for intent-flow inconsistencies, a policy that formally connects intent and flow state more thoroughly, and a policy for updating intent state after a timer tracking flow configuration failure. We

explore future areas of research on the importance of syncing the data and control plane from both dimensions.

Contributions. Our contributions are as follows:

- We explore discrepancies between the intent state and the flow rule state.
- We introduce two *intent-flow* attacks: *max-flow*, which takes advantage of a switch's flow limits, and *flow-tamper*, which removes flow rules. In both attacks, due to the intent-flow state misalignment, the operator is unaware of the attack.
- We propose three mitigation strategies for intent-flow attacks: monitoring, connection policy, and policy enforcement. We discuss future research to sync intent-flow updates.

Vulnerability disclosure. We identified a vulnerability in the ONOS SDN controller. We responsibly disclosed our discovery to the ONOS and Aether Project teams.

II. Background

A. Intent-Based Networking (IBN)

In traditional networking, an operator has to configure devices and switches by interacting physically with the hardware, a complex and non-modular process. The development of *software-defined networking (SDN)* separated the network's logic from its traffic into two respective planes: the *control plane* and the *data plane*. *Intent-based networking (IBN)* furthered this semantic gap by syntactically simplifying the control plane such that operators only needed to specify *what* they wanted the network to do instead of *how* they wanted the network to do it. Operators would only need to specify an *intent*, *e.g.,* "connect host A to host B", which would then be compiled into a set of underlying flow rules by the IBN.

Across different intent implementations, the life cycle of an intent can generally be organized into five possible stages [24].

1) *Intent profiling*: The intent is created by the operator in a syntactically simple way. Some examples of input could be using a template, *e.g.,* a command, natural language processing, a graphical interface, etc.
2) *Intent translation*: The created intent is translated into low-level system-friendly instructions, such as a set of flow rules.
3) *Intent resolution*: If the submitted intent results in conflicts, a resolution policy of the IBN will handle and resolve the conflicting intents to the best of its ability.
4) *Intent activation*: Once the IBN is sure the intent can be installed correctly, the compiled instructions from Intent Translation are implemented.
5) *Intent assurance*: The IBN continuously monitors the network to ensure the intent is satisfied through shifts in its environment, such as network topology changes.

Implementations. IBN is currently being developed in several ways, including research in academia, proprietary builds, and open-source projects. Leivadeas *et al.* [24], Kim *et al.* [21], and Ahmad *et al.* [6] all cover the general framework and theoretical challenges of an IBN implementation. Proprietary IBSs are being developed by IBM [15], Huawei [14], Juniper [18], Google [12], and Cisco [11]. IBN has open-source implementations in the Open Network Operating System (ONOS) [29], OpenDaylight (ODL) [36], and the Open Network Automation Platform (ONAP) [34].

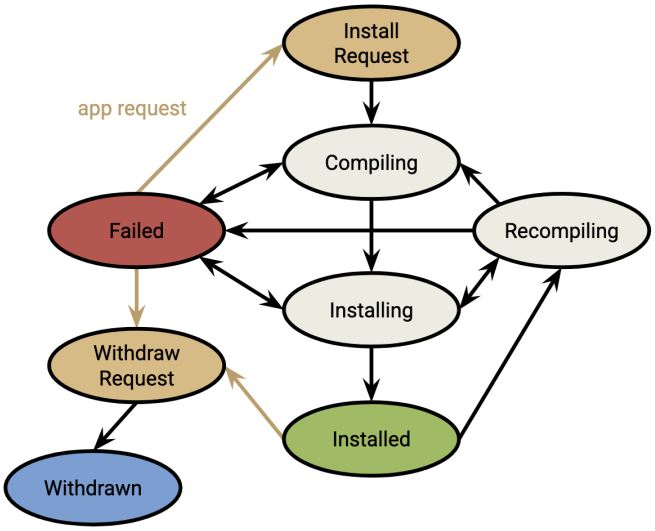

Fig. 1. Various intent states of a single intent in ONOS IBN [26]. *COMPILING, RECOMPILING, INSTALLING* are *transitional*, brief states. *INSTALLED, FAILED, WITHDRAWN* are *parking* states where the intent will remain in this state for most of its time. Golden states and arrows *INSTALL REQUEST, WITHDRAW REQUEST* are input events by an IBN app/operator. Intent states are commonly changing; network events can cause a successfully installed intent to recompile.

B. Open Network Operating System (ONOS)

ONOS is an open-source IBN implementation developed by the Open Networking Foundation [29]. We will use ONOS as the primary IBN implementation due to its high quality, large-scale use, and open source code. In ONOS, the process of creating an intent and its subsequent installation onto hardware is similar to other IBN specifications: (1) the intent is created by an operator in ONOS using its CLI/GUI/API, (2) the intent is compiled into a set of flow rules, (3) the flow rules are installed onto the switch depending on the network logic set by the intent.

An intent's *state* will cycle through various descriptions of its current status. From an operator requesting an intent to be installed to when the intent is no longer needed and withdrawn, Figure 1 shows how the various states of an intent transition to and from one another. A simple, successful intent installation will pass through the following states: `INSTALL_REQ`, `COMPILING`, `INSTALLING`, `INSTALLED`. When the operator does not need the intent anymore, an intent withdrawal will pass through the states: `WITHDRAW_REQ`, `WITHDRAWING`, `WITHDRAWN`. An intent's state can be changed either actively from operator requests or reactively from network events. If a device in an intent's compiled path goes down, an `INSTALLED` intent may go through

RECOMPILING to determine whether it fails (FAILED) or successfully installs (INSTALLED) again.

While compiling an intent down to its flow rules is straightforward in concept, in practice, there are many complexities during and after intent installation that need to be considered. Topology changes, apps with specific permission models, and flow rule modifications are all possible network events that can impact an intent and its flow rules. In our experiments with intent-flow compilation to study whether current IBN implementations successfully adapt to these events, we found vulnerabilities in both the data and control plane from intent and flow-rule implementation. We will show these vulnerabilities below by running two concrete examples on ONOS as a representative IBN system.

III. Threat Model

We first explain our threat model, including our assumptions for the attacker and the attacker's goal. We provide two attacker models based on where the attacker is located: (1) the control plane, and (2) the data plane.

A. Assumptions

Devices. The devices we refer to in our threat model are primarily hosts and switches. *Hosts* represent end-point devices such as computers, servers, and databases that can send and receive traffic with other hosts. *Switches* are devices that forward the traffic between hosts based on flow rules installed by an IBN controller. We assume that switches have a restricted memory with a fixed size of flow rules (*e.g.,* 8192 in Pica8 OpenFlow hardware switch [37]).

Intents. We assume that the network is controlled and managed by an operator using intents, which are compiled into flow rules that direct network traffic to their specified locations. Furthermore, we assume that all intents are compiled correctly and an intent's initially generated flow rules enforce the operator's intended design.

Controller. We assume that the IBN controller manages the IBN-based network with intents. The IBN controller may be built upon an SDN controller, allowing toolkits for malicious attacks [43]. We assume that the IBN controller does not restrict applications using the flow-rule API. In addition, applications running on a general-purpose machine can install flow rules to execute fine-grained packet modifications beyond the capability of hardware forwarding devices (*e.g.,* IDS, IPS).

B. Attacker Model

We consider two different attacker models in IBN: a control plane attacker in a malicious application and a data plane attacker in a remote host. Aside from malicious entities, other devices and apps in the model are benign. We assume a default-deny model, where hosts cannot access other hosts unless explicitly granted permission. An attacker's goal is to violate the integrity of existing or newly created intents by benign network operators.

Control plane attacker. The attacker exists on the control plane by commanding a malicious application on the IBN-controlled network, as previously assumed in other works [10], [22], [39], [40]. This position can be achieved by injecting an attacker-controlled software app onto the system — the attacker could exploit a vulnerable open-source third-party app using supply-chain attacks. The IBN controller has role-based access control to restrict applications' permissions [28], [45]. We assume that the malicious application only has FLOW_WRITE permissions, so it can only manage the flow rules it installs.

Data plane attacker. The attacker exists on the data plane as a remote host connected to an IBN-managed network. The malicious host can fingerprint an SDN controller [31], SDN applications [9], and flow rules [4]. The attacker can leverage the information to determine if the network is managed by the IBN controller [43]. The attacker can exploit reactive SDN apps such as fwd to create an excessive number of flow rules [8]. The malicious attacker can flood a victim device with packets from multiple hosts to execute a link-flooding attack [19].

IV. Intent-Flow Discrepancy Attacks

Based on our assumptions and attack model from Section III, we show how an adversary can exploit *intent-flow* discrepancies to attack benign intents and execute DoS attacks on an IBS. We demonstrate preliminary experiments of intent-flow attacks on both the control and data plane: we present two categories of IFD attacks *max-flow* and *flow-tamper*, then run them on ONOS. We carry out a case study on ONOS to investigate how intents are compiled into flow rules.

A. Intent-Flow Attacks

IBN realizes high-level network objectives for low-level network implementations. To fulfill network objectives, IBN translates intents into network configurations (*e.g.,* flow rules) and installs them. To ensure network objectives, IBN continuously verifies that the configurations satisfy intents. However, during such reconciliation processes in IBN, the discrepancy between the two different abstraction layers can occur due to bugs in IBN. Moreover, the attacker can exploit such discrepancies to violate the integrity of existing intents installed by benign users.

Whenever the status of an intent incorrectly describes the behavior of the underlying flow configuration, and an adversary exploits this to violate network and user integrity, we refer to this as an *intent-flow attack*. As such, an intent operator accessing intent status across an IBN can be misled into believing that a network is operating without fault through incorrect intent status, even if the underlying flow rules prove otherwise; an attacker can purposefully trigger this state.

Assume an attacker has control of a malicious Host M, with specifications as described in Section III. Host M executes intent-flow attacks, aiming to disrupt benign existing intents. Because the attack modifies low-level configurations, which the IBN does not reflect, the network operator is blind to the attack.

We describe three possible intent-flow attacks to execute: two examples of *max-flow attacks* and one *flow-tamper attack*.

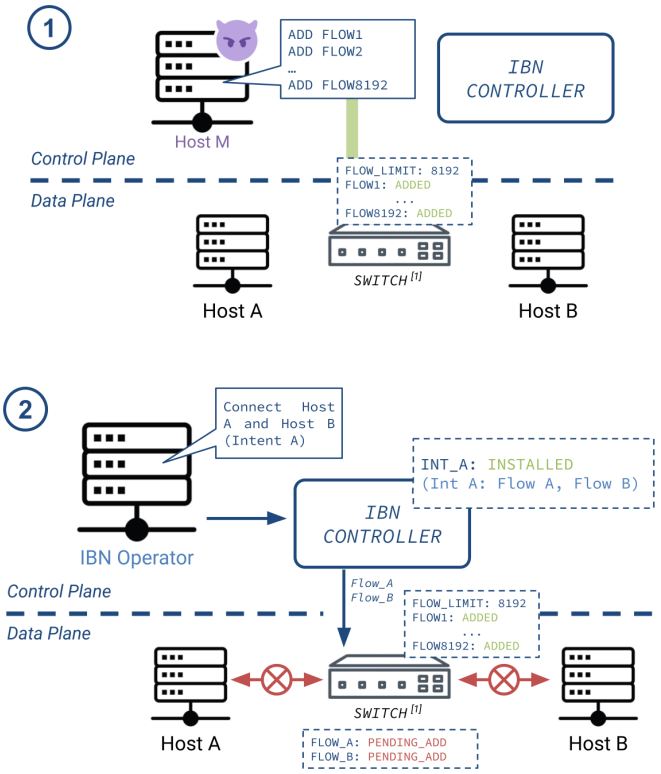

Fig. 2. *Max_flow attack.* (1) Host A and Host B exist in an IBN. An adversary with malicious Host M floods a switch with `FLOW_LIMIT = 8192`, and all intents and flows are successfully installed. (2) The Intent Operator tries to connect Host A and Host B with an intent A. Since Host M has already flooded the switch, the compiled flow rules report `PENDING_ADD`. However, the IBN reports intent A as installed. Therefore, the operator is unaware that the intent has failed.

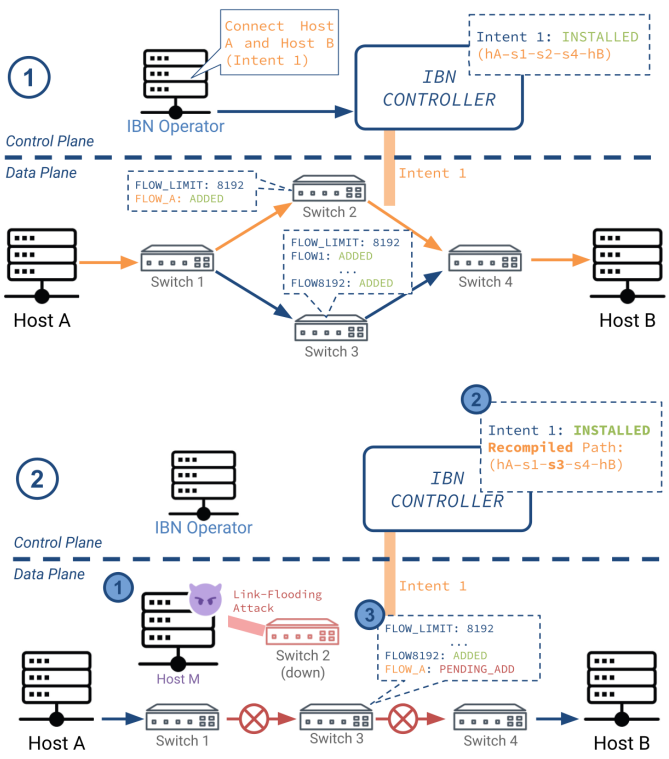

Fig. 3. *Max_flow attack.* (1) Host A and Host B are connected via Intent A, which connects Host A and B through S1-S2-S4. S3 is at its `FLOW_LIMIT`. (2.1) An adversary with malicious Host M floods S2 with packets, causing S2 to go down. (2.2) The IBN recompiles Intent A to go through S3, causing S2's previous flow rules (`FLOW_A`) to fall onto S3. (2.3) Intent A still reports `INSTALLED`, but `FLOW_A` is unable to be `ADDED`, causing Host A and B to lose connectivity.

We reproduce these attacks on an emulated network to show how they could violate the integrity of benign intents. While we use ONOS IBN to demonstrate these attacks, the attacks generalize to any IBN implementation, e.g., OpenDaylight (ODL), as they exploit a common IBN architecture. Since ODL uses an intent/flow API and translates intents into low-level flow rules, it is also vulnerable to these attacks.

Environmental setup. We used a Docker image of the latest ONOS version on a MacBook Pro with an Apple M1 Pro Chip and 16 GB of memory. For our simulated network, we used mininet [25] emulated with UTM Virtual Machine for Mac [42]. We activated two ONOS apps: (1) `org.onosproject.openflow` for the OpenFlow protocol and (2) `proxyarp` to specify IP addresses and ports. On UTM, we used mininet to create Open vSwitch (OvS) topologies. To modify intents, we used ONOS' REST API, CLI, and GUI. To modify at the flow level, we used mininet's CLI and Open vSwitch's configuration database [35] to simulate a physical max-limit of a switch and other switch characteristics. We assume a physical switch's max-limit is 8192 based on a standard pica8 switch [37]).

B. Max-Flow Attacks

This category of attack exploits the intent and flow rule discrepancy by overflowing the maximum physical flow limit of a switch [30]. The attacker can exploit *max-flow* attacks to cause a denial-of-service (DoS). We categorize these two examples based on the *method* of saturating a switch: (1) flow flooding and (2) induced overflow.

(1) Flow Flooding. Figure 2 shows an example of overflow using flow rules. An attacker saturates a switch by simply installing flow rules. Once a switch is flooded, an operator submits an intent to connect benign hosts A and B, which compiles into a set of flow rules to be assigned to switch S. Since switch S has hit its switch limit, the flow rules end up in the `PENDING_ADD` state. However, the intent state reports `INSTALLED`. Host A and Host B cannot connect to each other due to the incomplete intent, yet the operator only sees the `INSTALLED` intent. In this attack, the attacker uses the *Control plane attacker* model.

(2) Induced Overflow via Link-Flooding. Figure 3 shows an example of overflow via link-flooding. An attacker without permission removes a link using a link-flooding attack [19]; this link could be hosting an intent's flow rules connecting benign hosts. The removal of this link can cause the link's configured flow rules to fall onto another switch, which over-

flows the switch's limit. While the overflowed flow rules are in the PENDING_ADD state, the intent state stays INSTALLED. Host A and Host B cannot connect to each other due to the incomplete intent, yet the operator only sees the INSTALLED intent. In this attack, the attacker uses the *Data plane attacker* model.

ONOS Demonstration.

Flow Flooding. Using mininet on UTM, we set up a simple topology of two hosts connected by a switch. We used ovs-vsctl to create a Flow_Table with a flow_limit to simulate a max switch limit of five. Using ONOS REST API, we submitted five flow rules onto the switch to simulate a saturated device. Then, we created a Host-to-Host intent to connect the two hosts. We dumped the switch's installed flow rules with ovs-vsctl and found that the flow rule compiled from the intent had the state PENDING_ADD. We used mininet to attempt to ping one host from the other and found that all packets were dropped. On the ONOS CLI, the intent status incorrectly reported as INSTALLED.

Induced Overflow. Using mininet on UTM, we set up a topology of two hosts (referred to as H1 and H2) and four switches (S1, S2, S3, S4), with links (H1, S1), (S1, S2), (S2, S4), (S1, S3), (S3, S4), (S4, H2); this resembles the topology in Figure 3.1. Using ONOS CLI, we submitted one Host-to-Host intent connecting H1 and H2, which compiled into flow path H1, S1, S2, S4, H2. We used ovs-vsctl to create a Flow_Table on S3 with a flow_limit to simulate a max flow limit of five. Then, we used mininet CLI to run the command link s3 s4 down to simulate a link dropping due to a link-flooding attack. As a result, trying to ping h2 from h1 resulted in no packets being sent. However, the intent status still remained INSTALLED.

C. Flow-Tamper Attack

This attack exploits benign, previously existing intents. Figure 4 shows how an attacker can exploit *flow-tamper* attack. An attacker with control of an operator-installed, third-party malicious App M can guess an intent connecting two hosts, Host A and Host B, by leveraging existing fingerprinting tools [4]. The attacker can then modify the flow rules, such that the existing intent's underlying flow configuration will fail. However, the intent state will still remain as INSTALLED. In this attack, the attacker uses the *Control plane attacker* model.

ONOS Demonstration. Using mininet on UTM, we set up a simple topology of two hosts connected by a switch. Then, using ONOS CLI we created a *Host-to-Host Intent* connecting the two hosts. We used ONOS' REST API to get the id of the installed flow rules and delete the flow rules. Despite the flow rules being removed from the intent entirely, the intent status still remained INSTALLED.

V. Mitigation Strategies

We discuss several possible solutions to mitigate intent-flow discrepancy attacks.

Monitoring. An IBN can continuously monitor the network for topology changes or network events, specifically checking

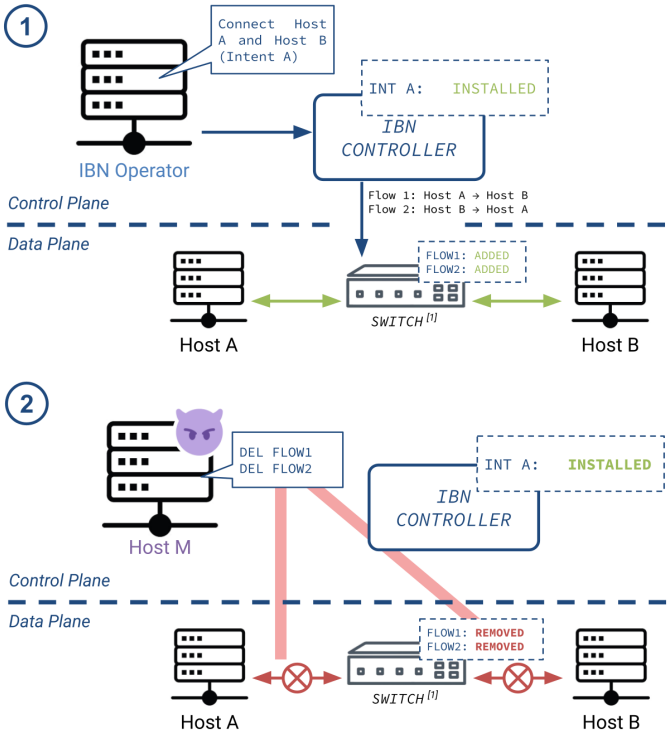

Fig. 4. *Flow-tamper attack.* (1) Host A and Host B are connected via an intent A, which has compiled into Flow 1 and Flow 2, connecting A to B and B to A, respectively. (2) An adversary controlling malicious Host M deletes Flow 1 and Flow 2. However, the IBN Controller still reports Intent A is INSTALLED.

to update intent status if a flow rule is modified according to a condition. While current IBN implementations already contain network monitoring, the intent update condition fails to update the intent status for all flow modifications. Any sort of auditor would need a small enough runtime and memory usage such that it would be worth the additional burden. A monitor would be a retroactive mitigation, where the intent-flow discrepancy would already have happened before detection.

Connection Policy. A monitor can check an IBN to assure intent-flow connection *e.g.,* an intent and flow's states constantly reflect each others states. To ensure that an intent correctly reflects flow status, a policy can be proposed that details how intents should be connected to flow rules. Specifically, what intent state describes which flow rule configuration, and which changes should be made to an intent based on flow modification, whether passively through topology changes or actively through operator input. For example, in ONOS, the state *INSTALLING* is only meant as a short transitional state; however, in *max-flow*, where flows are *PENDING_ADD*, the *INSTALLING* state could be an accurate descriptor of the status of the switch (as the intent's flow rules are waiting to be added). Therefore, a comprehensive coupling of intent state to flow state can allow IBNs to work with a standard of connection, which would mitigate intent-flow discrepancies. A connection policy would be a proactive mitigation strategy to eliminate all possible intent-flow discrepancies from the

beginning.

Policy Enforcement. IBN can further inform the operators of unavailable intents with *FAILED*, when the network is unable to support current intents. IBN can have a timeout to make the intent state as *FAILED* when its flow rules show *PENDING_ADD*. In the *max-flow* with the link-flooding attack, IBN simply attempts to install flow rules on different switches to victim switches. While reinstalling such flow rules, IBN can prioritize flow rules of intents with a higher priority. When victim switches cannot support flow rules, IBN can change the state of remaining intents as *FAILED*.

VI. Discussion and Future Work

Attacker Permissions. We assume that an adversary has malicious app access to create and modify their intents and/or flow rules. While a skilled attacker is needed, achieving prohibited access in an SDN has been researched. *Cross-App Poisoning* [39] allows an unprivileged app to puppet a privileged app. SVHunter facilitates the creation of D^2C^2 exploit chains to increase app reachability [44].

Other IBN Implementations. Though we focused on ONOS to demonstrate our attacks, we believe that other IBN controllers are also prone to intent-flow attacks. ONOS was found to be vulnerable due to control-data plane decoupling in certain scenarios. In our study of OpenDaylight, another widely used open-source IBN implementation, we found its intents were not updated by flow rule modifications either, rendering it vulnerable. For proprietary IBN implementations like IBM Cloud Pak for Network Automation [15], Huawei CloudFabric [14], Juniper Astra [18], Google Orion [12], and Cisco [11], we do not have access to their products, so we cannot test for intent-flow discrepancies.

Future Work. We discussed three mitigations for intent-flow coupling in Section V. Future work could be done in pinpointing all intent-flow discrepancy conditions that an IBN would need to monitor for, as well as the most cost-effective monitoring algorithm that detects intent-flow discrepancies. This would include studying the benefits and consequences of retroactive and proactive approaches and exploring different types of detection, such as provenance [38], fuzzing [23], etc. Further work should be done to explore the causation of intent-flow attacks, as IBN already considers flow rule event handling; pinpointing the vulnerability or bug for why intent-flow attacks can lead to complete mitigation of intent-flow discrepancies.

VII. Related Work

IBN Surveys. Theoretical IBN frameworks, implementations, policies, and security have been studied, analyzed, and categorized in multiple works [6], [21], [24], [46]. While these papers discuss general intent compilation and policy, none of them provides a detailed, low-level implementation of intents and intent compilation to address intent-flow discrepancies.

Intent-Based Tools. IBN can be used as a tool to improve the security of a network due to its semantically simple operation, allowing network operators to manage security without needing tedious, intricate, and low-level specifications. There are multiple tools that utilize the idea of intents to reinforce network security [5], [13], [20]. However, since they do not consider the security of the IBN itself, these tools do not detect intent-flow discrepancies.

IBN Security. Security within the IBN itself is a field that is currently being researched. Intender [23] is a semantically aware fuzzing framework that uses existing network topology information to create more effective fuzzing inputs. Spotlight [43] exploits temporal vulnerabilities in the flow-rule installation order to gain access to unauthorized networks. ProvIntent [38] extends SDN-provenance tools to account for intent-specific concerns such as intent semantics. While these tools detect IBN vulnerabilities, they do not account for *intent-flow discrepancies*.

SDN Security. IBNs can be implemented as a higher-level abstraction layer on SDN controllers. ONOS implements its own monitoring system in its Security Mode (SM-ONOS) [45], which introduces a necessary policy file that determines the permissions of an ONOS application and monitors the network to enforce these permissions. ProvSDN [39] uses data provenance to prevent cross-app poisoning attacks. However, since our research concerns the specific relation of intents to flows, these do not cover intent-flow vulnerabilities. The security of SDNs has been researched thoroughly [7], [22], [32], [33], [39], [41]. Since IBN is implemented as an abstraction layer of SDN, these works complement our work.

VIII. Conclusion

We discovered a vulnerability in current IBN implementations due to a discrepancy between intent and flow state in specific scenarios. We proposed two attacks: *max-flow* and *flow-tamper*. We exploited a vulnerability in IBN intent-flow status coupling, which an adversary can exploit to override and modify existing benign intents. We proposed three mitigation strategies and discussed future work for defending against intent-flow attacks.

IX. Acknowledgments

We thank the anonymous reviewers at IEEE SecDev 2025 for their invaluable comments and helpful suggestions. This material is based upon work supported by the National Science Foundation under Grant No. CNS-2339882.

References

[1] 3GPP. Telecommunication management; study on scenarios for intent driven management services for mobile networks. https://www.3gpp.org/DynaReport/28812.htm, 2020.

[2] 3GPP. Study on enhanced intent driven management services for mobile networks. https://www.3gpp.org/DynaReport/28912.htm, 2023.

[3] 3GPP. Management and orchestration; intent driven management services for mobile networks. https://www.3gpp.org/DynaReport/28312.htm, 2025.

[4] Stefan Achleitner, Thomas La Porta, Trent Jaeger, and Patrick McDaniel. Adversarial network forensics in software defined networking. In *Proceedings of the Symposium on SDN Research*, pages 8–20, 2017.

[5] Adeola Adewa, Vincent Anyah, Omoniyi David Olufemi, Adedeji Ojo Oladejo, and Toluwanimi Olaifa. The impact of intent-based networking on network configuration management and security. *Global Journal of Engineering and Technology Advances*, 22(01):063–068, 2025.

[6] Ijaz Ahmad, Jere Malinen, Filippos Christou, Pawani Porambage, Andreas Kirstädter, and Jani Suomalainen. Security in intent-based networking: Challenges and solutions. In *2023 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 296–301. IEEE, 2023.

[7] Jay Barach. Towards zero trust security in sdn: A multi-layered defense strategy. In *Proceedings of the 26th International Conference on Distributed Computing and Networking*, pages 331–339, 2025.

[8] Jiahao Cao, Mingwei Xu, Qi Li, Kun Sun, Yuan Yang, and Jing Zheng. Disrupting sdn via the data plane: A low-rate flow table overflow attack. In *International Conference on Security and Privacy in Communication Systems*, pages 356–376. Springer, 2017.

[9] Jiahao Cao, Zijie Yang, Kun Sun, Qi Li, Mingwei Xu, and Peiyi Han. Fingerprinting {SDN} applications via encrypted control traffic. In *22nd international symposium on research in attacks, intrusions and defenses (RAID 2019)*, pages 501–515, 2019.

[10] Mingming Chen, Thomas La Porta, Teryl Taylor, Frederico Araujo, and Trent Jaeger. Manipulating openflow link discovery packet forwarding for topology poisoning. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 3704–3718, 2024.

[11] Cisco. Intent-based networking (ibn). https://www.cisco.com/site/us/en/solutions/intent-based-networking/index.html, 2025.

[12] Andrew D Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, et al. Orion: Google's {Software-Defined} networking control plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 83–98, 2021.

[13] Nicolas Herbaut, Camilo Correa, Jacques Robin, and Raul Mazo. Sdn intent-based conformance checking: application to security policies. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pages 181–185. IEEE, 2021.

[14] Huawei. Huawei launches the intent-driven networking for cloud-fabric solution. https://www.huawei.com/en/news/2018/6/Intent-Driven-Networking-CloudFabric-Solution, 2018.

[15] IBM. Ibm brings ai-powered automation software to networking to help simplify broad adoption of 5g. https://newsroom.ibm.com/2021-06-28-IBM-Brings-AI-Powered-Automation-Software-to-Networking-Helps-Simplify-Broad-Adoption-of-5G, 2021.

[16] Internet Research Task Force (IRTF. Intent classification. https://www.rfc-editor.org/rfc/rfc9316.pdf, 2022.

[17] Arthur S Jacobs, Ricardo J Pfitscher, Rafael H Ribeiro, Ronaldo A Ferreira, Lisandro Z Granville, Walter Willinger, and Sanjay G Rao. Hey, lumi! using natural language for {intent-based} network management. In *2021 usenix annual technical conference (usenix atc 21)*, pages 625–639, 2021.

[18] Juniper. Apstra intent-based networking. https://www.juniper.net/us/en/products/network-automation/apstra.html, 2025.

[19] Min Suk Kang, Soo Bum Lee, and Virgil D Gligor. The crossfire attack. In *2013 IEEE symposium on security and privacy*, pages 127–141. IEEE, 2013.

[20] Jinyong Kim, Eunsoo Kim, Jinhyuk Yang, Jaehoon Jeong, Hyoungshick Kim, Sangwon Hyun, Hyunsik Yang, Jaewook Oh, Younghan Kim, Susan Hares, et al. Ibcs: Intent-based cloud services for security applications. *IEEE Communications Magazine*, 58(4):45–51, 2020.

[21] Jiwon Kim, Hamed Okhravi, Dave Tian, and Benjamin E Ujcich. Security challenges of intent-based networking. *Communications of the ACM*, 67(7):56–65, 2024.

[22] Jiwon Kim, Dave Jing Tian, and Benjamin E Ujcich. Chimera: Fuzzing p4 network infrastructure for multi-plane bug detection and vulnerability discovery. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 2865–2883. IEEE Computer Society, 2025.

[23] Jiwon Kim, Benjamin E Ujcich, and Dave Jing Tian. Intender: Fuzzing {Intent-Based} networking with {Intent-State} transition guidance. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4463–4480, 2023.

[24] Aris Leivadeas and Matthias Falkner. A survey on intent-based networking. *IEEE Communications Surveys & Tutorials*, 25(1):625–655, 2022.

[25] Mininet Project. Mininet. https://mininet.org/, 2025.

[26] Open Networking Foundation. Intent framework. https://wiki.onosproject.org/display/ONOS/Intent+Framework, 2016.

[27] Open Networking Foundation. Intent nbi – definition and principles. https://opennetworking.org/wp-content/uploads/2014/10/TR-523_Intent_Definition_Principles.pdf, 2016.

[28] Open Networking Foundation. Onos application permissions. https://wiki.onosproject.org/display/ONOS/ONOS

[29] Open Networking Foundation. Open network operating system (onos). https://opennetworking.org/onos/, 2025.

[30] Ying Qian, Wanqing You, and Kai Qian. Openflow flow table overflow attacks and countermeasures. In *2016 European Conference on Networks and Communications (EuCNC)*, pages 205–209, 2016.

[31] Seungwon Shin and Guofei Gu. Attacking software-defined networks: a first feasibility study. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, page 165–166, New York, NY, USA, 2013. Association for Computing Machinery.

[32] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424, 2013.

[33] Richard Skowyra, Lei Xu, Guofei Gu, Veer Dedhia, Thomas Hobson, Hamed Okhravi, and James Landry. Effective topology tampering attacks and defenses in software-defined networks. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 374–385. IEEE, 2018.

[34] The Linux Foundation. Open network automation platform. https://docs.onap.org/en/newdelhi/, 2025.

[35] The Linux Foundation. Open vswitch. https://www.openvswitch.org/, 2025.

[36] The Linux Foundation. Opendaylight. https://www.opendaylight.org/, 2025.

[37] Tracy Yang. Flow scalability per broadcom chipset. https://pica8-fs.atlassian.net/wiki/spaces/PicOS442sp/pages/4261487/Flow+Scalability+per+Broadcom+Chipset, 2023.

[38] Benjamin E Ujcich, Adam Bates, and William H Sanders. Provenance for intent-based networking. In *2020 6th IEEE conference on network softwarization (NetSoft)*, pages 195–199. IEEE, 2020.

[39] Benjamin E Ujcich, Samuel Jero, Anne Edmundson, Qi Wang, Richard Skowyra, James Landry, Adam Bates, William H Sanders, Cristina Nita-Rotaru, and Hamed Okhravi. Cross-app poisoning in software-defined networking. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 648–663, 2018.

[40] Benjamin E Ujcich, Samuel Jero, Richard Skowyra, Adam Bates, William H Sanders, and Hamed Okhravi. Causal analysis for {Software-Defined} networking attacks. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3183–3200, 2021.

[41] Benjamin E Ujcich, Samuel Jero, Richard Skowyra, Steven R Gomez, Adam Bates, William H Sanders, and Hamed Okhravi. Automated discovery of cross-plane event-based vulnerabilities in software-defined networking. In *Network and Distributed System Security Symposium*, 2020.

[42] UTM. Utm. https://mac.getutm.app/, 2025.

[43] Ben Weintraub, Jiwon Kim, Ran Tao, Cristina Nita-Rotaru, Hamed Okhravi, Dave Tian, and Benjamin E Ujcich. Exploiting temporal vulnerabilities for unauthorized access in intent-based networking. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 3630–3644, 2024.

[44] Feng Xiao, Jinquan Zhang, Jianwei Huang, Guofei Gu, Dinghao Wu, and Peng Liu. Unexpected data dependency creation and chaining: A new attack to sdn. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1512–1526, 2020.

[45] Changhoon Yoon, Seungwon Shin, Phillip Porras, Vinod Yegneswaran, Heedo Kang, Martin Fong, Brian O'Connor, and Thomas Vachuska. A security-mode for carrier-grade sdn controllers. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 461–473, 2017.

[46] Engin Zeydan and Yekta Turk. Recent advances in intent-based networking: A survey. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5. IEEE, 2020.