Automatic Discovery of Evasion Attacks Against Stateful Firewalls

Soo-Jin Moon, Yves Bieri, Ruben Martins, Vyas Sekar

January 19, 2021

CMU-CyLab-21-001

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Automatic Discovery of Evasion Attacks Against Stateful Firewalls

Soo-Jin Moon, Yves Bieri[§,*], Ruben Martins, Vyas Sekar

Carnegie Mellon University, [§]*Compass Security*

Abstract

Stateful firewalls (FW) play a critical role in securing our current network infrastructure in various deployments. In this work, we focus on discovering evasion attacks that arise due to semantic implementation vulnerabilities of the intended stateful behaviors. Such attacks enable firewall evasion even if the rules are configured correctly. This is in contrast to prior work that focused on software bugs for privilege escalation and/or policy misconfigurations. Given the black-box and proprietary nature of firewall implementations, we design and implement a model-guided approach for uncovering such evasion vulnerabilities. Specifically, we infer a behavioral model of a specific FW implementation and then use the inferred model to synthesize attack strategies for a given deployment and threat model. In designing *Pryde*, we address key technical challenges in ensuring that our model inference is tractable and our attack synthesis can cover multiple semantic vulnerability opportunities. We evaluate *Pryde* on four production-quality firewalls. We discover thousands of distinct attack sequences for 4 popular firewalls (FW).

1 Introduction

Network firewalls (FW) play a critical role in securing our current network infrastructures in various deployment settings – including enterprise networks [14], cloud virtualized networks [3], and modern containerized settings [10]. Specifically, these firewalls impose restrictions on undesirable traffic from the outside network to the intranet.

Of particular interest are *stateful* firewalls. As opposed to simple access control lists, stateful firewalls track the state of individual TCP connections (together with firewall rules) to determine which packets are forwarded. A canonical policy in such a setting is that only packets on TCP connections that have previously been established by hosts inside the intranet are allowed by the firewall.

Administrators deploying firewalls implicitly assume that the vendor implements the stateful semantics correctly. Unfortunately, if vendor implementations are erroneous, then it can weaken the security posture. For instance, an enterprise firewall with such errors could allow external attackers to reach internal hosts that should not be reachable. Note that this is orthogonal to prior work in firewall analysis that examines the safety of the firewall rules and misconfigurations (e.g., [18, 20, 42]). The types of vulnerabilities we consider are a more fundamental semantic vulnerability in tracking internal states and hence are viable even if the rules are configured correctly.

Unfortunately, operators have few, if any, tools to check if the vendor firewall implementations have such semantic vulnerabilities. Manual investigation will be ineffective and not scalable across many vendors and versions. Ideally, we need to automatically identify such evasion vulnerabilities. Complicating this further, FWs are proprietary and acquired from vendors. Hence, operators have limited visibility into the code/internals of these FWs.

This black-box setting makes the analysis even more challenging. While fuzzing or black-box pen-testing tools (e.g. [2, 13, 26, 27]) or recent work on censorship evasion (e.g., [22, 41]) appear as strawman solutions, in practice they have shortcomings. First, their focus is often orthogonal to our intent. For instance, pen-testing focuses on finding privilege escalation or application-layer problems in the management APIs. The prior work on censorship evasion focuses on the opposite problem of an internal host accessing an external service. Second, they cannot handle the large search space of possible stateful or connection-oriented packet sequences. Third, they are not robust in discovering subtle attacks across FW implementations with varying degrees of implementation complexity.

In this work, we present *Pryde*, a black-box analysis framework for automatically uncovering semantic evasion vulnerabilities for firewalls. *Pryde* only requires input-output access to the firewall and does need visibility into the binary or the code. We specifically focus on evasion vulnerabilities that al-

[*]Contributions by Yves Bieri were made during the time he was a visiting researcher at Carnegie Mellon University.

low an external attacker to circumvent the firewall to send an undesirable "data" packet to an internal target which should not be reachable by the policy. This capability can subsequently be used as a starting point for more detailed attack campaigns for persistent threats, lateral movement, and/or exfiltration of sensitive information.

Pryde uses a *model-guided approach* [40] that proceeds in two logical phases. The first phase uses blackbox model inference to reason about the stateful behavior a given firewall implements for different packet sequences. Then, given this inferred model, we consider different deployment and threat scenarios to identify evasion vulnerabilities. In contrast to random fuzzing or randomly generating test packet sequences, our approach is *efficient* and can uncover many semantically different vulnerabilities (§6). Our approach is general across firewall implementations and extensible to support future evasion scenarios. In designing and implementing *Pryde*, we address key technical challenges to (1) make the model inference robust to consider different types of packets, and (2) efficiently encode the scenarios in a model checker and define custom refinement constraints to enable the model checker to explore new attack pathways efficiently.

Findings: Realizing the above workflow and ideas, we design and implement *Pryde*.[1] We evaluate *Pryde* on four popular (virtual) firewalls where three are commercial-grade and one open-source. Two of these 4 are taken from the Amazon EC2 marketplace [1] and also used in the cloud deployments. (We anonymize vendors' names.) We summarize our findings below (§6):

- *Many semantically distinct attacks*: We uncover thousands of semantically distinct (i.e., with respect to the connection states traversed) attacks—2,591 for FW-1, 2,355 for FW-2, 8,220 for FW-3, and 294 for FW-4. Post-processing these attacks, we find that these attacks exploit different aspects of the TCP. For instance, some of these attacks exploit a FW forwarding an external DATA packet "even before" the three-way handshake has been completed or after an incomplete handshake has been disrupted. Some exploit the simultaneous open feature of the TCP and/or SYN retries. At a high level, these attacks exploit (1) non-traditional sequences of TCP packets; (2) interference from other TCP connections (e.g., same headers in reverse direction or non-compliant sequence/ acknowledgement numbers), and combination of (1) and (2).
- *Many evasion attacks are subtle*: While some attack sequences are simple (i.e., requiring only one or two TCP packets), many others are quite subtle and require a nuanced TCP packet sequence. Specifically, we uncover more attacks involving longer sequences that are not captured in smaller sequences; e.g., 47 attacks for an attack length of 1 to 3 vs. 9,231 for an attack sequence length of just 7. For instance, FW-1 entails a carefully constructed TCP packets involving RST packets after the SYN retries to allow a DATA. Several attacks are also unique to specific vendors and do not extrapolate across vendors.
- *Strawman solutions are ineffective*: In contrast to *Pryde*, random fuzzing only finds a handful of attacks (0 to 3) for FW-3 and FW-1 after more than 15K tries. This approach is not robust across FW implementations. Recent work on automatic censorship evasion using genetic algorithms (e.g., [22]) is also ineffective at uncovering these attacks. (To be fair, censorship circumvention focuses on an orthogonal problem with a different system goal and threat model that makes their strategies ineffective in our context.)

Ethics and Disclosure: We have disclosed preliminary findings to vendors.

2 Background and Motivation

In this section, we provide the background on stateful firewalls (FW) and the deployment model we consider. Then, we highlight motivating examples of evasion vulnerabilities against a commercial firewall to motivate our work.

2.1 Background on Stateful Firewalls

A FW is the central network device that stops or mitigates unwanted access to the private (protected) networks from other untrusted external networks such as the Internet. We specifically consider a layer-3 stateful firewall (FW) in this work. A stateful FW decides whether to drop or forward a network packet based on the 5-tuple defining a connection (srcip, srcport, dstip, dstport, proto), and the configured rules. Typically, a stateful FW is configured with rules of the following form: { interface, srcip, srcport, dstip, dstport $\rightarrow$ action} where interface refers to an interface where incoming TCP packets are matched to (also, can be a wildcard). A FW configured with each rule keeps the states of network connections (i.e., a TCP connection state) and tracks a state during a connection's lifetime based on the 5-tuple. Further, a FW also monitors both incoming and outgoing packets across interfaces to update the connection state accordingly. This connection state may include details such as the sequence (seq) and acknowledgment (ack) numbers of the packets traversing the connection.

Deployment model: In an enterprise network, stateful FWs (Figure 1) are typically configured to drop all external packets that do not belong to an established connection initiated from an internal host. For instance, the firewall must allow internal hosts to initiate a connection to a web service, W, and also allow the return "data" packets from W to arrive at the host that initiated the connection. This is the "stateful" part; if we only had simple stateless rules to allow traffic from internal hosts to W and block incoming traffic from the Internet, then the return data packets from W would never arrive.

[1] The name *Pryde* is inspired by the Marvel X-men superhero Kitty Pryde who has the ability to use her "phasing" powers to walk through walls [8].

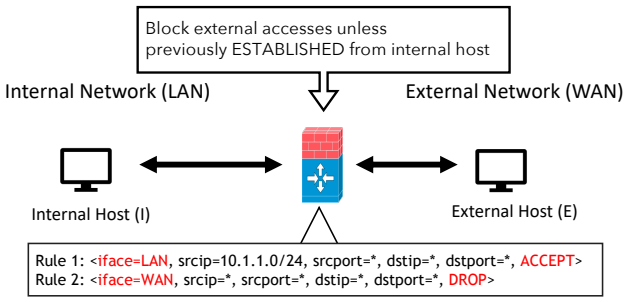
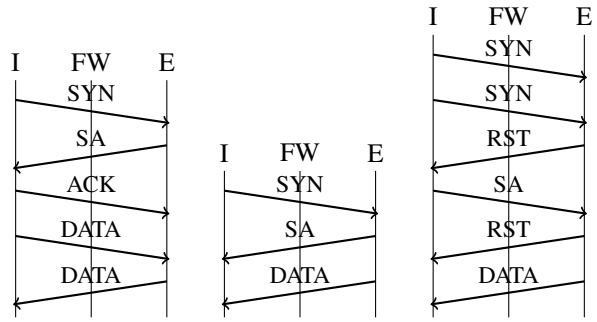


Figure 1: **A Stateful Firewall in an Enterprise Network**

More generally, suppose a TCP packet with srcip:A, srcport:Ap, dstip:B, dstport:Bp enters via an external interface, the FW checks whether the connection from srcip:B, srcport:Bp, dstip:A, dstport:Ap is in the established state. If it is, the packet will be forward and otherwise, this packet will be dropped.

The enterprise scenario we consider differs from other scenarios such as censorship. In the censorship scenario, FWs are typically configured with the "default-allow" policy for packets originating from both directions. However, these FWs will inject RST packets (to terminate the connection) if a client accesses "blocked" content. As such, the evasion attacks we focus on are orthogonal to those considered in censorship circumvention (e.g., [22, 41]).



(a) Normal Scenario (b) Scenario 1 (c) Scenario 2

Figure 2: **Packet sequences played against FW-1**

2.2 Motivating Scenarios

Note that the logic of how the FW decides to a particular action to the current state plays a critical role in determining whether a packet is dropped or not. Therefore, a flawed implementation on how a FW tracks a connection state can have a detrimental effect on the security posture. To motivate this, we use concrete evasion attacks we uncovered with a real commercial FW-1. Before discussing the concrete attack scenarios, we first showcase *vulnerable* sequences of packets (which can be exploited for attacks). Specifically, this leads to a FW allowing a DATA packet from an untrusted external host (E) to an internal host (I).

To explain these suspicious sequences, we contrast them with a normal (standard) packet sequence (probably, a sequence a FW expects to see) as shown by Figure 2a. An internal host wants to access W and initiates a connection setup by sending a TCP SYN. W acknowledges this by sending a SYN-ACK (SA for short), followed by an ACK from an internal host, thereby completing a proper three-way handshake. At that point, an internal host and W can freely exchange DATA packets. Now, we show two strange sequences of TCP packets, that will start allowing a DATA packet from an untrusted external host; these strange packet sequences drive a connection state to some limbo (vulnerable) state and that is when the FW also starts to accept a DATA packet from untrusted external hosts.

Scenario 1 (Incomplete handshake): As seen in Figure 2b, the FW-1 FW allows a DATA packet from W "even before" a three-way TCP handshake has been completed; i.e., the FW is not checking whether the last ACK packet from an internal host has been sent or not. Such a simple error highlights that implementing even a very basic stateful semantics of checking for a complete handshake can be erroneous. In practice, this problem is much worse than what it appears as such an error can manifest in so many different ways (i.e., polymorphic variants of this attack as we will see in §6.2).

Scenario 2 (SYN retries + Teardown): We now show a more complex and a different sequence (Figure 2c) from the Scenario 1. This scenario exploits an implementation error in how a FW-1 FW handles a combination of SYN retries and connection teardown.[2] Here, an internal host first sends two subsequent SYN packets (like SYN retries). Then, an external host sends a RST packet, which drives this connection state to some "limbo" state. After two non-traditional TCP packet exchanges (an internal SA followed by an RST), a FW allows a DATA packet from an external host! One may wonder whether all of these 5 packets are necessary for a FW to allow a DATA packet. In fact, that is the case as each packet in a sequence modifies the connection state. Further, only after the first two SYN packets, the FW does not allow a DATA packet for a FW-1 FW (however, a FW-3 stateful FW does, which is motivating our work). These sequences are discovered by our tool and as we discuss in §5.1, we specifically only output semantically-distinct attacks (from a black-box perspective).

Attack scenarios: Now, let us think about how such erroneous implementation can lead to concrete evasion attacks. One precondition for concrete attacks for the above two scenarios is that an internal host needs to send specific TCP packets (e.g., the first SYN packet in Scenario 1) in coordination with an external attacker. But, this isn't too difficult as any internal host can easily *spoof* the source IP-port and send

[2]While prior works on censorship evasion (e.g., [22, 41]) find similar attacks, their focus is orthogonal to ours as their deployment and the system model differs. Their tool/findings do not directly apply to our setting (§6.1)

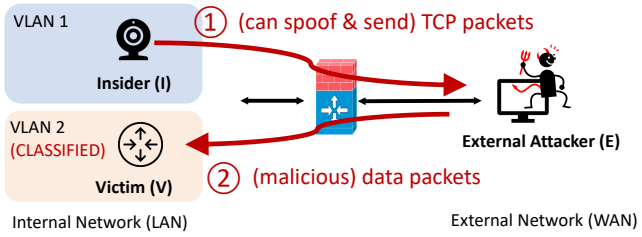


Figure 3: **Attack scenario setup**

it outwardly. Consider a case where we have a compromised insider such as an IoT device (e.g., a smart printer) in the intranet [7, 15]. In fact, first compromising these IoT devices is increasingly gaining traction for hackers due to IoT devices' prevalence in today's enterprise network [11] and their lack of built-in-security [7]. In fact, it is recently reported that Russian-state hackers, Strontium (APT28), have been caught attempting to hack IoT devices (e.g., an office printer, a video decoder) to gain entry points into their targets' internal networks [9]. Such an insider (Figure 3) may lack direct access to the target victims as it is not located in the same VLAN as the target victims (e.g., router, end host). However, this insider colludes with an external attacker and exchange a sequence of pre-defined TCP packets (including Step ①). Finally, a FW allows a DATA packet directed at the target victim (Step ②). The best micro-segmented network with VLANS using the best practices today [5] is also vulnerable to these attacks.

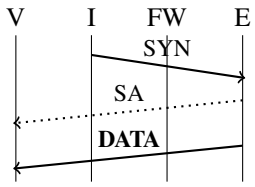


Figure 4: **Scenario 1 mapped to an attack setup**

Figure 4 shows the Scenario 1 mapped to a concrete attack. In this attack, an insider first sends a SYN packet with a source IP-port that of a victim, followed by a SA from an attacker. Then, an external attacker can circumvent the FW policy by sending a DATA packet (containing malicious payload) at a victim. It is outside the scope to precisely guarantee whether all victim software stacks actually accept and process this data packet as such. As such, we observe that there are many cases (e.g., routers, IoT devices) that will accept the data packet. At this point, the attacker gained an entry into a highly-classified VLAN. An attacker could either compromise this target victim or use this victim as another stepping stone to enable more sophisticated multi-stage attacks.

Attack characteristics: Having described motivating examples, we now derive several characteristics of these attacks:

- *Semantic evasion attacks are subtle*: These attacks exploit subtle implementation nuances in a FW's logic of tracking a per-connection state. Specifically, to make these attacks to work, one needs to carefully construct packet headers with the above TCP flags and seq numbers. Moreover, these attacks may be specific to each FW vendor's implementation.
- *Semantic diversity of evasion opportunities*: As we saw brief examples for FW-1, as these attacks the fundamental issue in tracking per-connection states, there tend to be multiple such attacks exploiting diverse mechanisms (e.g., handling teardown packets, incomplete handshake, SYN retries). Further, even within attacks that exploit a similar mechanism, there are multiple polymorphic variants that are semantically different [33] (detail in §6) that explore the different stateful semantics.

The above suggests that we need a general and robust framework to uncover such evasion attacks. Given the subtle and implementation-specific nature of these attack opportunities and stateful behaviors involved, strawman solutions such as randomly generating packet sequences are inefficient (§6).

3 Problem Overview

In this section, we formulate the problem of enabling a model-guided approach and formulate our problem. We provide an overview of the *Pryde* workflow and discuss the key challenges in realizing our workflow.

3.1 Threat Model

We begin by scoping the adversary's goals and capabilities.

Adversary goals and capabilities: The attacker's goal is to circumvent the FW and send a DATA packet to an "unreachable" internal victim. We assume the following attacker capabilities and constraints.

- *Send constructed packets*: An external attacker can craft TCP packets and send them to internal hosts.
- *A colluding insider with minimal privileges*: The attack may optionally have a "weak" insider that can spoof the source of the TCP packets and send them to external hosts. A "weak" insider cannot directly send packets to the victim; e.g., internal firewalls or VLAN policies may prevent this.
- *FW-specific knowledge*: The attacker does not know the rules that the firewall is configured with. We assume the attacker knows the FW vendor/version; if not this can be obtained by known fingerprinting mechanisms such as banner grabbing [6]. We assume the attacker has no visibility into the internal implementation or code. However, we do assume that the attacker can have offline "black-box" access to the FW (e.g., obtain a virtual FW appliance [1]).

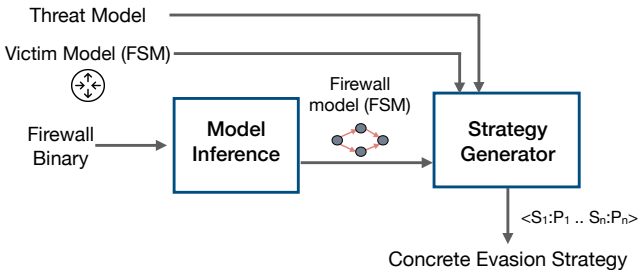


Figure 5: **Pryde System Overview**

3.2 Problem Formulation

Input and output: *Pryde* takes as inputs: (1) FW binary or virtual appliance; and (2) a deployment and threat model defining different entities (i.e., an insider, an external attacker) and their capabilities (e.g., an insider can spoof and send TCP packets). The output of *Pryde* is a set of semantically-distinct concrete evasion strategies (Def. in §5.2). Specifically, each concrete evasion strategy is an ordered sequence of *input TCP packets* mapped to the corresponding sender (i.e., an insider or an external attacker); an example was shown in Figure 4.

Scope: In this work, we focus on sending a DATA packet from an external attack to an "unreachable" internal victim (as defined by the policy). We acknowledge that not all victims may actually accept and process this data packet. As such, we observe that there are many cases (e.g., routers, IoT devices) that will accept the data packet. Note that the attack's goal after this circumvention (e.g., installing back-doors or lateral movement) is outside our scope. Our attack is a fundamental first step that can enable future attacks.

Challenges: There are two main challenges in enabling our vision. First, the input space is too *large* for an unstructured search (i.e., random search). Specifically, as we deal with an adversarial scenario, we need to consider *sequences of packets* where each packet can come from diverse sets (e.g., a sequence of non-standard TCP flags, out-of-window packets, a flow with a flipped direction). Second, there may be *multiple such evasion strategies* that could exploit different features or code paths. While we cannot guarantee coverage, our goal is to discover as many attacks as possible and also attacks that are semantically different (from the point of view of the black-box analysis).

3.3 High-Level Design

A case for a model-guided approach: The evasion attacks we consider exploit nuanced FW-specific aspects. For instance, Scenario 2 (§2) incorrectly allows an external DATA packet, after SYN retries and teardown packets. That is, identifying such attacks require carefully-constructed sequences of TCP packets, triggering internal state transitions that will not be exercised by normal TCP sequences. As a result, strawman solutions (e.g., random fuzzing) will not discover many of these attacks for many FW (§6.1). Instead, we adopt a *model-guided approach*, where we first infer a behavior model of the stateful semantics. We do note that this model has to be specific to each FW implementation, since the connection handling semantics of different FWs may be different. Having a model enables us to systematically search over this state-space to discover semantically distinct attack opportunities.

A case for a two-phase approach: One option of a model-guided approach is to couple threat model encoding with model inference. Unfortunately, this is not *extensible* as our system and deployment assumptions change; e.g., modeling a weak insider option would require us to re-learn the model for each possible scenario of the insider's capabilities. By decoupling model inference from attack generation, our workflow is *extensible* to future threat and deployment models.

Thus, *Pryde* consists of two logical modules (Figure 5):

1. *Model Inference (§4)*: Given a black-box FW implementation/binary, the Model Inference engine outputs a Finite-State Machine (FSM) model. This model describes the input and output packets of the FW in a given connection state. Recent work demonstrates the feasibility of black-box model inference for stateful FWs [33]. However, they make several simplifying assumptions and focus on inferring a FW model under typical or normal packet sequences and thus cannot directly be used in our context; e.g., how a FW behaves in presence of out-of-window packets that "interfere" with an existing connection. We address key challenges in extending prior work to infer model under more general or anomalous packet sequences.
2. *Strategy Generator (§5)*: Given a FW model, we need a systematic way to uncover of evasion attacks under the interactions of different entities (i.e., attacker, victim, insider, and the FW). To this end, in this module, we formulate these system interactions using SMT and use Z3 [24] to build a custom model checker. We model the problem similar to bounded model checking [23], where we check if a bounded length path sequence exists. To uncover semantically different attacks, after finding one attack, we refine our constraints in the model checker to uncover more semantically-different attacks.

4 Model Inference

In this section, we discuss the design of the Model Inference module.

4.1 Prior work on model inference for stateful FWs

A FW's processing behavior can intuitively be captured as a finite state machine or FSM [25, 33]. Thus, we can potentially use classical algorithms (e.g., L* [21]) for a black-box FSM inference. At a high-level, L* adaptively constructs sequences

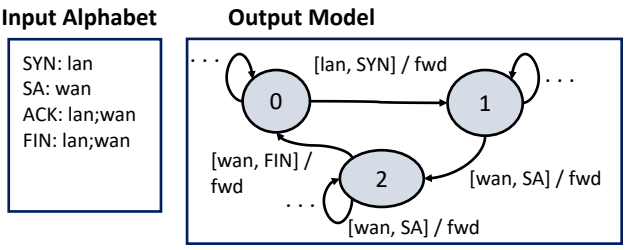


Figure 6: **An example of an output model and the corresponding input alphabet used (lan for internal and wan for external due to space)**

of varying lengths using these input symbols (in an input alphabet), injects them into the blackbox under consideration, and infers the hidden states and transitions by observing the outputs. However, directly applying L* in a network setting is challenging as we would need to handle NF (network function) specific behavior (e.g., drops, modify) and a large space of possible TCP/IP packets to consider.

Moon et al. [33] proposed a tool called *Alembic* that addresses some of these network-specific challenges to make the model inference for network functions or NFs (e.g., FW) tractable. At a high-level, this tool takes an input alphabet (Σ) describing a scoped set of packet types of interest (e.g., TCP SYN from A to B, TCP SA from B to A) to extract the NF behaviors. Further, they propose using specific optimizations to reduce the size of an Σ. For instance, rather than considering the entire possible space of TCP headers such as sequence (seq) and acknowledge (ack) numbers (32-bits each), their system re-writes these seq and ack number of TCP packets during the actual inference to adhere to the TCP semantics. By doing so, *Alembic* does not have to search the space of seq and ack numbers.

Concrete example: Consider a FW configured with a policy (Figure 1). The intended policy is to "drop all external plackets unless it belongs to a connection established from an internal host." Here, the default rule for packets originating from external hosts is to drop. To reason about the stateful semantics, we need to infer a model with an input alphabet (Σ) as shown in Figure 6 (i.e., SYN from an internal/lan interface, SA from an external/wan interface). Each output model is a Mealy machine, where inputs and outputs defining the machine are located packets (e.g., a SYN packet with source A and destination entering from an internal interface). Each located packet is a tuple of (interface, TCP flag, srcip, srcport, dstip, dstport). For simplicity, the model in Figure shows the model for one connection only. (To be more precise, *Alembic* produces an ensemble of FSM, one for each connection, but that is not relevant for our work.)

4.2 Limitations of the prior work

While Alembic [33] is a good starting point, we now elaborate on why it is insufficient for *Pryde*.

1. *Need to consider diverse input alphabets*: This prior work [33] is designed to model the FW for mostly TCP-compliant workloads. However, we need to consider adversarial scenarios (e.g., out-of-window packets) that "interfere" with the TCP-compliant connection states. For instance, Scenario 2 (§2) would not have been discovered if we hadn't considered an internal SA. While this is just a simple example, we need a systematic way to generate input alphabets to reason about potential evasions.

2. *Support for rewriting logic*: As this tool, *Alembic*, makes optimizations to reduce the search space by rewriting seq and ack numbers during the inference. Unfortunately, we need to consider adversarial cases where such re-writing logic may not help us to uncover certain types of evasion attacks. We need to come up with a *general* way of handling seq and ack headers during the inference to support various types of input alphabets.

4.3 Generating evasion-centric input alphabets

We discuss how we generate input alphabets (Σ) for the evasion attacks. A strawman solution is to just generate all possible packets as Σ. Unfortunately, the size of input will grow exponentially as we need to consider possible combinations of directions, TCP flags, and those TCP packets that adhere to the TCP semantics and those that do not. Instead, our idea is to come up with an ensemble of *independent* Σ where each model learned with a given Σ sheds light on the potential evasion scenario (i.e., TCP states interfering with packets with the reverse direction). That way, our method is systematic and extensible to future Σ we may consider.

Further, evasion can happen with or without interfering packets. Hence, we first set a *basic* Σ that can reason about attacks using just "non-traditional" sequences of packets (i.e., a sequence of non-standard TCP flags). Then, we showcase how we generate Σ to reason about interference. In this work, we only consider interference from packets that share the *same* bi-directional tuple; i.e., a TCP packet from lan has source A and destination B and a packet from wan has source B and destination A. This is a conscious decision as from observations/anecdotes suggest the FWs mostly have flaws in processing the state for packets with the same 5-tuples. (It is easy to extend our design to check if two connections with different source and/or destination interfere with each other.)

Basic input alphabet: Figure 7 shows the two basic input alphabets (Σ). We also denote abbreviations for input symbols or packet types (e.g., SA and SYN-ACK, DA for DATA).

Data Injection (DI)	Data Injection with Teardown (DI-T)	
SYN: lan	SYN: lan	R: lan; wan
SA: wan	SA: wan	RA: lan; wan
A: lan; wan	A: lan; wan	F: lan; wan
DA: wan	DA: wan	FA: lan; wan

Abbreviation:
SA : SYN-ACK
R : RST ; RA: RST-ACK
F : FIN ; FA: FIN-ACK
DA : DATA

Figure 7: **Basic Input Alphabet for *Pryde***

- *Data Injection (DI)* : This Σ helps to reason about potential circumvention just using the connection setup packets. This Σ has SYN from lan, SYN-ACK from wan, ACK from lan and wan, and DATA from wan (left column in Figure 7).

Further, from anecdotes and prior works [41], we also know that the connection teardown packets can transition the connection to some "TIME-WAIT" or some limbo state where certain packets (e.g., a DATA) can slip through. Hence, we also need to reason about such a scenario and introduce an additional *basic input alphabet*.

- *Data Injection with Teardown (DI-T)* : This alphabet adds teardown packets to the DI input alphabet.

Interference input alphabet: We now need to consider Σ to also reason about potential evasion from interference with non-compliant TCP packets; e.g., packets that do not belong to the same connection (e.g., [22, 41]). Specifically, we observe from anecdotes/prior works that the FWs can incorrectly map the state with (1) TCP connection with flipped direction (e.g., SYN from lan vs wan); and (2) packets with out-of-window seq numbers. These cases are "subtle" such that these packets share the same 5-tuple. However, if a FW was implemented correctly, it should have not been considered so.

In this work, we consider four possible types of interference (IX) input alphabet. Given each IX set, we append them to the basic Σ to reason about FW's behavior when it encounters both types of packets. While we come up with a representative Σ, our design is easily extensible for a new Σ. Further, we consider interference from connection setup-relevant packets.

1. *Reverse directions (*IX^{dir}*)* : These TCP 3-way handshake packets have flipped direction only; i.e., SYN coming from the wan network, SYN-ACK from lan, and ACK from wan. The seq and ack numbers are *not* out-of-window.
2. *Reverse direction and random seq/ack numbers (*$\text{IX}^{\text{dir}}_{\text{rand}}$): These TCP 3-way handshake packets have flipped direction and also are out-of-window. Specifically, the seq & ack numbers are randomly initialized.
3. *Reverse direction and random seq/ack numbers (*$\text{IX}^{\text{dir}}_{\text{conn}}$): This case is similar to the previous case. However, these out-of-sequence packets themselves form a connection; i.e., their seq/ack are are in-window among themselves.
4. *Packets with out-of-sequence* (IX_{rand}): These TCP 3-way handshake packets have the same direction but are out-of-window and randomly initialized.

A generative model for input alphabets: The generative model for the input alphabet is formed by considering each of the four basic Σ independently. For each basic input alphabet

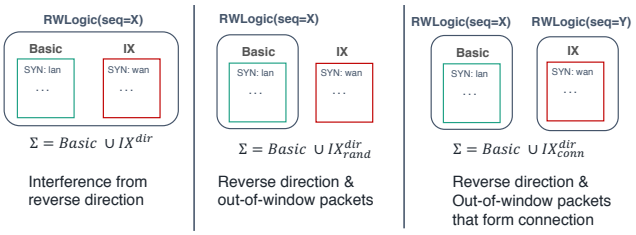


Figure 8: **Rewriting logic to handle interference** (IX) **set**

(Σ), we append the IX set and reason about the FW's behavior. This gives a total of 2 basic Σ and × 5 (1 for basic and 4 for interference) = 10 input alphabet for each FW. However, note that during the model interference, the model may not converge for certain FWs; i.e., *Alembic* assumes that the underlying model is deterministic and fail if this is not the case. Hence, if the model does not converge for the basic Σ, we do not model for the corresponding *input alphabet* appended with IX set.

4.4 Extending the inference algorithm

We now discuss how we enable the model inference under these diverse input alphabets.

Baseline rewriting logic (RWLogic()**):** We briefly discuss the seq/ack rewriting logic used by the prior work [33] to reduce |Σ| At a high-level, their tool tracks seq and ack of the transmitted packets and rewrites them during the inference to adhere to the correct semantics. For instance, suppose the underlying L* plays this packet sequence of length 2: 1) SYN from A to B from the lan network (SYN:lan), and 2) SYN-ACK from wan. If the FW forwards the first SYN to a wan interface, then the seq number of the transmitted SYN is used to update the seq and ack number of the TCP packets with a source *B* and a destination *A*. We refer to this entire logic as RWLogic($seq = X$) where X is the initial seq number.

Rewriting logic to support interference set: We now discuss how we adjust this logic to handle the "interference" packets. The beauty of this extension is in the generality of this design that makes it extensible and general for future Σ to consider. Note that from the IX sets from §4.3, there are cases where we need to subject only a certain group of packets to rewriting, or subject multiple groups of packets to independent rewriting (i.e., reasoning when two connections with the same 5-tuple but initialized with different seq number such as $\text{IX}^{\text{dir}}_{\text{conn}}$).

For a systematic design, we group packets accordingly and subject each group to a corresponding rewriting logic (as shown in Figure 8). For instance, if we consider an interference set where only a direction is reversed (i.e., IX^{dir}), then we would subject both the basic and the IX set to have the seq/ack numbers in-sync (the left side of Figure 8 shows both sets

being subject to RWLogic(X)). Now, consider an interference set where these packets have a reverse direction and have out-of-window seq/ack numbers are (i.e., $\text{IX}^{\text{dir}}_{\text{rand}}$). Then, packets in the basic Σ are subject to RWLogic(X) and packets in the IX set are only randomly initialized and not re-written during the inference. Similarly, for IX where packets themselves form a connection, then we will independently rewrite seq % ack for them RWLogic(Y). The internal of modified implementation of *Pryde* keeps track of which "set" each packet belongs and decides if/how the rewriting logic is applied. These are all exposed configurable parameters.

5 Attack Strategy Generator

In this section, we discuss how we used the inferred models from §4 to generate concrete evasion strategies. We discuss how we encode the entire system model and the interaction between different entities (§5.1). Then, we demonstrate how we achieve coverage across distinct attacks (§5.2).

5.1 Encoding the system model

Strategy Generator takes an input of a firewall (FW) model (from §4) and the system model. The system model is defined by (1) a model of a victim, and (2) the threat model that defines each entity (i.e., insider, attacker) and their capabilities w.r.t. the packets that they can send. Specifically, we encode into the model checker that an insider can *spoof* the source IP and address of another internal host (e.g., victim).

The output is a *concrete evasion strategy*, which is an ordered sequences of *located input packets* (a concept borrowed from prior work in network verification [31]) mapped to the corresponding sender (i.e,. an insider or an external attacker).

At a high-level, a *located input packet* that comes to either of an interface of a FW and we define it below:

Definition 1 (A located input packet)**.** *A located input packet, σ, is defined by the following tuple $(intf, srcip, srcport, dstip, dstport, tcp, data, pre, seq, ack)$: (1) $intf$, an incoming interface (i.e., internal or external), (2) $srcip$, a source IP, (3) $srcport$, a source port, (4) $dstip$, a destination IP, (5) $dstport$, a destination port, (6) tcp, TCP flags, (7) $data$, a Boolean indicating the presence of a* DATA *(payload), (8) pre, a prefix indicating whether the seq/ack numbers were rewritten during the inference to follow the TCP semantics, (9) seq, a sequence number , (10) ack, an acknowledgement number. For (9) and (10), we use a variable such as X, X+1 to denote a relation across packets.*

A model of a FW (from §4) is a Mealy machine and is defined by the following tuple (Q, Δ, O, Φ): (1) Q, a set of states, (2) Δ, a set of input packets, (3) O, a set of output packets, and (4) Φ, a set of transitions. Similarly, a model of a victim is defined in the same manner as a FW.

Encoding a FW as a function of time: As mentioned, a concrete evasion strategy is an *ordered sequence* at discrete timesteps; i.e., we need to model the progression of time as a function of timesteps. For instance, when a FW gets an event (i.e., get a located input packet), the timestep T advances to $T+1$. Such an event changes the "state" of the entire system. Therefore, we use the following functions to describe a state of a FW at a given timestep, T:

1. State : $Q \times T \rightarrow$ Bool. Boolean function that indicates if a given state of the FW, $s \in Q$, occurs at a timestep T;
2. Input : $\Delta \times T \rightarrow$ Bool. Boolean function that indicates if an input packet, $\sigma \in \Delta$, occurs at a timestep T;
3. Output : $O \times T \rightarrow$ Bool. Boolean function that indicates if an output packet, $o \in O$, occurs at a timestep T;
4. Trs : $\Phi \times T \rightarrow$ Bool. Boolean function that indicates whether a specific transition, $\phi \in \Phi$, occurs at a timestep T. A particular transition is determined by an input packet (σ), output packet (o), and a current state (s).

We determine the possible sending entities based on the pre-specified thread model. As dictated by the threat model, an insider can spoof the source IP as a victim. Our encoding is extensible and more attributes can be easily be added.

Encoding constraints: We briefly describe how we encode these entities. Specifically, we encode a FW function using propositional logic with the following constraints:

- At a given $timestep$ T, we encode the following restrictions: (1) exactly one state occurs, (2) at most one input packet occurs, (3) at most one output packet occurs, and (4) exactly one transition happens.
- To encode a FW input model (a Mealy machine), we add pre- and post-conditions to specify the state transition.

Intuitively, a transition ϕ_j at a timestep T implies occurrences of a specific state and an arrival of a located input packet at the same timestep T. After a transition ϕ_j happens, then the state of a FW changes and as a result, we observe a corresponding output packet (defined by a Mealy machine). This can be represented as:

$$(\text{State}(s_i, T) \wedge \text{Input}(\sigma_i, T)) \implies \bigvee_j \text{Trs}(\phi_j, T))$$

$$\text{Trs}(\phi_i, T)) \implies (\text{State}(s_{i+1}, T+1) \wedge \text{Input}(\sigma_{i+1}, T+1) \wedge \text{Output}(o_{i+1}, T+1))$$

We also encode the victim's model using a similar logic. If an attacker's packet reaches the victim, then the next input is dictated by the victim's model (where in our current victim, a victim accepts all TCP packets). We discuss how this model could be extended in §8.

Encoding the goal: The goal is to find an ordered sequence of σ that leads to the FW to be "evadable" at a given timestep T; i.e., a DATA packet from an external attacker reaches an internal victim.

5.2 Discovering semantically-different attacks

To discover concrete attack strategies (i.e., sequences of a located input packet mapped to a corresponding sender), we model the problem similar to bounded model checking (BMC) [23] where we find counterexamples with a bounded length. BMC is a common technique to find bugs in software that can be identified within a few iterations (i.e., timesteps in our case). By default, the solver terminates upon finding one counterexample. To find a new counterexample, we must add additional constraints to block this counterexample and make a new call to the solver. However, this procedure would find many attacks that may be semantically-identical. Since the model-checker can be time-consuming, instead of outputting thousands of semantically-identical attacks and then do post-processing, we made a design decision to encode additional constraints that block semantically-identical attacks during the search. This procedure is repeated until no more counterexamples are discovered.

We now discuss the invariant of the attacks we output and the refinement strategy we use to enable the discovery of semantically-distinct attacks.

Loop-free invariant: To efficiently search over the state space of a FW's model, we want to output an attack string that uses a minimum of packets in traversing the state-space of a FW. Hence, we encode a loop-free invariant into our model.

Definition 2 (Loop-free invariant)**.** *A state, s_i, can appear at most once in a state sequence $s_1 \cdots s_n$ transitioned by an input sequence $p_1 \cdots p_n$, where p_i is a located input packet.*

Refinement strategy: Given this loop-free invariant, when we discover an attack packet string, a, composed of a sequence of located input packets, $\{p_1 \cdots p_n\}$, we exclude the exact packing string match. Hence, our refinement strategy corresponds to exclude equivalent strings.

Semantically-distinct attacks: Having defined the loop-free invariant and the refinement strategy, we can define semantically-distinct attacks. Note that, we can only provide this definition given the same input template (where the input space is identical).

Definition 3 (Semantically-distinct attack)**.** *Given two loop-free attack strings, a and a', they are semantically distinct if $a \neq a'$.*

By construction, the attack sequences (strings) generated by our tool are loop-free (from Def. 2). Additionally, as we do an exact string matching as a refinement strategy, all attacks that we output are *semantically distinct*.

6 Evaluation

System implementation: We implemented *Pryde* in Java atop Learnlib [38], an implementation of L* [21] for the Model Inference. We also built a custom *Python* based model checker using the Z3 [24] SMT solver. We implement other supporting modules for packet generation using *Scapy* [12]. Additionally, we have an automated framework in EC2 to spin up the Model Inference in EC2.

Setup: We use 4 off-the-shelf firewall implementations (i.e., FW-4, FW-1, FW-2, FW-3); three of them are proprietary and one has an open-source implementation (but we emulate it in a black-box manner). We ran FW-4 and FW-3 in VMs in VirtualBox [16] in CloudLab [4]. Two proprietary firewalls were from the Amazon EC2 marketplace [1] and were set up in Amazon EC2. We ran the Strategy Generator to find attacks of length 1 to 7. To test concrete attacks, we set up a sandbox network with an insider, a victim, an external attacker, and the stateful FW (configured to only allow TCP traffic from external hosts on already established connections as discussed in §2). We inject the packets via attacker and insider as dictated by the concrete attack strategy.

For each FW, we consider the following candidate input alphabets to identify evasion opportunities:

1. Baseline: Basic Σ without involving interference packets
2. IX^{dir}: Interference by reverse direction only;
3. $\text{IX}^{\text{dir}}_{\text{rand}}$: Interference by reverse direction and out-of-window packets with random seq and ack numbers;
4. $\text{IX}^{\text{dir}}_{\text{conn}}$: Interference by reverse direction and out-of-window packets that adhere to the connection semantics w.r.t. seq and ack numbers;
5. IX_{rand}: Interference by out-of-window packets with random seq and ack numbers.

For each template, we run the model inference and the attack generation for 1) Σ involving connection setup packets (i.e., Data Injection from §4.3), and 2) Σ for Data Injection with Teardown.

6.1 Aggregate summary of attacks

We first start with an aggregate summary and discuss the effectiveness of the strawman solutions.

Aggregate summary: Figure 9 shows the number of successful evasion attacks across all input alphabets. We found 2,591 semantically distinct attacks against FW-1, 2,355 against FW-2, 8,220 against FW-3, and 294 against FW-4.[3] From the Baseline templates, we have 1 attack (out of 294) for FW-4, 1253 (out of 2,591) for FW-1, 844 (out of 2,335) for FW-2, and 1,253 (out of 8,220) for FW-3. The attacks found using this template are the ones that come from non-traditional sequences of the TCP packets. The rest of the templates help

[3]These attacks are loop-free and semantically distinct given Σ (§5.2). For attacks across templates, we compressed identical attack strings (composed of located input packets). We also collapsed identical attacks identified by both connection setup and teardown alphabets

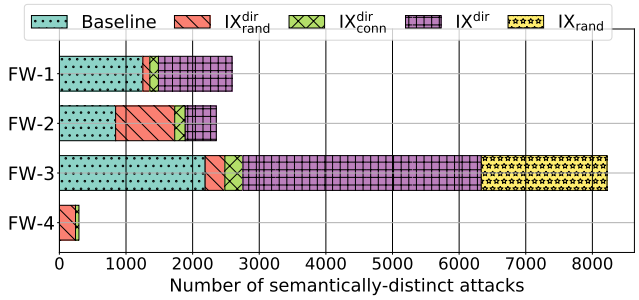


Figure 9: **Aggregate summary of semantically-distinct attacks found 4 FWs across all input templates**

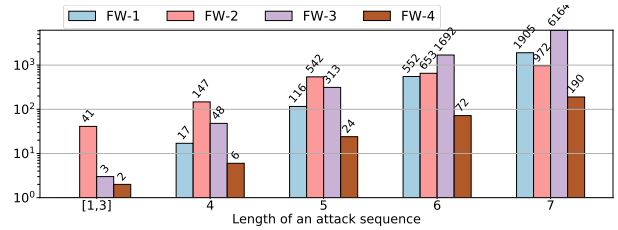


Figure 10: **Breakdown of the distinct attacks we found for each attack length across all FWs (Y-axis on a log scale)**

Pryde to discover attacks that come from the TCP packets (with identical 5-tuple) inferring the seq/ack numbers and/or other TCP connection with the reverse direction. For completeness, we also present the number of states for the inferred models (Table 2 in §A).

The number of distinct attacks is correlated with the number of states (i.e., the complexity of the stateful semantics) and whether that attack can be discovered with the bounded length. We see a relatively smaller number of attacks for FW-4 as (1) the size of the inferred FSMs are smaller in contrast to FW-3 (i.e., 2 for Baseline with teardown vs. 56 for FW-3), or (2) for one large FSM (more than 200 states), we did not find an attack within a bounded length.[4] Here, the take away is that there are hundreds to thousands of attacks leading to circumvention; i.e., patching one such code-path or sequences will be insufficient.

We also summarize the successful attacks based on the length of an attack sequence (Figure 10), where the y-axis is in a log-scale. We see a magnitude higher number of attacks for larger attack lengths. While all attacks are equally important, the attacks with longer sequences are likely more *subtle* in exploiting the implementation error/nuances of the FWs (more detail in §6.2).

[4]Using certain teardown-involving input templates, the model inference did not converge due to non-deterministic actions. While for one template, we inferred a large model, state transitions are rarely caused by the IX set. For non-Baseline template, we constrained the checker to only use packets from only IX set and we did not find an attack using less than 7 packets.

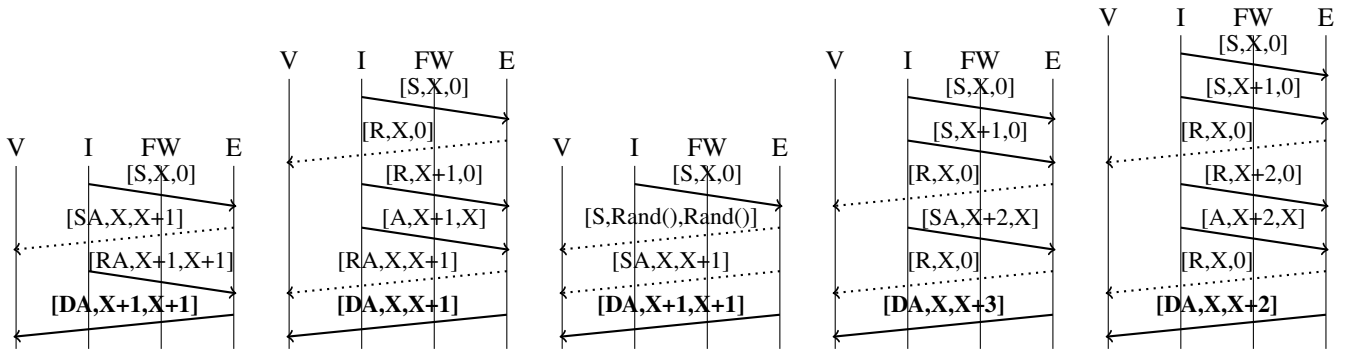
	Attack length							Total
	1	**2**	**3**	**4**	**5**	**6**	**7**	
# generated attacks	1	5	25	125	625	3,125	15,625	19,531
Without an insider								
FW-1	0	0	0	0	0	0	0	0
FW-2	1	5	25	125	624	3,124	12,068	19,096
FW-3	0	0	0	0	0	0	0	0
FW-4	0	0	0	0	0	0	0	0
With an insider								
FW-1	0	0	0	0	0	1	2	3
FW-2	1	5	25	125	625	3,123	15,618	19,522
FW-3	0	0	0	0	0	0	0	0
FW-4	0	0	1	3	15	91	586	696

Table 1: **# of "raw" attacks found using random fuzzing**

Comparison with strawman solutions: First, we consider a random fuzzing strategy that randomly generates packet sequences of lengths 1 to 7; the last packet in a sequence is a DATA packet from an external attacker. Hence, we only have one attack for a length of 1. For an attack sequence of a length, $L+1$, we generate $5\times$ the number of attacks for a length of L (giving a total of 19,531 sequences). Now, to pick each TCP packet in a sequence, we randomly sample values from "valid" TCP flags (i.e., SYN $\cdots$, and a DATA packet), a direction (i.e., internal vs. external), and concrete values of seq/ack numbers. Despite the strategy being called the random fuzzing, we only generate "valid" TCP packets (i.e., being generous). Further, as we lack information on the state each packet traverses to enforce the loop-free invariant and the refinement strategy (§5.2), we only report the "raw" number of attacks (it turns out non-trivial and there is no one-to-one mapping to project this randomly-generated sequence to our inferred models).

Note that using an insider was a pre-condition (found by *Pryde*) and hence is one of our ideas. We first consider a version with *no insider*. The top part of Table 1 shows the results where for FW-1, FW-3, and FW-4, we find 0 raw attacks. However, for FW-2, we see a high success rate.[5] As we will see in §6.2, FW-2 allows a DATA originating from an external network (even with an explicit "drop" rule). We also evaluate this strategy with an insider (Table 1). Even with using an insider, the fuzzing strategy is highly ineffective; specifically, for FW-3 and FW-1, the strategy discovers only 0 to 3 raw attacks (from 20K generated ones). This contrasts with 2,591 *distinct* attacks we discovered against FW-1, and 8,220 against FW-3. For FW-4, the random fuzzing found 696 raw attacks (may not necessarily be distinct). This is natural as the state space of FW-4 is quite simplistic (i.e., only 3 states for the IX^{dir} with teardown packets). Hence, it is relatively *easy* to get to a goal state. FW-2 has a close to 100% success rate for a similar reason as the case without an insider. At a high-level, this strategy is ineffective and not robust across FW implementations.

[5]2164 sequences caused an error, 12068 succeeded, and 38 sequences failed.



(a) **Incomplete handshake**(C1) (b) SYN + RST (C2+C3) (c) **Simultaneous open** (C4) (d) SYN **retries** (C5) (e) SYN **retries variant** (C5)

Figure 11: **Evasion attacks against FW-1 across 5 clusters.**

We briefly also evaluate the strategies found by a related work [22] on censorship evasion, which is an orthogonal problem to our own. The system model in this body of work [22, 41] is considerably different as these censorship FWs need to allow users accessing (un-censored) contents and, hence, has a default-allow policy. However, we still evaluated the 24 published strategies from Geneva. To map their attacks to our setting, an external web server maps to our internal victim, serving content, and their internal evader maps to our external attacker (evading an enterprise FW). Across all 4 FWs, none of these 24 strategies worked (i.e., a victim doest not receive a DATA packet). This is even true for FW-2 as the initial SYN from an external attacker is dropped (due to the default-drop policy). We discuss more about this body of work and also broadly, about applying genetic algorithms or model-free approaches for our problem context in §7.

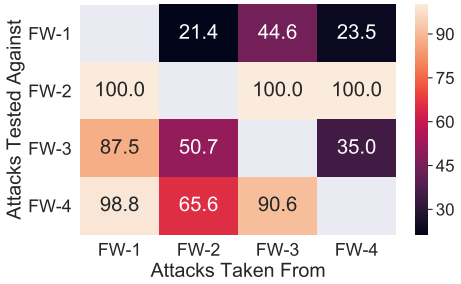


Figure 12: **Cross-validating the discovered attacks by taking successful attacks against a FW (x-axis) and testing on a FW (y-axis) and reporting the attack success rate**

Pairwise overlaps of successful attacks: First, we took the successful attack sequences from each vendor and replayed the sequence on other vendors. Figure 12 shows the results. For FW-2, attacks from the other three vendors lead to successes. This is because the FW-2 FW forwards a DATA packet to an internal host in all states (more details in §6.2). However, other than FW-2, we see low success rates for attacks seen in FW-4 and FW-2 on other FWs; only 23.5% of attacks from FW-4 work on FW-1. We revisit this when we look at the structure of these attacks in-depth.

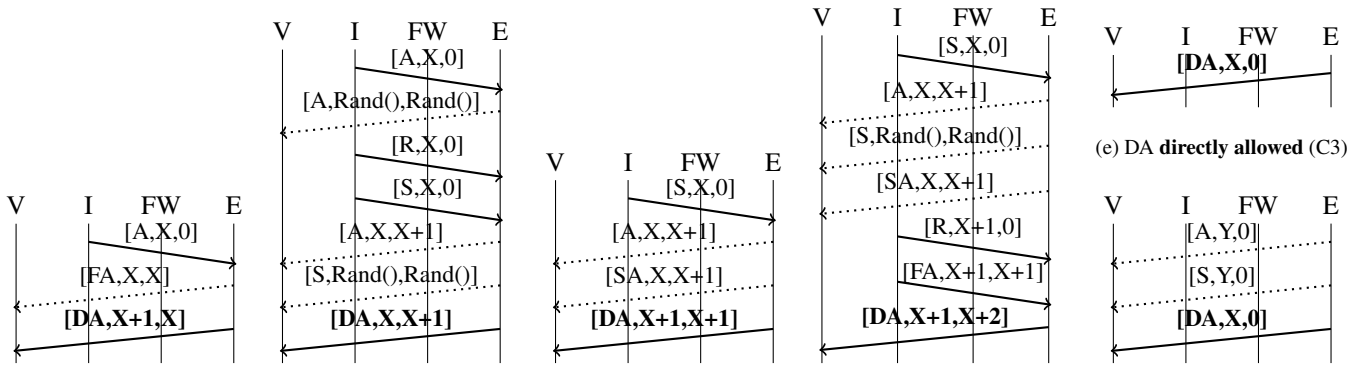
6.2 Structure of evasion attacks

Clustering attack sequences: To help us shed light on the structure of the uncovered attack sequences, for each FW vendor we cluster the packet sequences as follows. In our clustering formulation, each data point is an attack sequence composed of an ordered sequence of located input packets (Def. 3). From each located packet, for the clustering purposes, we exclude the specific values used for seq/ack numbers but a prefix (that indicates whether the seq/ack numbers was rewritten to comply to the TCP semantics). For each pair of sequences, we compute the Levenshtein edit distance. Given this metric, we run a complete-linkage hierarchical clustering algorithm, with a pre-specified target number of clusters. As the attacks differ across vendors, we used a different number of clusters (3 to 7) for each FW vendor.

For each cluster, we report a concrete attack sequence with the shortest attack length as a canonical example. We also depict other polymorphic variants within the cluster as required. Similar to §2, we use timing diagrams to specify these canonical attacks. In our diagrams, V refers to the victim, I is the insider, and E is the external attacker. A "dotted" line indicates whether a non-data packet reaches a victim. A bold line means that a DATA packet reaches a victim (i.e., successful circumvention). Further, each label in a line specifies the (TCP flag, seq, ack) from a located input packet; we use abbreviations for TCP flags (e.g., S for SYN, DA for DATA).

FW-1: From 2,591 attacks, we learned 5 clusters of size 2057, 163, 147, 144, and 80, respectively described below:

- *C1) Incomplete handshake and variants.* Scenario 1 (Figure 4) from §2 is the shortest-length attack in this cluster. Here, the FW allows a DATA packet just after seeing an



(a) **Internal** ACK (C1) (b) **Internal** ACK **variant**(C1) (c) **Internal** SYN (C2+C4) (d) **Internal** SYN (C2+C4) (e) DA **directly allowed** (C3) (f) **External** ACK (C3)

Figure 13: **Evasion attacks against FW-2 across 4 clusters**

SYN from an insider followed by a SYN-ACK from an attacker. A natural question is whether patching this specific sequence may remove this vulnerability. Unfortunately, this is not the case. Figure 11a shows other polymorphic variants that is more subtle and involves the connection state being "disrupted" by a teardown packet (i.e., RA). We also find hundreds of variants traversing other parts of the FW state-space; i.e., patching this problem can be non-trivial as an attacker may use other sequences.

- *C2+C3) SYN disrupted by* RST *or* RST-ACK *and variants.* The next two clusters contain attacks involving an initial SYN packet disrupted by an external RST packet (C2) or a RST-ACK packet (C3). The shortest sequence in this cluster is 6, indicating this attack is subtle; i.e., random fuzzing cannot discover these. Figure 11b shows that after an insider sends a SYN followed by a RST packet from an attacker, an insider and an attacker exchange three additional TCP packets, leading to circumvention of a DATA packet. There are many variants of this basic attack as well (not shown for brevity).
- *C4) Simultaneous open and variants.* The fourth cluster with 144 attacks exploits how the FW-1 FW handles the case where two SYN packets are concurrently sent from both directions. (This was found using the IX templates.) Figure 11c shows that in the shortest attack sequence. After the first SYN from an insider, the attacker sends a SYN packet, which drives the FW to another state (i.e., simultaneous open). After that point, the attacker sends a SYN-ACK followed by a DATA packet, reaching the victim. Again, we find many variants that explore the other regions of the state space using a variety of TCP flags (e.g., FIN-ACK, RST, and even DATA packets).
- *C5) SYN retries and variants.* The last cluster of 80 attacks exploits possibly incorrect handling of connection state after SYN retries. Figure 11d shows the shortest attack of length 6. We may think that to exploit SYN retries, we need a SYN-ACK to drive the FW to an incomplete handshake state (similar to C1). However, we also find an interesting variant (Figure 11e) that does not involve any SYN-ACK packet to exploit the SYN retries feature!

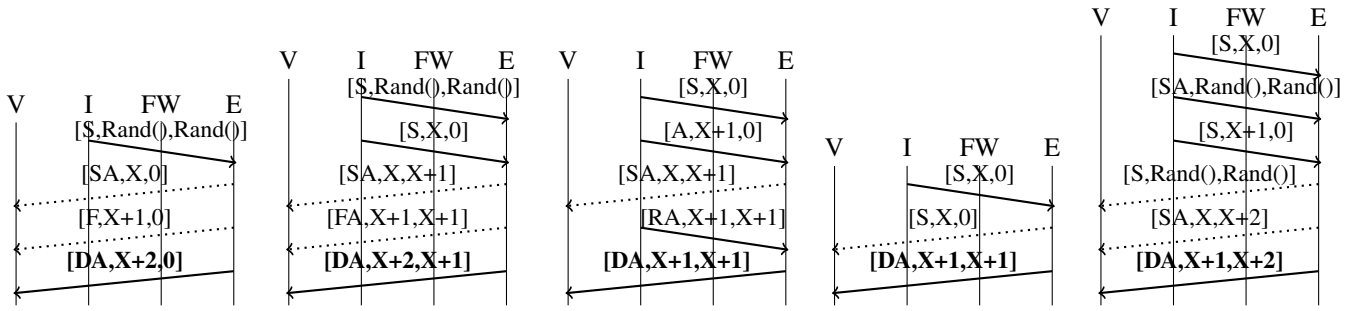
One invariant we observe here is that the first SYN packet that needs to be sent from an insider. However, as we will see shortly, this is not the case for other vendors (i.e., FW-2).

FW-2: We found 4 clusters of size 824, 806, 422, and 303, respectively. Recall that (§6.1), FW-2 allows a DATA packet from an external attacker even with an explicit drop rule (Figure 13e).

- *C1) Internal* ACK *and variants.* Attacks in this cluster use an external ACK from an attacker as the first packet. Figure 13a shows the shortest example. After an internal ACK followed by an external FIN-ACK (FA) packet, an attacker can circumvent and send a DATA packet. It is surprising that an ACK transitions the connection state without a SYN packet! This is the largest cluster and again has many variants (not shown).
- *C2+C4) Internal* SYN *and variants.* The second and the fourth clusters entail using an internal SYN packet followed by non-traditional packet sequences. Figure 13c shows one shortest example and Figure 13d shows an attack of length 7. This is interesting as for the other 3 FWs, having the first SYN was a requirement but for FW-2, this is just 2 clusters out of 4.
- *C3) External TCP packets with* ACK *flags and variants.* This cluster involves a first TCP packet with an ACK flag (e.g., a DATA packet with an ACK bit or an ACK packet). The shortest attack involves one DATA packet (Figure 13e), but there are numerous variants involving a range of lengths.

FW-3: We identify 7 clusters of sizes 7,621, 212, 198, 63, 58, 37, and 31, respectively.

- *C1+C4) Incomplete handshake and variants.* The attacks in these clusters exploit a connection state being disrupted



(a) **Incomplete handshake** (C1) (b) SYN **retries** (C2+C3) (c) SYN **+** ACK (C5) (d) **Simultaneous open** (C6) (e) **Multiple** SYN (C7)

Figure 14: **Evasion attacks against FW-3 across 7 clusters**

after an incomplete handshake. Figure 14a shows an example where after the initial SYN and SYN-ACK exchanges, a FW seeing a FIN-ACK packet leads to a circumvention. Cluster 4 is also a special case where the packet disrupted an incomplete handshake is a FIN-ACK packet (Figures not shown).

- *C2+C3)* SYN *retries + an external* SYN-ACK *and variants.* The attacks in these clusters exploit a connection state being disrupted. Figure 14b shows an example where after SYN retries, followed by an external SYN-ACK packets and other TCP packets lead to a circumvention.
- *C5) Internal* SYN+ACK *(optional) variants.* The shortest attack in this sequence is identical to that of a FW-1's attack. Specifically, after a SYN followed by a SYN-ACK packet, the FW-3 FW allows an external DATA packet.

  Other attacks in this cluster exploit a combination of an internal SYN and ACK packets. Figure 14c shows such an example. This cluster is quite interesting as these attacks are neither simultaneous open, SYN retries nor an incomplete handshake, but rather some strange packet combinations.
- *C6) Simultaneous Open and Variants.* Figure 14d shows an example that only involves 3 attack packets. That is, after the SYN exchanges, the FW directly allows an external DATA packet. This is in contrast with the attacks against FW-1 (Figure 11c) and FW-4 (left of Figure 16) exploiting simultaneous sequence; these require longer sequences. However, in the case of FW-3, only after SYN exchanges, an external DATA packet is allowed! There are many variants that also required a longer attack path (now shown).
- *C7) Multiple* SYN *packets and Variants:* Attacks in this cluster involve multiple SYN packets in both directions. Figure 14e shows such an example. Explaining this fully is outside our scope; we posit that each packet is responsible for affecting the connection state, and hence, critical in enabling an attack.

FW-4: We clustered FW-4 attacks using 3 clusters. From

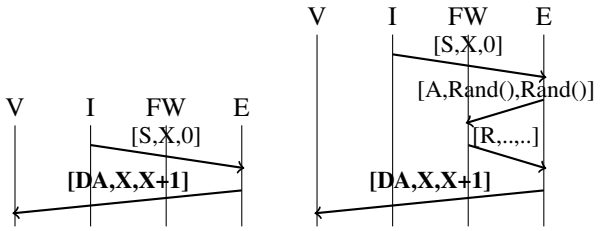
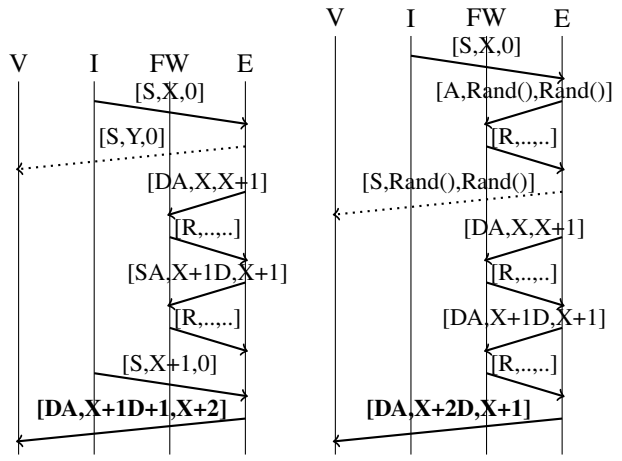


Figure 15: **Evasion attacks against FW-4 exploiting** SYN **+ (optional)** ACK **from (C1)**

294 attacks, we learned 3 clusters of sizes 199, 84, and 11, respectively.

- *C1)* SYN+ *(optional)* ACK *and variants.* Many attacks in this cluster contain some combination of internal SYN and ACK packets. Some also exploit an incomplete handshake (Left side of Figure 15). After the initial SYN packet from an insider, the FW-4 FW forwards a DATA packet from an attacker to otherwise an unreachable victim. The right side of Figure 15 shows another attack where 3 packets are injected. That is, after the initial SYN followed by an ACK from an attacker, the FW-4 FW replies with a RST packet. However, an attacker can send a DATA packet. This was flagged as a distinct attack from the previous one as the state space traversed differs.
- *C2) Simultaneous open and variants.* Attacks here exploit the simultaneous open mechanism (Figure 16a). Again, the shortest attack length is 6, indicating the subtlety required (and contrasting with FW-3 which had an attack sequence length of 3 as shown in Figure 14d). Interestingly, a FW-4 FW sends RST packets when it sees unexpected TCP packets (unlike, FW-1, for instance).
- *C3)* SYN+ *multiple* DATA *and variants.* The last cluster is interesting in that these attacks use multiple DATA packets (Figure 16b). The intermediate DATA packets are required to drive the connection state but are dropped by a FW (who replies with a RST packet). However, eventually, the FW allows the third attempt! While omitted for brevity, we find

multiple variants.



(a) **Simultaneous open** (C2) (b) SYN **+ multiple** DATA (C3)

Figure 16: **Evasion attacks against FW-4 from (C2) and (C3)**

7 Related Work

Network testing and verification: There is a large body of works (e.g., [25, 34, 39]) on network testing and verification. Many use network function (NF) models to guide testing and verification. Unfortunately, the models of these NFs are hand-generated, and using a low-fidelity NF model can affect the effectiveness of verification tools [33]. To address this issue, *Alembic* [33] uses L*-based workflow to automatically synthesize high-fidelity NF models. However, *Alembic* makes certain design choices as it assumes that TCP packets are adhering to the TCP semantics. Hence, the toolcannot directly be applied to synthesizing evasion attack strategies for FWs.

Firewall policy checking: Prior works (e.g., [18, 20, 42]) have also focused on testing and verifying the correctness of firewall ruleset. Many of them use abstractions such as Binary Decision Diagrams (FIREMAN [42]) or a directed graph [19] to verify the FW ruleset. Further, the work by El-Atawy et al. [18] generates test cases to identify misconfiguration by designing a mechanism to reduce the search space. At a high-level, these works are orthogonal to *Pryde* since they do not focus on evasion attacks enabled by implementation errors in stateful semantics but on "how" the rules are configured.

Protocol fuzzer: Fuzz testing has been applied to reverse-engineer or find bugs in protocol implementations. It is a well-established knowledge that fuzzing purely randomly for stateful protocol is not going to work [26]. Hence, there have been many works in stateful network fuzzers (e.g., [17,26,27]). The closest to our work [26, 27] extracts state semantics from a protocol implementation via black-box analysis and testing. Unfortunately, these focus on specifically inferring the stateful semantics from protocol implementation, which is a different problem from inferring the model for a FW [33]. Further, these require having a priori access to network traces for inference [26] and suffer from the same limitations as the FW model inference tool such as *Alembic*. Specifically, these also cannot reason about potentially adversarial scenarios and do not discover evasion strategies.

Side-channel attacks on TCP: Other works have looked at exploiting TCP semantics to launch attacks. Chen et al. [36] abused a side channel in the Wi-Fi half-duplex implementation to launch TCP injection attacks against major operating systems (OS). To do so, the authors use vulnerabilities to first infer existing TCP connections and poison these connections. Similarly, [37] discovered off-path TCP sequence number inference that can hijack a TCP connection to inject malicious content. We find that their focus is different from ours as they focus on side-channel attacks whereas our goal is to evade the detection and prevention of a FW.

Censorship evasion: A seemingly plausible solution is to directly use the tools that have reported evasion attacks for censorship FWs [22, 41] and/or IDS [35]. For instance, INTANG [41] manually validates hypotheses to evade the Great Firewall of China (GFW). Unfortunately, this approach will not scale if the inner-working of GFW changes. Similarly, prior work on IDS [35] also has similar issues of doing manual analysis. A recent work, Geneva [22], uses genetic algorithms to automate this process for censorship FWs. However, the system model that they consider has different goals and configuration settings and hence, cannot directly be applied (§6.1). Further, with its model-free approach, the tool is unaware of the internal FW states, which are useful to discover subtle, but semantically-different attacks. However, such model-free approaches (e.g., genetics algorithms) may provide complementary tool-set that can help the model-guided approach to be more efficient (more in §8).

Model-guided attack generation: Other prior works (e.g., [29,30,32]) were successful in using a model-guided approach to generate attacks in other settings. For instance, [30] generates attacks exploiting TCP congestion control mechanism to degrade throughput. LTEInspector [29] discovers designs flaws of the 4G LTE protocol.

8 Discussion

Countermeasures: In the short term, we envision two fixes. First, vendors could use our generated models and attack strategies and identify bug fixes. While fixing such logic bugs completely may be difficult, as a starting point, we could focus on fixing bugs up to a length X. Here, the models learned from *Pryde* could help narrow down the exact code region/path for any given attack sequence. We envision doing this iteratively; i.e., after patching specific vulnerable sequences, we can rerun

Pryde to validate/or identify new evasion attacks. Second, operators can use our attacks to synthesize policies for traffic normalizers [28]. Some of our post-processing analysis for summarizing the patterns of attacks may help in this process (e.g., generating signatures). A longer-term option would perhaps be to use some type of program synthesis or formal verification techniques to generate the FSM handling parts of the FW that are correct by construction (e.g., [43]).

Handling more complex victims: Our current victim model is simple and accepts all TCP packets. As future work, we can consider more complex attacks (e.g., data exfiltration from a stateful server). To handle this, we would need the attacker to establish a TCP connection with the victim. Thus, we would need we need additional logic to adjust the seq and ack numbers of the generated strategies and ensure that an attacker can establish a connection with a victim.

Extending model inference: We currently consider four interference input alphabets. With the current templates, we could discover hundreds to thousands of distinct attack strategies. We could consider additional input alphabets that would involve bad checksum, low TTL, among others, which would help to discover additional attack strategies. Here it might be useful to combine our approach with model-free efforts [22] since they can quickly explore more TCP fields.

Handling application-layer attacks: We focus on TCP-level evasion in this paper. An interesting future direction is to extend our design to handle layer-7 FWs. To do so, we would need to extend both the model inference (e.g., HTTP GET, POST) and the attack synthesis steps.

9 Conclusions

Stateful firewalls are the "workhorse" of operational network security but are surprisingly hard to implement correctly. As such, vulnerabilities in the semantics of the stateful processing can lead to fundamental sources of evasion attacks that can manifest even if the policies are configured correctly. Our work on *Pryde* automatically synthesizes evasion strategies with a model-guided approach by taking as input only a black-box FW implementation. *Pryde* is extensible and can be used for analyzing a variety of scenarios, though in this work as a starting point we focus on the circumventing a DATA to a victim host. Our analysis of multiple production-grade firewalls reveals that: 1) there are more than hundreds (or a thousand for some cases) of distinct attack sequences for each FW; and 2) these attacks are subtle that would be difficult to discover if we had done them manually.

References

[1] AWS Marketplace. https://aws.amazon.com/marketplace, last accessed July 31, 2020.

[2] boofuzz: Network protocol fuzzing for humans. https://boofuzz.readthedocs.io/en/latest/, last accessed July 31, 2020.

[3] Cloud-based firewalls are key to protecting employees while working remotely. https://securityboulevard.com/2020/05/cloud-based-firewalls-are-key-to-protecting-employees-while-working last accessed July 31, 2020.

[4] Cloudlab. https://www.cloudlab.us/, last accessed July 31, 2020.

[5] FBI recommends that you keep your IoT devices on a separate network. https://www.zdnet.com/article/fbi-recommends-that-you-keep-your-iot-devices-on-a-separate-network last accessed July 31, 2020.

[6] Firewall Penetration Testing: Steps, Methods And Tools That Work. https://purplesec.us/firewall-penetration-testing/, last accessed July 31, 2020.

[7] IoT security fail: The weird devices that employees are connecting to the office network. https://www.zdnet.com/article/iot-security-warning-employees-are-connecting-these-unauthorised-de last accessed July 31, 2020.

[8] Katherine Pryde. https://marvel-movies.fandom.com/wiki/Katherine_Pryde, last accessed July 31, 2020.

[9] Microsoft: Russian state hackers are using IoT devices to breach enterprise networks. https://www.zdnet.com/article/microsoft-russian-state-hackers-are-using-iot-devices-to-breach-ent last accessed July 31, 2020.

[10] Red Hat Sprucing OpenShift for Network Functions on Kubernetes. https://www.lightreading.com/nfv/red-hat-sprucing-openshift-for-network-functions-on-kubernetes/d/d-id/754828, last accessed July 31, 2020.

[11] Rogue IoT devices are putting your network at risk from hackers. https://www.zdnet.com/article/rogue-iot-devices-are-putting-your-network-at-risk-from-hackers/, last accessed July 31, 2020.

[12] Scapy. http://www.secdev.org/projects/scapy/, last accessed July 31, 2020.

[13] Sulley: Fuzzing Framework. http://www.fuzzing.org/wp-content/SulleyManual.pdf, last accessed July 31, 2020.

[14] The Importance of Using a Firewall for Threat Protection. https://www.websecurity.digicert.com/security-topics/importance-using-firewall-threat-protection, last accessed July 31, 2020.

[15] The IoT: Gateway for enterprise hackers. https://www.csoonline.com/article/3148806/the-iot-gateway-for-enterprise-hackers.html, last accessed July 31, 2020.

[16] Virtualbox. https://www.virtualbox.org/, last accessed July 31, 2020.

[17] Abdelnur, H. J., State, R., and Festor, O. Kif: a stateful SIP fuzzer. In *IPTComm* (2007), ACM, pp. 47–56.

[18] Adel El-Atawy, Ibrahim, K., Hamed, H., and Ehab Al-Shaer. Policy segmentation for intelligent firewall testing. In *NPSec* (2005), pp. 67–72.

[19] Adiseshu, H., Suri, S., and Parulkar, G. M. Detecting and resolving packet filter conflicts. In *INFOCOM* (2000), IEEE Computer Society, pp. 1203–1212.

[20] Al-Shaer, E., El-Atawy, A., and Samak, T. Automated pseudo-live testing of firewall configuration enforcement. *IEEE J. Sel. Areas Commun. 27*, 3 (2009), 302–314.

[21] Angluin, D. Learning regular sets from queries and counterexamples. *Inf. Comput. 75*, 2 (1987), 87–106.

[22] Bock, K., Hughey, G., Qiang, X., and Levin, D. Geneva: Evolving censorship evasion strategies. In *ACM Conference on Computer and Communications Security* (2019), ACM, pp. 2199–2214.

[23] CLARKE, E. M., BIERE, A., RAIMI, R., AND ZHU, Y. Bounded model checking using satisfiability solving. *Formal Methods in System Design 19*, 1 (2001), 7–34.

[24] DE MOURA, L. M., AND BJØRNER, N. Z3: an efficient SMT solver. In *TACAS* (2008), vol. 4963 of *Lecture Notes in Computer Science*, Springer, pp. 337–340.

[25] FAYAZ, S. K., YU, T., TOBIOKA, Y., CHAKI, S., AND SEKAR, V. BUZZ: testing context-dependent policies in stateful networks. In *NSDI* (2016), USENIX Association, pp. 275–289.

[26] GASCON, H., WRESSNEGGER, C., YAMAGUCHI, F., ARP, D., AND RIECK, K. Pulsar: Stateful black-box fuzzing of proprietary network protocols. In *SecureComm* (2015), vol. 164 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer, pp. 330–347.

[27] GORBUNOV, S., AND ROSENBLOOM, A. Autofuzz: Automated network protocol fuzzing framework. *IJCSNS 10*, 8 (2010), 239.

[28] HANDLEY, M., PAXSON, V., AND KREIBICH, C. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *USENIX Security Symposium* (2001), USENIX.

[29] HUSSAIN, S. R., CHOWDHURY, O., MEHNAZ, S., AND BERTINO, E. Lteinspector: A systematic approach for adversarial testing of 4g LTE. In *NDSS* (2018), The Internet Society.

[30] JERO, S., HOQUE, M. E., CHOFFNES, D. R., MISLOVE, A., AND NITA-ROTARU, C. Automated attack discovery in TCP congestion control using a model-guided approach. In *NDSS* (2018), The Internet Society.

[31] KAZEMIAN, P., VARGHESE, G., AND MCKEOWN, N. Header space analysis: Static checking for networks. In *NSDI* (2012), USENIX Association, pp. 113–126.

[32] LI, C., TU, G., PENG, C., YUAN, Z., LI, Y., LU, S., AND WANG, X. Insecurity of voice solution volte in LTE mobile networks. In *ACM Conference on Computer and Communications Security* (2015), ACM, pp. 316–327.

[33] MOON, S., HELT, J., YUAN, Y., BIERI, Y., BANERJEE, S., SEKAR, V., WU, W., YANNAKAKIS, M., AND ZHANG, Y. Alembic: Automated model inference for stateful network functions. In *NSDI* (2019), USENIX Association, pp. 699–718.

[34] PANDA, A., LAHAV, O., ARGYRAKI, K. J., SAGIV, M., AND SHENKER, S. Verifying reachability in networks with mutable datapaths. In *NSDI* (2017), USENIX Association, pp. 699–718.

[35] PTACEK, T. H., AND NEWSHAM, T. N. Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. rep., SECURE NETWORKS INC CALGARY ALBERTA, 1998.

[36] QIAN, Z., AND MAO, Z. M. Off-path TCP sequence number inference attack - how firewall middleboxes reduce security. In *IEEE Symposium on Security and Privacy* (2012), IEEE Computer Society, pp. 347–361.

[37] QIAN, Z., AND MAO, Z. M. Off-path tcp sequence number inference attack - how firewall middleboxes reduce security. *2012 IEEE Symposium on Security and Privacy* (2012), 347–361.

[38] RAFFELT, H., AND STEFFEN, B. Learnlib: A library for automata learning and experimentation. In *FASE* (2006), vol. 3922 of *Lecture Notes in Computer Science*, Springer, pp. 377–380.

[39] STOENESCU, R., POPOVICI, M., NEGREANU, L., AND RAICIU, C. Symnet: Scalable symbolic execution for modern networks. In *SIGCOMM* (2016), ACM, pp. 314–327.

[40] UTTING, M., AND LEGEARD, B. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[41] WANG, Z., CAO, Y., QIAN, Z., SONG, C., AND KRISHNAMURTHY, S. V. Your state is not mine: a closer look at evading stateful internet censorship. In *Internet Measurement Conference* (2017), ACM, pp. 114–127.

[42] YUAN, L., MAI, J., SU, Z., CHEN, H., CHUAH, C., AND MOHAPATRA, P. FIREMAN: A toolkit for firewall modeling and analysis. In *IEEE Symposium on Security and Privacy* (2006), IEEE Computer Society, pp. 199–213.

[43] ZAOSTROVNYKH, A., PIRELLI, S., PEDROSA, L., ARGYRAKI, K. J., AND CANDEA, G. A formally verified NAT. In *SIGCOMM* (2017), ACM, pp. 141–154.

A Size of the inferred models

	FW-1		FW-2		FW-3		FW-4	
Template	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
Baseline	8	23	3	12	4	56	2	2
IX^{dir}	14	27	14	15	4	63	2	3
IX^{dir}_{rand}	11	36	5	16	10	63	57	N/A
IX^{dir}_{conn}	15	50	7	11	10	78	30	N/A
IX_{rand}	10	10	3	11	18	62	55	247

Table 2: Number of states for inferred models (N/A means not converged); (1) involves only connection setup, DI, and (2) involves teardown packets, DI-T