On the Completeness of Attack Mutation Algorithms

Shai Rubin, Somesh Jha, and Barton P. Miller
University of Wisconsin, Madison
Computer Sciences Department
{shai,jha,bart}@cs.wisc.edu

Abstract

An attack mutation algorithm takes a known instance of an attack and transforms it into many distinct instances by repeatedly applying attack transformations. Such algorithms are widely used for testing intrusion detection systems. We investigate the notion of completeness of a mutation algorithm: its capability to generate all possible attack instances from a given set of attack transformations.

We define the notion of a Φ-complete mutation algorithm. Given a set of transformations Φ, an algorithm is complete with respect to Φ, if it can generate every instance that the transformations in Φ derive. We show that if the rules in Φ are uniform and reversible then a Φ-complete algorithm exists. Intuitively speaking, uniform and reversible transformations mean that we can first exclusively apply transformations that simplify the attack, then exclusively apply transformations that complicate it, and still get all possible instances that are derived by the rules in Φ.

Although uniformity and reversibility may appear severe restrictions, we show that common attack transformations are indeed uniform and reversible. Therefore, our Φ-complete algorithm can be incorporated into existing testing tools for intrusion detection systems. Furthermore, we show that a Φ-complete algorithm is useful, not only for testing purposes, but also for determining whether two packet traces are two different mutations of the same attack.

1. Introduction

The goal of a network intrusion detection system (NIDS) is to detect malicious activities, or attacks, on the network. A misuse NIDS defines an attack via an attack signature, typically a regular expression that matches a pattern of the attack [18, 23]. Ideally, each time an ongoing activity matches an attack signature, the NIDS raises an alarm.

Conceptually, a NIDS signature corresponds to a single attack, a sequence of events that exploits a given vulnerability. In practice, however, a signature should match many equivalent attack forms, or *attack instances*. For example, the same attack can be split into TCP or IP packets in many different ways. Therefore, the reliability of a NIDS ultimately depends on its ability to detect any instance of a given attack. Unfortunately, researchers (and attackers) have successfully evaded many NIDS by mutating an attack instance that the NIDS recognizes into an instance that it misses. For example, to evade a NIDS that only uses a signature of ASCII characters, they used the *URL encoding* transformation that replaces the ASCII characters of a URL with their equivalent hexadecimal values [8, 29].

To increase NIDS reliability, NIDS developers should test the NIDS against as many attack instances as possible. To generate many instances of the same attack, developers commonly use an *attack mutation system* [13, 17, 24, 25, 29]. Such a system usually has two components: a set of *attack transformation rules*, such as the URL encoding above, and a *mutation algorithm*. To use such a system for testing, a developer first constructs an *exemplary instance* of a given attack. Then, the developer feeds the exemplary instance to the mutation algorithm. The algorithm repeatedly applies the transformations according to some predetermined (or random) order and generates new instances of the attack for testing purposes.

Attack mutation systems have successfully uncovered vulnerabilities in various NIDS [13, 17, 21, 24, 29]. However, to the best of our knowledge, the fundamental question underlying NIDS testing is yet to be investigated. This question is the *testing coverage question*: which instances, out of all the instances that can be derived by the rules, does the mutation algorithm generate and which does it miss?

We address the coverage question: we develop a Φ-*complete* mutation algorithm. Given an exemplary attack instance and a set of transformations Φ, a mutation algorithm is Φ-complete if it can generate all the instances, up to a given length k, that are derived from the exemplary instance using the rules in Φ.

Two observations should be noted about a Φ-complete mutation algorithm. First, an algorithm that exhaustively applies the rules in all possible combination is not necessarily Φ-complete. The problem is that it is unclear when to

stop the generation, because instances that are longer than k might eventually derive instances that are shorter than k.

Second, a Φ-complete algorithm does **not** necessarily generate **all possible** instances of a given attack. It can do so, theoretically at least, if we assert that the rules in Φ represent all possible ways to transform the attack. Nevertheless, the ability to prove that an algorithm is Φ-complete is the first step toward a mutation system that generates all possible instances of an attack. Having said so, however, determining whether a system contains all possible transformations is beyond the scope of this paper.

Achieving Φ-completeness. To achieve Φ-completeness, our algorithm requires that the rules in Φ are *reversible* and *uniform*. Reversibility means that each transformation in our system has a corresponding inverse. Uniformity means that if an attack instance σ derives an instance τ, then there exists a derivation from σ to τ in which we first simplify σ as much as possible and then complicate the result until we reach τ. We define "simplify" and "complicate" using a novel complexity metric for attack instances. For example, we say that an attack instance that contains HEX encoding is more complex than an instance that does not contain such encodings.

We show that when the rules in Φ are uniform and reversible, the instances that Φ derives can be derived from a few representative instances, called *atoms*. We prove that atoms split attack instances into equivalence classes: two instances are in the same class if and only if they are derived from the same atom. Using this property, we developed a two-phase mutation algorithm. Given an attack instance, we first automatically compute its atom; then, we generate all instances that are derivable from this atom.

We also develop the *union property* for preserving reversibility and uniformity of two sets of transformations. Given Φ_1 and Φ_2, where each set is uniform and reversible, we show that if Φ_1 and Φ_2 are *positively commutative* (as defined Section 4.4), then $\Phi_1 \cup \Phi_2$ is also uniform and reversible. Practically speaking, this property helps us prove the uniformity and reversibility of a large set of rules. For example, we develop one set of uniform and reversible rules for TCP and one for HTTP. We use the union property to show that the union of the two sets is uniform and reversible and therefore our algorithm is Φ-complete with respect to our TCP and HTTP rules.

Other usages of a Φ-complete algorithm. During NIDS development we usually encounter the *forensics problem*: given a set of rules, determine whether two attack instances are derived from each other. This problem arises when we need to determine whether a trace of packets is an instance of a known attack. As we show in Section 4, a Φ-complete algorithm can be used to assert whether two instances are derived from each other. When a Φ-complete algorithm asserts that two instances are not derived from each other, we know that only one of the two following options are possible. First, the instances are derived from each other but our algorithm does not use the transformations that were used to derive the instances. In this case, the algorithm helps us uncover a new transformation. Second, the instances are not derived from each other; in that case, the algorithm helps us define a new attack. Although the distinction between the two cases requires manual intervention, note that an incomplete algorithm is even less useful because it introduces a third case in which the instances are derivable from each other but the algorithm was not able to determine that fact.

We show that a Φ-complete algorithm can efficiently solve the forensics problem when the rules in Φ are uniform and reversible. Given two instances, σ and τ, the algorithm first computes the atom of σ and then checks whether τ can be derived from this atom. The correctness of this algorithm stems from the fact that two instances are derivable from each other if and only if they have the same atom.

In summary, this paper makes the following contributions:

1. **The notion of Φ-complete attack mutation algorithm.** Such an algorithm addresses the coverage question which is the core of any rigorous NIDS testing process.
2. **Conditions for Φ-completeness.** We develop the notion of uniformity and reversibility for attack transformations. We show how to use these concepts to prove that our proposed algorithm is Φ-complete. We also develop the union property that helps proving the uniformity of a union of two sets of transformations.
3. **A practical instance of a uniform and reversible set of transformations.** We show that common attack transformations are uniform and reversible. Our set of rules include transformations like TCP-fragmentation, TCP-permutation, and TCP-retransmission as well as application-level transformation like the URL encoding.

2. Related Work

We review related work in the areas of attack transformations, NIDS testing, abstract reduction systems, and using uniform proofs in logic programming.

Attack transformations. Fundamentally, network attacks can be modified, or transformed, at any level of the protocol stack. Ptacek and Newsham [20, 21] as well as Handley and Paxson [10, 18] were the first to introduce IP and TCP transformations (e.g., fragmentation, packet reordering).

Based on their work, tools that use attack transformations for NIDS testing, or evasion, have been developed. Fragroute, which transforms TCP-based attacks [28], and Whisker, which transforms HTTP attacks [22], randomly combine transformations specified by the user. Mucus [17]

uses attack transformations to perform cross-testing of two NIDS: it builds packets that match a signature of the first NIDS, transforms them, and checks whether the other NIDS identifies the modified packets. Recently, Vigna et al. [29] developed a tool that applies application-level transformations (e.g., HTTP encoding, injection of Telnet escape characters) in addition to TCP/IP transformations. Other testing tools that are based on attack transformations are Snot [27], Stick [9], and Thor [1, 13].

The tools mentioned above successfully found attack instances that evade the NIDS they had tested. However, to the best of our knowledge, the researchers that developed these tools have not addressed the completeness question. Recently, Rubin et al. [24] developed a tool called AGENT that exhaustively applies transformation rules in all possible combinations. However, they did not provide formal proof that AGENT can really generate all possible instances. Indeed, as we argue in Section 4.1, exhaustiveness does not guarantee completeness.

Dacier et al. [7] use attack mutation to evaluate the potential of a set of different IDSs to handle a large set of transformations. However, unlike our work here, they did not investigate the completeness property of their system.

Reduction systems and uniform proofs. Our formal methodology is closely related to abstract reduction system [2]. A reduction system is a pair $(A, \rightarrow)$, where $x \rightarrow y$ is a binary relation such that $x, y \in A$. In our case, x and y are attack instances and the relation $\rightarrow$ is defined using transformation rules.

However, to the best of our knowledge, a classic reduction system does not distinguish between shrinking (i.e., simplifying) and expanding (i.e., complicating) rules. Hence, the general results for such systems cannot be used unmodified. For example, our concept of an atom (Section 4.2.2) is equivalent to the concept of a normal form in lambda calculus [3]: an element x that cannot be further reduced (i.e., there is no y such that $x \rightarrow y$). However, an atom is an element that cannot be reduced using shrinking rules only, while a uniform form in lambda calculus cannot be reduced by any rule. Also note that every instance in our reduction system is strongly normalized with respect to our shrinking rules, that is every instance has an atom.

Miller et al. [15] describe uniform proofs where right-introduction rules, which are analogous to shrinking rules (Section 4.2.1), appear before left-introduction rules, which are similar to expanding rules. The main intuition behind introducing uniform proofs was to capture goal-directed search. They also proved that in the framework of logic programming uniform proofs are complete, i.e., if a term is provable then it has an uniform proof. Uniform proofs have also been in explored in other contexts [11, 26]. Special structures of derivation also have also been used in security-protocol verification [4, 5, 6, 14]. To our knowledge, our paper is the only work that explores uniform derivations as the basis for generating attack mutations for NIDS testing.

3. Technical Overview

We use an exemplary attack to demonstrate the fundamental concepts of attack mutation that we use later in the paper: transformations, mutation algorithm, atoms, and uniform derivation.

The *perl-in-cgi* exploit (CAN-1999-0509 [16]): a Perl interpreter is installed in the `cgi-bin` directory on a Web server, allowing remote attackers to execute arbitrary commands.

Attack transformations. Consider an instance of *perl-in-cgi*, denoted σ, that contains a single HTTP `GET` request: "`GET <web page>/cgi-bin/perl.exe`". Assume that σ uses a single TCP segment (not including the TCP handshake segments). Consider the following transformation rules that we can use to create other instances of *perl-in-cgi* from σ:

1. *frag*$^+$ (TCP-fragmentation): if τ is obtained from σ by copying σ's segments, or TCP packets, and then fragmenting a single segment into two segments, then τ is an instance of *perl-in-cgi*.
2. *url*$^+$ (URL encoding): if τ is obtained from σ by replacing a printable character in σ's URL with its hexadecimal ASCII value, then τ is an instance of *perl-in-cgi*.
3. *http-pipe*$^+$ (HTTP pipelining): if τ is obtained from σ by inserting a benign HTTP `GET` request (e.g., "`GET <web page>/index.html`") before the malicious `GET` request, then τ is an instance of *perl-in-cgi*.

Denote the set of the three rules as Φ_3. We say that an instance τ is *derivable from σ with respect to* Φ_3 if τ is the result of applying a rule from Φ_3 on σ. Naturally, we extend the definition of derivability to a sequence of rule applications.

An attack mutation algorithm generates many instances of *perl-in-cgi* by repeatedly applying the rules in Φ_3 on the initial instance σ. We say that the algorithm is *complete with respect to* Φ_3, denoted Φ_3-complete, if it can generate all (up to a certain length k) instances that are derived from σ using the rules in Φ_3. Notice that a Φ_3-complete algorithm does **not** generate all possible instances of *perl-in-cgi*, but only the instances that can be derived using the rules in Φ_3. For example, a Φ_3-complete algorithm will not generate instances that are based on other TCP transformations, such as TCP-retransmission.

Partial order of attack instances. It is clear that *frag*$^+$, *url*$^+$, and *http-pipe*$^+$ can be used to complicate σ: we can add arbitrary benign HTTP commands, obfuscate URLs, and fragment σ into smaller TCP segments. At the same time, the impact of the rules is *reversible*: we can undo *frag*$^+$ by merging TCP segments, undo *url*$^+$ by normalizing

URL to only use printable characters, and undo *http-pipe*$^+$ by removing benign HTTP requests.

Thus, a transformation has two forms: an *expanding* form that complicates an instance and a *shrinking* form that simplifies it. Given an arbitrary attack instance, an attack mutation algorithm should use both expanding and shrinking transformations to generate all possible instances. We denote the shrinking, or reverse, versions of *frag*$^+$, *url*$^+$, and *http-pipe*$^+$ as *frag*$^-$, *url*$^-$, and *http-pipe*$^-$, respectively.

Expanding and shrinking transformations imply a partial order over the instances of *perl-in-cgi*. The length (in bytes) of an instance can be used to rank the instance complexity: the longer the instance the higher its complexity. Note that *frag*$^+$, *url*$^+$, and *http-pipe*$^+$ increase instance complexity, while *frag*$^-$, *url*$^-$, and *http-pipe*$^-$ reduce it. (*frag*$^+$ increases the complexity because each additional TCP segment requires an additional TCP header.)

Atoms. Intuitively, the instance σ is atomic. First, we cannot shrink σ any further because it uses a single TCP segment, does not include benign HTTP requests, and contains only printable characters. Second, σ is the simplest form of the attack, any byte in σ is required for a successful attack. Third, with respect to our rules, σ is the building block of all other instances. Using expanding rules alone, σ derives any *perl-in-cgi* instance that is fragmented into several (non-overlapping) TCP segments, contains benign HTTP commands, and its URLs use either printable characters or their hexadecimal ASCII values.

A uniform derivation. In a uniform derivation all shrinking transformations precede all expanding ones. As we discuss in the next section, to prove that our proposed mutation algorithm is Φ_3-complete, we need to show that if σ derives τ, then there is also a uniform derivation from σ to τ. For example, it is easy to see that if we first expand an instance by fragmenting it (i.e., using *frag*$^+$) and then replacing an hexadecimal ASCII value with a printable character (i.e., using *url*$^-$), then it is possible to first replace the character and then to fragment the instance.

Summary of observations. Shrinking and expanding transformations correspond to our intuition that we can simplify or complicate attack instances. Atoms correspond to our intuition that some attack instances cannot be simplified any further and these instances are the building blocks for other attack instances. Uniformity corresponds to our intuition that it is possible to derive all instances from a given instance by first simplifying the instance as much as possible, using shrinking rules, and then only use expanding rules to generate all instances.

4. Achieving Φ-Completeness

Our goal in this section is to develop a Φ-complete attack mutation algorithm. To do so, we first formally define transformation rules, a mutation algorithm, and the notion of Φ-completeness (Section 4.1). We also discuss why, for a general set of rules, an algorithm that recursively applies the rules is unlikely to be Φ-complete. Next, we discuss the reversibility and uniformity of transformations (Section 4.2) and prove that our proposed algorithm is Φ-complete **if** Φ is reversible and uniform (Section 4.3). Last, we develop the *union property* that states the necessary conditions under which a union of two sets of rules is uniform and reversible.

Notice that the discussion in this section **does not** imply that every set of transformations is uniform and reversible. Our goal is just to reveal the properties necessary for proving Φ-completeness. While an arbitrary definition of transformations is unlikely to have these properties, we show that common TCP transformations (Section 5) and HTTP transformations (Section 6) can be defined such that they are indeed uniform and reversible.

4.1. An Attack Mutation Algorithm

In this section we model attack instances as strings over the alphabet Σ.

Let Σ be an alphabet set, $\Sigma^\star$ be the set of strings over Σ, and $\Sigma^k \subseteq \Sigma^\star$ be the set of strings of length $\leq k$. A transformation rule r has the following form:

$$\frac{\sigma, pre(\sigma)}{\sigma', post(\sigma, \sigma')}$$

where σ and σ' are strings over Σ, and pre and $post$ are predicates. The rule is interpreted as follows: if a string σ satisfies the predicate pre, then σ' is derivable from σ provided that $post(\sigma, \sigma')$ is true. If a string σ' can be derived from σ using a rule r, we write it as $\sigma \xrightarrow{r} \sigma'$.

Let Φ be a set of transformation rules. We say that a string σ' is derivable from σ with respect to Φ, denoted $\sigma \overset{\Phi}{\Rightarrow} \sigma'$, if and only if there exists a sequence of rules $\langle r_1, \ldots, r_k \rangle$ in Φ, called a *derivation*, such that $\sigma \xrightarrow{r_1} \sigma_1 \xrightarrow{r_2} \ldots \sigma_{k-1} \xrightarrow{r_k} \sigma_k = \sigma'$. Given a string σ and a set of rules Φ, the *closure* of σ with respect to Φ, denoted $Cl_\Phi(\sigma)$, is the set of strings that are derivable from σ. Formally, $Cl_\Phi(\sigma) = \{\sigma' \mid \sigma \overset{\Phi}{\Rightarrow} \sigma'\}$. Given a finite set of strings $S \subseteq \Sigma^\star$, its closure $Cl_\Phi(S)$ is given by $\bigcup_{\sigma \in S} Cl_\Phi(\sigma)$.

A *mutation algorithm*, denoted *MA*, takes a finite set of strings $S \subseteq \Sigma^\star$ and returns another set of strings $MA(S)$ such that $S \subseteq MA(S)$. Intuitively, a mutation algorithm takes a set of attack instances and returns a larger set of instances that are mutations of the original ones.

Definition 1 (A sound and complete mutation algorithm)**.** *Let MA be a mutation algorithm, Φ a set of transformation rules, and $S \subseteq \Sigma^\star$ a set of strings.*

- *MA is called* **sound with respect to** Φ*, denoted* Φ*-sound, if an only if for all $S \subseteq \Sigma^\star$, $MA(S) \subseteq \mathrm{Cl}_\Phi(S)$. Intuitively, MA is Φ-sound if its mutation algorithm only*

generates attack instances that are derivable from S with respect to Φ.

- *MA is called* **complete with respect to** Φ*, denoted* Φ*-complete, if and only if for all* $S \subseteq \Sigma^\star$*,* $MA(S) \supseteq \mathrm{Cl}_\Phi(S)$*. Intuitively, MA is* Φ*-complete if its mutation algorithm covers all possible strings that are derivable from* S *with respect to* Φ*.*
- *MA is called* **k-complete with respect to Φ***, denoted* Φ^k*-complete, if and only if for all* $S \subseteq \Sigma^\star$*,* $MA(S) \cap \Sigma^k \supseteq \mathrm{Cl}_\Phi(S) \cap \Sigma^k$*. Intuitively, MA is* Φ^k*-complete if its mutation algorithm covers all possible strings of length* $\leq k$ *that are derivable from* S *with respect to* Φ*.*

For practical applications, we would like to bound the number of instances that a mutation algorithm derives. Hence, our focus in the rest of this paper is on Φ^k-complete algorithms. Furthermore, since a sound mutation algorithm is trivial to construct, we take soundness for granted and do not mention this property unless required.

To better understand the difficulty in constructing a Φ^k-complete mutation algorithm, consider a standard work-list algorithm that builds a closure by recursively deriving the successors of the initial instance σ. It is difficult to determine when to terminate such a derivation process. Suppose we derive an instance σ' such that $length(\sigma') > k$. Intuitively, since σ' is too long to be included in $Cl_\Phi(\sigma) \cap \Sigma^k$, we would be inclined to believe that σ' cannot derive any instance that is part of $Cl_\Phi(\sigma) \cap \Sigma^k$. However, in a general mutation system, each rule might have an arbitrary effect. So, even though σ' is too long, it might derive a shorter instance that is part of the closure.

4.2. Uniformity and Reversibility

The difficulty in constructing a Φ^k-complete algorithm suggests that such a system requires ordering of attack instances. The goal of uniformity and reversibility is to formalize the concepts of simplifying and complicating an attack instance.

4.2.1 Uniformity and Reversibility

Let $\preceq$ be a partial order on the set $\Sigma^\star$. We say that $\sigma \prec \beta$ if and only if $\sigma \preceq \beta$ and $\sigma \neq \beta$.

Given a set of transformations Φ and a partial order $\preceq$, a rule r is called a *shrinking rule* if for all σ and σ' such that $\sigma \xrightarrow{r} \sigma'$ we have that $\sigma \succ \sigma'$. A rule r is called an *expanding rule* if for all σ and σ' such that $\sigma \xrightarrow{r} \sigma'$ we have that $\sigma \prec \sigma'$. Φ^- and Φ^+ denote subsets of Φ consisting of shrinking and expanding rules in Φ, respectively. Intuitively, shrinking rules are used to simplify an attack instance while expanding rules are used to complicate it.

A derivation $\langle r_1, \ldots, r_k \rangle$ is called *uniform* if there does not exist an $i < j$ such that r_i is an expanding rule and r_j is a shrinking rule. Alternatively, in a uniform derivation, shrinking rules are applied before expanding rules.

Definition 2 (Uniformity of Φ)**.** *Let Φ be a set of transformation rules. Φ is called* **uniform** *if there exists a partial order $\preceq$ on $\Sigma^\star$ such that the following conditions hold: (i) with respect to $\preceq$ each rule in Φ is either shrinking or expanding, and (ii) for all σ and σ' such that $\sigma \overset{\Phi}{\Rightarrow} \sigma'$, there exists a uniform derivation from σ to σ'. In other words, any derivation in Φ has a corresponding uniform one.*

Definition 3 (Reversibility of Φ)**.** *Let Φ be a set of transformation rules. Φ is called* **reversible** *if every rule in Φ has an inverse. Inverse of a rule r, denoted r^{-1}, is a rule such that for all σ and σ' holds: $\sigma \xrightarrow{r} \sigma'$ if and only if $\sigma' \xrightarrow{r^{-1}} \sigma$.*

Two important observations should be noted. First, in a uniform and reversible set of transformations each shrinking rule is the inverse of an expanding rule, and vice-versa. We use this observation when we construct a Φ-complete algorithm (Section 4.3).

Second, as already mentioned in the beginning of this section, not every set of transformations is uniform and reversible. However, in Sections 5 and 6 we show that it is possible to define common transformations used by existing mutation systems such that they are uniform and reversible.

In the rest of the paper, we assume that for a partial order used by our mutation system any descending chain is finite. A chain of attack instances $\langle \sigma_0, \sigma_1, \ldots \rangle$ is called descending if and only if $\sigma_i \succ \sigma_{i+1}$. This assumption states that shrinking rules cannot be applied infinitely, which corresponds to the fact that we cannot simplify an attack instance beyond a certain point. For example, the instance σ in Section 3 is the simplest form of the *perl-in-cgi* attack with respect to the rules we considered.

4.2.2 Computing Atoms

An atom is the simplest instance of an attack. We formalize this intuition using shrinking and expanding rules.

Given a partial order $\preceq$ and a set of transformations Φ in which each rule is either expanding or shrinking with respect to $\preceq$, a string σ is called a Φ-*atom* if there does not exist a shrinking rule r in Φ such that $\sigma \xrightarrow{r} \sigma'$: no shrinking rule from Φ can be applied to a Φ-atom. Given a string σ, the set $atoms_\Phi(\sigma)$ is the set of Φ-atoms that are derived from σ. For a finite set of strings S, the set $atoms_\Phi(S)$ is defined as $\cup_{\sigma \in S}\{atom_\Phi(\sigma)\}$.

Theorem 1. Let Φ be a set of transformations. If Φ is uniform and reversible, then for every string σ, the set $atoms_\Phi(\sigma)$ is a singleton set.

Proof of Theorem 1: Suppose there are two sequences σ_A and σ_B in the set $atoms_\Phi(\sigma)$. By definition, there are derivations $\langle r_1, r_2, \ldots, r_i \rangle$ and $\langle r'_1, r'_2, \ldots, r'_j \rangle$ from σ to σ_A and from σ to σ_B, respectively. Since Φ is reversible, $\langle r_i^{-1}, \ldots, r_1^{-1} \rangle$ is derivation from σ_A to σ and the following sequence of rules is a derivation from σ_A to

input : A string σ and
a set of uniform and reversible rules Φ.
output: $atoms_\Phi(\sigma)$ (a singleton set).

1 *currentString* $= \sigma$;
2 **while** $true$ **do**
3 **if** *a shrinking rule cannot be applied to* currentString **then** break;
4 **else** Pick a rule r from Φ^- that can be applied to *currentString*, then perform *currentString* $= r$(*currentString*);
5 **end**
6 **return** *currentString*;

Algorithm 1: Computing $atoms_\Phi(\sigma)$ for a uniform and reversible Φ. r(*currentString*) is the string obtained by applying the rule r to *currentString*.

σ_B: $\langle r_i^{-1}, \ldots, r_1^{-1}, r'_1, r'_2, \ldots, r'_j \rangle$. Hence, σ_B is derivable from σ_A. Since Φ is uniform, there is a uniform derivation $\langle r''_1, r''_2, \cdots, r''_l \rangle$ from σ_A to σ_B. There can be two cases:

1. r''_1 is a shrinking rule. Then, a shrinking rule can be applied to σ_A, violating the fact that σ_A is an atom.
2. r''_1 is an expanding rule. Since $\langle r''_1, r''_2, \cdots, r''_l \rangle$ is a uniform derivation that starts with an expanding rule, by definition all rules r''_i for $1 \leq i \leq l$ must be expanding rules. Hence, $(r''_l)^{-1}$ is a shrinking rule that can be applied to σ_B, violating the fact that σ_B is an atom. □

Algorithm 1 shows how to compute *atoms*$_\Phi(\sigma)$. Initially, the algorithm sets *currentString* to σ. Each time in the while loop, a shrinking rule r is applied to *currentString*. If a shrinking rule cannot be applied to *currentString*, the algorithm terminates.

Claim 1. Let Φ be a set of transformations and σ be a string. If Φ is uniform and reversible then Algorithm 1 computes $atoms_\Phi(\sigma)$.

Proof of Claim 1: Algorithm 1 computes only descending chains. Since we assume that any descending chain is finite (Section 4.2.1), the algorithm must terminate. It is clear that the algorithm computes an atom. Theorem 1 proves that for a uniform and reversible set of transformations, the set $atoms_\Phi(\sigma)$ is a singleton set. Hence, the algorithm computes the set $atoms_\Phi(\sigma)$. □

4.3. A Φ^k-Complete Mutation Algorithm

We show that **if** Φ is uniform and reversible **then** there exists a Φ-complete mutation algorithm.

Algorithm 2 presents a Φ-complete mutation algorithm when Φ is uniform and reversible. First, we compute $atoms_\Phi(S)$ using Algorithm 1 (Lines 3-6). Then, we apply expanding rules from Φ^+ to all sequences in $atoms_\Phi(S)$ to generate additional sequences. Notice that when a sequence

input : A set of strings S, and a set of transformations rules Φ.
output: A set of test strings.

1 *worklist* = $\emptyset$;
2 // compute $atoms_\Phi(S)$.
3 **forall** $\sigma \in S$ **do**
4 Compute *atoms*$_\Phi(\sigma)$ using Algorithm 1;
5 *worklist* $=$ *worklist* $\cup$ *atoms*$_\Phi(\sigma)$
6 **end**
7 // Compute the closure.
8 *tests* $=$ *worklist*;
9 **while** *worklist* $\neq \emptyset$ **do**
10 Pick $\alpha \in$ *worklist*;
11 *worklist* $=$ *worklist* $-$ $\{\alpha\}$;
12 Compute $M = \Phi^+(\{\alpha\}) \cap \Sigma^k$;
13 **forall** *elements* β *of* M **do**
14 **if** $\beta \notin$ *tests* **then**
15 *worklist* $=$ *worklist* $\cup$ $\{\beta\}$
16 **end**
17 **end**
18 *tests* $=$ *tests* $\cup M$;
19 **end**
20 **return** *tests*;

Algorithm 2: A mutation algorithm. Theorem 2 proves that, when Φ is uniform and reversible, this algorithm is Φ^k-complete and Φ-sound.

α is picked from the *worklist*, only its successor sequences that are of length $\leq k$ denoted $\Phi^+(\{\alpha\}) \cap \Sigma^k$, are generated.

For instance-generation purposes, we assume that $\preceq$ is *length preserving*: if $\alpha \preceq \beta$ then *length*$(\alpha) \leq$ *length*(β). As we show in Sections 5 and 6, this assumption holds for common attack transformations.

Claim 2. *Algorithm 2 terminates.*

Proof of Claim 2: We need to show that after a finite number of steps *worklist* is empty. First, notice that any new instance generated in Line 12, is added only once into *worklist*. This is because we add every newly generated instance to *tests* (Line 18) and we add an instance to *worklist* only if the instance is not found in *tests* (Line 14). Second, notice that the total number of different instances that are added into *tests* (Line 18) is bounded, because Σ^k is finite. Therefore, the number of instances that are added to *worklist* is bounded. Third, notice that in each iteration of the while loop an instance is removed from *worklist* (Line 11). Also note that each instance that is removed must exist in *tests*, so it cannot be added again. We conclude that the size of *worklist* is bounded and each iteration removes an instance, therefore *worklist* must be emptied after a finite number of iterations. □

Theorem 2. Let Φ be a set of transformations and *MA* the mutation algorithm from Algorithm 2. If Φ is uniform and reversible according to a length preserving partial order, then *MA* is Φ^k-complete and Φ-sound.

Proof of Theorem 2: Soundness of Algorithm 2 follows from the fact that we only apply rules from Φ to generate test cases. To prove Φ^k-completeness we need to show that for every $\sigma \in S$ if $\sigma \overset{\Phi}{\Rightarrow} \sigma'$ and $\mathit{length}(\sigma') \leq k$ then Algorithm 2 generates σ'. More formally, we have to show that every sequence σ' in the set $Cl_\Phi(S) \cap \Sigma^k$ is generated by Algorithm 2.

Assume $\sigma \in S$ and consider an arbitrary $\sigma' \in Cl_\Phi(\sigma) \cap \Sigma^k$. Since we assume that Φ is uniform and reversible, there is a uniform derivation from σ to σ'. Let the uniform derivation be of the following form:

$$\sigma = \sigma_0 \overset{r_1^-}{\rightarrow} \sigma_1 \cdots \overset{r_j^-}{\rightarrow} \sigma_j \overset{r_1^+}{\rightarrow} \sigma_{j+1} \cdots \overset{r_m^+}{\rightarrow} \sigma_{j+m} = \sigma'$$

Consider σ_j, the last sequence obtained after applying shrinking rules. There are two cases:

1. σ_j is an atom of σ, that is $\sigma_j \in atoms_\Phi(\sigma)$. σ' will be generated by Algorithm 2 because (i) we start derivation from $atoms_\Phi(\sigma)$ (Line 5), and (ii) since $\mathit{length}(\sigma') \leq k$ and $\preceq$ is length preserving, then $\sigma_j, \ldots, \sigma_{j+m}$ have length $\leq k$. The fact that $\preceq$ is length preserving is important because it ensures that the length of $\sigma_j \ldots \sigma_{j+m}$ is less than or equal to k and therefore the algorithm would generate all those instances, including σ'.
2. σ_j is not an atom of σ. Denote σ_A as the atom of σ. We construct a new uniform derivation that derives σ' from σ and "passes through" σ_A. Note that showing such a derivation implies that Algorithm 2 generates σ'.

   Since $\sigma \overset{\Phi}{\Rightarrow} \sigma_j$ and Φ is uniform and reversible, then $atoms_\Phi(\sigma) = atoms_\Phi(\sigma_j) = \sigma_A \in atoms_\Phi(\sigma)$ (see Claim 3 below). Hence, there exists a derivation $\sigma_j \overset{\Phi^-}{\Rightarrow} \sigma_A$ and since Φ is reversible there exists a derivation $\sigma_j \overset{\Phi^-}{\Rightarrow} \sigma_A \overset{\Phi^+}{\Rightarrow} \sigma_j$. Note that since $\preceq$ is length preserving the length of every instance in this derivation is $\leq k$. Now, insert this derivation after σ_j in the original derivation from σ to σ'. We obtained a uniform derivation that passes through a sequence in $atoms_\Phi(\sigma)$. □

Claim 3. *Let Φ be a set of transformations. If Φ is uniform and reversible and $\sigma \overset{\Phi}{\Rightarrow} \sigma'$, then $atoms_\Phi(\sigma) = atoms_\Phi(\sigma')$.*

Proof of Claim 3: According to Theorem 1, $atoms_\Phi(\sigma)$ and $atoms_\Phi(\sigma')$ are singletons. Assume by contradiction that $\sigma_A = atoms_\Phi(\sigma) \neq atoms_\Phi(\sigma') = \sigma_B$. First, note that $\sigma \overset{\Phi^-}{\Rightarrow} \sigma_A$ and therefore by reversibility we get $\sigma_A \overset{\Phi^+}{\Rightarrow} \sigma$. So, we got a derivation from σ_A to σ_B: $\sigma_A \overset{\Phi^+}{\Rightarrow} \sigma \overset{\Phi}{\Rightarrow} \sigma' \overset{\Phi^-}{\Rightarrow} \sigma_B$. Second, since Φ is uniform, we conclude that there exists a uniform derivation from σ_A to σ_B. Last, we follow the proof of Theorem 1 and show that such a uniform derivation does not exist unless either σ_A or σ_B is not an atom. □

4.4. Combining Mutation Systems

For the purpose of attack mutation, it is usually convenient to have separate sets of rules for different network protocols. For example, a set for TCP transformations and a set for HTTP transformations. Such separation facilitates a modular testing process in which we test our NIDS first against TCP transformations, then against HTTP transformations, and last against attack instances that are derived using both TCP and HTTP transformations.

Theorem 2 proved that for Φ_1 and Φ_2 that are uniform and reversible, Algorithm 2 is Φ_1^k-complete and Φ_2^k-complete, respectively. However, this does not mean that Algorithm 2 is Φ_3^k-complete where $\Phi_3 = \Phi_1 \cup \Phi_2$.

One way to prove that Algorithm 2 is Φ_3^k-complete is to prove, from scratch, the reversibility and uniformity of the transformations in Φ_3. However, we show that this is not necessary when Φ_1 and Φ_2 are uniform and reversible with respect to the same partial order **and** are *positively commutative*. This result simplifies the completeness proof of complex mutation systems because it is usually easier to prove commutativity of two sets of rules rather than the uniformity of their union.

Definition 4 (Positively Commutative Transformations)**.** *Let r^+ be an expanding rule and s^- a shrinking rule with respect to some partial order $\preceq$. We say that r^+ and s^- are **positively commutative** if for all σ and τ such that $\sigma \overset{r^+}{\rightarrow} \rho \overset{s^-}{\rightarrow} \tau$, then $\sigma \overset{s^-}{\rightarrow} \hat{\rho} \overset{r^+}{\rightarrow} \tau$.*

*Let R and S be sets of transformations. Let $\preceq$ be a partial order such that for all $r \in R$ and $s \in S$, r and s are either expanding or shirking rules with respect to $\preceq$. We say that R and S are **set-wise positively commutative** if:*

1. *For all $r^+ \in R^+$ and $s^- \in S^-$, r^+ and s^- are positively commutative; and*
2. *for all $s^+ \in S^+$ and $r^- \in R^-$, s^+ and r^- are positively commutative.*

Note that positive commutativity is a weaker condition than full commutativity. Positive commutativity does not require that if $\sigma \overset{r^+}{\rightarrow} \rho \overset{s^+}{\rightarrow} \tau$ then also $\sigma \overset{s^+}{\rightarrow} \hat{\rho} \overset{r^+}{\rightarrow} \tau$ or if $\sigma \overset{r^-}{\rightarrow} \rho \overset{s^+}{\rightarrow} \tau$ then also $\sigma \overset{s^+}{\rightarrow} \hat{\rho} \overset{r^-}{\rightarrow} \tau$.

Claim 4. *Let R and S be transformation sets that are set-wise positively commutative. Then, if $\sigma \overset{R^+}{\Rightarrow} \rho \overset{S^-}{\Rightarrow} \tau$ then also $\sigma \overset{S^-}{\Rightarrow} \hat{\rho} \overset{R^+}{\Rightarrow} \tau$ and if $\sigma \overset{S^+}{\Rightarrow} \rho \overset{R^-}{\Rightarrow} \tau$ then also $\sigma \overset{R^-}{\Rightarrow} \hat{\rho} \overset{S^+}{\Rightarrow} \tau$.*

Proof of claim 4: Intuitively, the claim expands the notion of positive commutativity to a derivation that is longer than two rules. Let $\langle r_1^+, \ldots, r_k^+, s_1^-, \ldots, s_j^- \rangle$ be a derivation. Since S and R are set-wise positively commutative, we can "shift left" s_1^- k times and get the derivation $\langle s_1^-, r_1^+, \ldots, r_k^+, s_2^-, \ldots, s_j^- \rangle$. We can repeat this shifting process and get the derivation $\langle s_1^-, \ldots, s_j^-, r_1^+, \ldots, r_k^+ \rangle$. Analogously, we prove that if $\sigma \overset{S^+}{\Rightarrow} \rho \overset{R^-}{\Rightarrow} \tau$, then also $\sigma \overset{R^-}{\Rightarrow} \hat{\rho} \overset{S^+}{\Rightarrow} \tau$. □

Notice that positive commutativity does not imply that $R \cup S$ is a uniform set of rules. Consider, for example, the derivation $\sigma \overset{R^+}{\Rightarrow} \rho_1 \overset{S^-}{\Rightarrow} \rho_2 \overset{R^-}{\Rightarrow} \tau$. Positive commutativity implies that $\sigma \overset{S^-}{\Rightarrow} \rho_1' \overset{R^+}{\Rightarrow} \rho_2 \overset{R^-}{\Rightarrow} \tau$ and this derivation is not uniform. Only if R is uniform, then the last derivation can be converted into $\sigma \overset{S^-}{\Rightarrow} \rho_1' \overset{R^-}{\Rightarrow} \rho_2' \overset{R^+}{\Rightarrow} \tau$ which is uniform. The next theorem generalizes this observation.

Theorem 3. *Let R and S be two sets of transformations rules such that (i) R and S are reversible and uniform with respect to a partial order $\preceq$, and (ii) R and S are set-wise positively commutative. Then, $\Phi = R \cup S$ is reversible and uniform with respect to $\preceq$.*

Proof of Theorem 3: It is clear that a union of transformation sets preserves reversibility. It is left to show that Φ is uniform: given a derivation $\sigma \overset{\Phi}{\Rightarrow} \tau$ there exists a derivation $\sigma \overset{\Phi^-}{\Rightarrow} \rho \overset{\Phi^+}{\Rightarrow} \tau$.

Let $\sigma \overset{\Phi}{\Rightarrow} \tau$ be a derivation from σ to τ. The proof is similar to the proof of Claim 4: we use the uniformity and commutativity of R and S to "shift" the shrinking rules to the beginning of the derivation from σ to τ.

First, we express $\sigma \overset{\Phi}{\Rightarrow} \tau$ in terms of subderivations that only use rules from either R or S (without lose of generality we assume the derivation starts with rules from R):

$$\sigma \overset{R_1}{\Rightarrow} \cdot \overset{S_1}{\Rightarrow} \cdot \ldots \cdot \overset{R_n}{\Rightarrow} \cdot \overset{S_n}{\Rightarrow} \tau \quad (1)$$

Second, since both R and S are uniform we express Derivation 1 using uniform subderivations:

$$\sigma \overset{R_1^-}{\Rightarrow} \cdot \overset{R_1^+}{\Rightarrow} \cdot \overset{S_1^-}{\Rightarrow} \cdot \overset{S_1^+}{\Rightarrow} \cdot \ldots \cdot \overset{R_n^-}{\Rightarrow} \cdot \overset{R_n^+}{\Rightarrow} \cdot \overset{S_n^-}{\Rightarrow} \cdot \overset{S_n^+}{\Rightarrow} \tau \quad (2)$$

Last, we prove by induction that Derivation 2 can be converted into the uniform derivation of the form:

$$\sigma \overset{R_1^-}{\Rightarrow} \cdot \overset{\hat{S}_1^-}{\Rightarrow} \cdot \ldots \overset{\hat{R}_n^-}{\Rightarrow} \cdot \overset{\hat{S}_n^-}{\Rightarrow} \cdot \overset{\hat{R}_1^+}{\Rightarrow} \cdot \overset{\hat{S}_1^+}{\Rightarrow}, \ldots, \overset{\hat{R}_n^+}{\Rightarrow} \cdot \overset{S_n^+}{\Rightarrow} \tau \quad (3)$$

We use the $\hat{S}$ instead of S, to indicate that the derivation might have been changed during the "shifting" process. Our induction is on the number of *uniformity violations* in Derivation 2. A uniformity violation is an occurrence of one of the following derivations: r^+ followed by s^- or s^+ followed by r^-. The induction is given in appendix B.

input : Two strings σ and σ', and
Φ a set of transformations
output: true/false
1 Compute $\sigma_A = atoms_\Phi(\sigma)$ using Algorithm 1;
2 Compute $\sigma_B = atoms_\Phi(\sigma')$ using Algorithm 1;
3 **if** $(\sigma_A = \sigma_B)$ **return** true;
4 **else return** false;

Algorithm 3: Given a uniform and reversible Φ, the algorithm solves the forensics problem.

4.5. Summary of Theoretical Results

We formulated attack transformations and their uniformity and reversibility. We proved that **if** Φ is a uniform and reversible set of transformations, **then** Algorithm 2 is Φ^k-complete and Φ-sound (Theorem 2). We developed an algorithm to compute atoms (Algorithm 1) and showed that when Φ is uniform and reversible $atoms_\Phi(\sigma)$ is a singleton set. We showed that when $\sigma \overset{\Phi}{\Rightarrow} \sigma'$ then $atoms_\Phi(\sigma) = atoms_\Phi(\sigma')$ (Claim 3). This observation immediately leads to an algorithm that, given a uniform and reversible Φ, solves the forensics problem (Algorithm 3). Last, we investigated the union property that ensures that a union of two sets of uniform and reversible transformations is also uniform and reversible.

In the next section, we show that, when carefully defined, common attack transformations are uniform and reversible.

5. Uniform and Reversible TCP Rules

We present a set of common TCP transformations that are reversible and uniform. To prove that our transformations are reversible and uniform, we first formally define the notion of a TCP sequence and the semantics of our transformations (Section 5.1). Next, we define a partial order over TCP streams, called *complexity* (Section 5.2). Last, we prove that our rules are reversible and uniform with respect to *complexity* (Section 5.3).

The reader should be advised that the proof of uniformity and reversibility is based on the semantics of our rules. In other words, we do not claim that all previous attack mutation systems that include these types of transformations define the transformations so they are reversible and uniform. The reader is encouraged to check that our semantics closely represents the nature of these transformations. Therefore, we believe that our rule definitions can be easily adopted by mutation systems with which we are familiar.

5.1. Semantics of TCP Transformations

A TCP sequence represents the communication between an attacker and a victim. A sequence is a list of segments, $\langle s_1, \ldots, s_n \rangle$, where each segment represents a single message that the attacker and victim exchange. Each segment

Name	Description	Example
frag$^+$	Fragments a payload of a TCP segment into two segments. Can create overlapping segments of the payload.	$\langle(0,\texttt{abcd})\rangle \xrightarrow{frag^+} \langle(0,\texttt{abc}),(1,\texttt{bcd})\rangle$
frag$^-$	Defragments two TCP segments into a single one. Removes overlapping segments of the payload.	$\langle(0,\texttt{abc}),(1,\texttt{bcd})\rangle \xrightarrow{frag^-} \langle(0,\texttt{abcd})\rangle$
swap$^+$	Swaps two TCP segments such that they are sent out-of-order.[a]	$\langle(0,\texttt{abc}),(1,\texttt{bcd})\rangle \xrightarrow{swap^+} \langle(1,\texttt{bcd}),(0,\texttt{abc})\rangle$
swap$^-$	Swaps two TCP segments such that they are sent in-order.	$\langle(1,\texttt{bcd}),(0,\texttt{abc})\rangle \xrightarrow{swap^-} \langle(0,\texttt{abc}),(1,\texttt{bcd})\rangle$
ret$^+$	Retransmits a TCP segment. Can retransmit only part of the payload. Note that application of this rule can produce an identical result to *frag*$^+$ application.	$\langle(0,\texttt{abcd})\rangle \xrightarrow{ret^+} \langle(0,\texttt{abcd}),(1,\texttt{bc})\rangle$
ret$^-$	Remove retransmitted segments from a TCP stream. Note that application of this rule may produce an identical result to a *frag*$^-$ application.	$\langle(0,\texttt{abcd}),(1,\texttt{bc})\rangle \xrightarrow{ret^-} \langle(0,\texttt{abcd})\rangle$

Table 1. Φ_{tcp}: a uniform and reversible set of TCP transformations.

[a] In TCP jargon, out-of-order means not in the order of their sequence numbers.

is formulated as a pair, (*seq*, *payload*), where s_i.*seq* represents the sequence-number of s_i and s_i.*payload* represents the message (in bytes) that s_i contains.

The position of a segment in a sequence determines the time this segment is sent by the attacker: s_i is sent only after s_j have been sent for all $j < i$, and before s_k for all $k > i$. For brevity, our TCP sequence definition only includes the segments sent by the attacker.

Table 1 presents Φ_{tcp}, a set of transformations in our TCP mutation system. The superscript $^+$ denotes expanding rules and the superscript $^-$ denotes shrinking rules. Formally, each rule has the form of $\frac{\sigma, pre(\sigma)}{\tau, post(\sigma,\tau)}$ (Section 4.1). Several observations should be noted:

1. Our system includes TCP rules that fragment a TCP stream, deliver segments out-of-order, and add retransmitted segments. Table 1 informally describes each rule and provides an example of its effects. The formal semantics of the rules are presented in Appendix A.
2. Our TCP fragmentation rule can create overlapping TCP segments. This definition is broader than previous ones that define fragmentation as splitting (e.g., [21]). It turns out that splitting alone is not enough for uniformity. The existence of retransmission with merging, or unsplitting, can create overlapping segments and the only way to simplify such segments is by defining de-fragmentation (i.e., *frag*$^-$) with overlapping.
3. There are cases in which fragmentation and retransmission have the same effect. For example, $\langle(0,\texttt{abc})\rangle \xrightarrow{ret^+} \langle(0,\texttt{abc}),(1,\texttt{bc})\rangle$ and $\langle(0,\texttt{abc})\rangle \xrightarrow{frag^+} \langle(0,\texttt{abc}),(1,\texttt{bc})\rangle$. However, each of the rules also has a unique effect. The retransmission rule always retransmits a substring of a segment, while the fragmentation rule can split a segment into two.
4. Our TCP retransmission rule retransmits the same data. This means that if two segments overlap, they transmit the same payload in their overlapping parts. This definition of retransmission facilitates the uniformity proof. Section 8 discusses retransmission of different data.

5.2. A Partial Order for TCP Sequences

To show that Φ_{tcp} (Table 1) is uniform, we must show that the rules in Φ_{tcp} are either shrinking or expanding with respect to a partial order.

We order TCP sequences according to their *complexity*. We say that σ is more complex than τ if it delivers a longer payload, delivers the same payload but uses more segments, or delivers the same payload with the same number of segments but the segments in σ are more disordered (as we define below) than the segments in τ.

Definition 5 (Length of a TCP sequence). *Let* $\sigma = \langle s_1, \ldots s_n\rangle$ *be a TCP sequence. Define* length(σ) *to be the pair* $(n, \sum_{i=1}^{n} \text{size_of}(s_i.\text{payload}))$.

Let length $(\sigma)=(n,k)$ *and* length$(\tau)=(m,j)$ *then:*

1. *We say that* length(σ)=length(τ) *if and only if* $n = m$ *and* $k=j$.
2. *We say that* length(σ)<length(τ) *if and only if* $(n<m)$ *or* $(n=m \wedge k < j)$.

The next component of *complexity* is the *disorder level* of a TCP sequence. The disorder level of σ counts the number of segment pairs that are sent out-of-order. For example, the disorder level of a sequence that sends segments ordered according to their sequence numbers is zero. Similarly, the disorder level of a sequence that sends the segments in their reverse order is $\frac{n(n-1)}{2}$.

Definition 6 (TCP sequence disorder level). *Let* $\sigma = \langle s_1,\ldots,s_i,\ldots,s_j,\ldots,s_n\rangle$ *be a TCP sequence. Define:*

1. $not_in_order(\sigma, s_i, s_j) = 1$ *if and only if* $i < j$ *and* s_i.seq $> s_j$.seq.
2. disorder$(\sigma) \equiv \sum_{1\le k<l\le n} not_in_order(\sigma, s_k, s_l)$.

Definition 7 (Complexity of a TCP sequence)**.** *Let σ be a TCP sequence.*
Define $\text{complexity}(\sigma) \equiv (\text{length}(\sigma), \text{disorder}(\sigma))$.

1. *We say that* $\text{complexity}(\sigma) = \text{complexity}(\tau)$ *if and only if* $\sigma = \tau$.
2. *We say that* $\text{complexity}(\sigma) < \text{complexity}(\tau)$ *if and only if* $(\text{length}(\sigma) < \text{length}(\tau))$ *or* $(\text{length}(\sigma) = \text{length}(\tau) \wedge (\text{disorder}(\sigma) < \text{disorder}(\tau))$.

We say that σ is less complex than τ, denoted $\sigma \prec \tau$, if $\text{complexity}(\sigma) < \text{complexity}(\tau)$.

Note that *complexity* is a partial order; it ranks sequence using *length* as the primary index and *disorder* as a secondary one. As required in Section 4.2.1, any descending chain of complexity is finite: we cannot simplify an attack instance infinitely simply because length is bounded by zero segments. Furthermore, *complexity* is length preserving as required by Theorem 2.

Claim 5. *With respect to* complexity*, each rule in Φ_{tcp} is either shrinking or expanding.*

The proof of Claim 5 is presented in Appendix B.

5.3. Uniformity Proof of Φ_{tcp}

To prove that Φ_{tcp} is uniform we prove that all the conditions of the following claim hold.

Claim 6. *Let Φ be a set of transformations. If (i) Φ is reversible, and (ii) each rule in Φ is either shrinking or expanding with respect to a partial order $\preceq$, and (iii) for all σ and τ such that $\sigma \overset{\Phi}{\Rightarrow} \tau$,* $\text{atoms}_\Phi(\sigma) = \text{atoms}_\Phi(\tau)$*, and (iv) for all σ,* $\text{atoms}_\Phi(\sigma)$ *is a singleton set, and (v) if σ_A is an atom of σ, there exists a derivation from σ to σ_A that only uses shrinking rules, then Φ is uniform with respect to $\preceq$.*

Proof of claim 6: We show that if $\sigma \overset{\Phi}{\Rightarrow} \tau$, then there exists a uniform derivation from σ to τ. According to conditions (iii) and (iv), we know that there exists σ_A such that $\sigma_A = \mathit{atoms}_\Phi(\sigma) = \mathit{atoms}_\Phi(\tau)$. According to condition (v), we know that the following derivations exist: $\sigma \overset{\Phi^-}{\Rightarrow} \sigma_A$ and $\tau \overset{\Phi^-}{\Rightarrow} \sigma_A$. Since Φ is reversible (condition (i)), we get $\sigma_A \overset{\Phi^+}{\Rightarrow} \tau$. So, we have a uniform derivation from σ to τ: $\sigma \overset{\Phi^-}{\Rightarrow} \sigma_A \overset{\Phi^+}{\Rightarrow} \tau$. □

Notice that for Φ_{tcp} condition (i) holds because the rules in Φ_{tcp} are based on string operations that are reversible (e.g., concatenation, permutation). Furthermore, condition (ii) holds because the rules in Φ_{tcp} are either shrinking or expanding with respect to *complexity* (Claim 5).

Let σ be a TCP sequence. The reader is encouraged to check that for any overlapping between σ segments, and for any order of σ segments, if we repeatedly apply shrinking rules, we get a TCP sequence in which (i) all segments are ordered according to their sequence numbers (otherwise we could apply swap^-), (ii) each byte is transmitted exactly once (otherwise we could apply either frag^- or ret^-), and (iii) we use the least number of TCP segments as possible (otherwise we could apply frag^-). Therefore, by repeatedly applying shrinking rules, we get an atom of σ, that is, condition (v) holds.

Consider σ and its σ_A. σ_A is the single atom of σ because our rules do not change the payload of the TCP sequence and according to the TCP specifications [19], there is only a single way in which one can transmit a payload such that each byte is transmitted exactly once, in order, with the least number of segments. So, $\mathit{atoms}_{\Phi_{tcp}}(\sigma)$ is a singleton set and condition (iv) holds.

Last, consider σ and τ such that $\sigma \overset{\Phi_{tcp}}{\Rightarrow} \tau$. Since according to the TCP specifications the rules in Φ_{tcp} do not alter the payload of a TCP sequence, if $\sigma \overset{\Phi_{tcp}}{\Rightarrow} \tau$ it means that both σ and τ transmits the same payload. Since both have a single atom and both atoms transmit the same payload, according to the TCP specification, the two atoms must be identical. So, condition (iii) holds.

We showed that for Φ_{tcp} all the conditions in Claim 6 hold, so Φ_{tcp} is uniform and reversible with respect to *complexity*.

6. Uniform and Reversible HTTP Rules

We illustrate a uniform and reversible set of transformations for the HTTP protocol. We illustrate two representative transformations: *HTTP padding* that pads an HTTP request with spaces (either before or after a URL) and *HTTP encoding* that encodes a URL using hexadecimal values. We chose these transformations because they have been successfully used to evade NIDS [24, 29]. Furthermore, these transformations represent other application-level transformations that modify the attack payload. We further discuss other application-level transformations in Section 8.

We abstract an HTTP attack as a single string, for example, "`GET /cgi-bin/perl.exe HTTP/1.1`". To define an HTTP attack, we use a regular language that conforms to the HTTP specifications [8], denoted L_{http}:

$$L_{http} = m \cdot (SP)^+ \cdot L_{url} \cdot (SP)^+ \cdot (\texttt{HTTP})$$

where:

- L_{url} defines a URL as a string over ASCII characters or their hexadecimal encodings. Formally, $L_{url} \subseteq \{(ASCII \cup h(ASCII))^*\}$ where ASCII is the standard ASCII character set and $h(ASCII)$ is a regular substitution [12] that maps an ASCII character to a string representing the character's hexadecimal encoding, for example $h(\text{`a'}) = \text{"\%61"}$.

Name	Pre Condition	Post Condition
pad_1^+	$(m)(SP)^i(url)(SP)^j(\texttt{HTTP}) \wedge (i \geq 1, j \geq 1)$	$(m)(SP)^{i+1}(url)(SP)^j(\texttt{HTTP})$
pad_1^-	$(m)(SP)^i(url)(SP)^j(\texttt{HTTP}) \wedge (i > 1, j \geq 1)$	$(m)(SP)^{i-1}(url)(SP)^j(\texttt{HTTP})$
pad_2^+	$(m)(SP)^i(url)(SP)^j(\texttt{HTTP}) \wedge (i \geq 1, j \geq 1)$	$(m)(SP)^i(url)(SP)^{j+1}(\texttt{HTTP})$
pad_2^-	$(m)(SP)^i(url)(SP)^j(\texttt{HTTP}) \wedge (i \geq 1, j > 1)$	$(m)(SP)^i(url)(SP)^{j-1}(\texttt{HTTP})$
url^+	$(m)(SP)^i(\alpha x \beta)(SP)^j(\texttt{HTTP}) \wedge x \in ASCII$	$(m)(SP)^i(\alpha \gamma \beta)(SP)^j(\texttt{HTTP}) \wedge (\gamma = h(x))$
url^-	$(m)(SP)^i(\alpha \gamma \beta)(SP)^j(\texttt{HTTP}) \wedge \gamma \in h(ASCII)$	$(m)(SP)^i(\alpha x \beta)(SP)^j(\texttt{HTTP}) \wedge (h^{-1}(\gamma) = x)$

Table 2. Φ_{http}:A uniform and reversible set of transformations for HTTP-based attacks. h is a regular substitution from ASCII characters to their hexadecimal encodings.

- *SP* stands for white-space characters.
- $m \in \{\texttt{GET}, \texttt{POST}\}$, these are the most common HTTP methods used in HTTP attacks.

Table 2 presents our set of HTTP transformations, denoted Φ_{http}. The rules pad_1 and pad_2 change the number of spaces between the attack components. The *url* rule, encodes a single ASCII character in the attack's URL into its hexadecimal encoding.

To proof the uniformity Φ_{http} we show that all the conditions in Claim 6 hold:

1. From Φ_{http} definition (Table 2) it is clear that each rule is reversible.
2. $\{pad_1^-, pad_2^-, url^-\}$ are shrinking rules with respect to *complexity* (Definition 7). Since each of these rules reduces the number of bytes of an instance, it reduces the instance's *length* (Definition 5). Analogously, $\{pad_1^+, pad_2^+, url^+\}$ are expanding rules.
3. The proofs that $atoms_{\Phi_{http}}(\sigma)$ is a singleton set and that $atoms_{\Phi_{http}}(\sigma) = atoms_{\Phi_{http}}(\tau)$ are similar to the proofs presented for Φ_{tcp} (Section 5.3). In this case, however, the proofs are based on the HTTP specification [8] which states that there is only a single most-concise way to deliver an HTTP attack.

7. Combining Φ_{http} with Φ_{tcp}

We show that $\Phi_{http} \cup \Phi_{tcp}$ is uniform and reversible. We can do that by showing that the conditions of Claim 6 hold. However, this becomes more difficult as we add transformations to our system. For example, in the uniformity proof of Φ_{tcp} (Section 5.3) we assumed that the rules do not change the attack payload, an assumption that is no longer true for the case of Φ_{http}.

We illustrate a different method for proving uniformity. We use Theorem 3 and show that $\Phi_{http} \cup \Phi_{tcp}$ is uniform because Φ_{http} and Φ_{tcp} are positively commutative.

Using Theorem 3 is particulary suitable for proving uniformity of sets of transformations in protocols that belong to different levels of the protocol stack. For example, since TCP is a transformation-level protocol while HTTP is an application-level protocol, TCP specification is indifferent to changes in the HTTP payload and HTTP specification is indifferent to changes in the way TCP transfers the payload. This independency is the basis of the commutativity proof.

The rules in Φ_{http} represent an HTTP attack as a single string while the rules in Φ_{tcp} represent an attack as a TCP sequence (Section 5.1). When we unify Φ_{http} and Φ_{tcp} we must use a single representation for attacks. Hence, we should adjust the definitions of the rules in Φ_{http} to work with multiple TCP segments. Due to space constraints, we discuss this adjustment in Appendix C.

Claim 7. $\Phi_{\text{tcp}} \cup \Phi_{\text{http}}$ *is uniform and reversible.*

Proof of Claim 7. Notice that all the rules in $\Phi_{tcp} \cup \Phi_{http}$ are either shrinking or expanding with respect to the partial order *complexity* (Definition 7). To show that $\Phi_{tcp} \cup \Phi_{http}$ is uniform, we need to show that these sets are set-wise positively commutative (Definition 4). Then, the uniformity of $\Phi_{tcp} \cup \Phi_{http}$ follows from Theorem 3.

To show that Φ_{tcp} and Φ_{http} are positively commutative, we show that for every derivation of the form $\sigma \xrightarrow{r \in \Phi_{http}^+} \sigma' \xrightarrow{s \in \Phi_{tcp}^-} \tau$ there exists an equivalent derivation of the form $\sigma \xrightarrow{s \in \Phi_{tcp}^-} \sigma'' \xrightarrow{r \in \Phi_{http}^+} \tau$ and that for every derivation of the form $\sigma \xrightarrow{r \in \Phi_{tcp}^+} \sigma' \xrightarrow{s \in \Phi_{http}^-} \tau$ there exists an equivalent derivation of the form $\sigma \xrightarrow{r \in \Phi_{http}^-} \sigma'' \xrightarrow{s \in \Phi_{tcp}^+} \tau$. Table 3 presents the major cases of all these derivations; other cases are similar and we omit them for brevity.

8. Modeling Other Transformations

We discuss the uniformity and reversibility of transformations that are not part of our Φ_{tcp} and Φ_{http}.

Modeling other TCP transformations. Header change TCP transformations operate on the header of a TCP segment; for example, they modify the TCP flags [8, 21]. While we do not prove it, we believe that these transformations are uniform because they only involve syntactic manipulation at the TCP level. To prove their uniformity one should first extend the representation of a TCP sequence (Section 5.1) to include a representation for a TCP header. Then, one should extend the definition of *complexity*, so it will enforce the notion of expanding and shrinking rules.

		Original	Change to
$r_1 \in \Phi^+_{http}$	$r_2 \in \Phi^-_{tcp}$	$\langle(0,\mathtt{a}),(0,\mathtt{b})\rangle \overset{pad_1^+}{\rightarrow} \langle(0,\mathtt{a_}),(1,\mathtt{b})\rangle \overset{frag^-}{\rightarrow} \langle(0,\mathtt{a_b})\rangle$	$\langle(0,\mathtt{a}),(0,\mathtt{b})\rangle \overset{frag^-}{\rightarrow} \langle(0,\mathtt{ab})\rangle \overset{pad_1^+}{\rightarrow} \langle(0,\mathtt{a_b})\rangle$
		$\langle(0,\mathtt{a}),(1,\mathtt{b})\rangle \overset{url^+}{\rightarrow} \langle(0,\mathtt{\%61}),(1,\mathtt{b})\rangle \overset{frag^-}{\rightarrow} \langle(0,\mathtt{\%61b})\rangle$	$\langle(0,\mathtt{a}),(1,\mathtt{b})\rangle \overset{frag^-}{\rightarrow} \langle(0,\mathtt{ab})\rangle \overset{url^+}{\rightarrow} \langle(0,\mathtt{\%61b})\rangle$
		$\langle(0,\mathtt{ab}),(1,\mathtt{b})\rangle \overset{url^+}{\rightarrow} \langle(0,\mathtt{a\%62}),(1,\mathtt{\%62})\rangle \overset{ret^-}{\rightarrow} \langle(0,\mathtt{a\%62})\rangle$	$\langle(0,\mathtt{ab}),(1,\mathtt{b})\rangle \overset{ret^-}{\rightarrow} \langle(0,\mathtt{ab})\rangle \overset{url^+}{\rightarrow} \langle(0,\mathtt{a\%62})\rangle$
$r_1 \in \Phi^+_{tcp}$	$r_2 \in \Phi^-_{http}$	$\langle(0,\mathtt{a\%62})\rangle \overset{ret^+}{\rightarrow} \langle(0,\mathtt{a\%62}),(2,6)\rangle \overset{url^-}{\rightarrow} \langle(0,\mathtt{ab}),(1,\epsilon)\rangle$	$\langle(0,\mathtt{a\%62})\rangle \overset{url^-}{\rightarrow} \langle(0,\mathtt{ab})\rangle \overset{ret^+}{\rightarrow} \langle(0,\mathtt{ab}),(1,\epsilon)\rangle$
		$\langle(0,\mathtt{a\%62})\rangle \overset{frag^+}{\rightarrow} \langle(0,\mathtt{a}),(1,\mathtt{\%62})\rangle \overset{url^-}{\rightarrow} \langle(0,\mathtt{a}),(1,b)\rangle$	$\langle(0,\mathtt{a\%62})\rangle \overset{url^-}{\rightarrow} \langle(0,\mathtt{ab})\rangle \overset{frag^+}{\rightarrow} \langle(0,\mathtt{a}),(1,b)\rangle$
		$\langle(0,\mathtt{a_b})\rangle \overset{ret^+}{\rightarrow} \langle(0,\mathtt{a_b}),(0,\mathtt{a_b})\rangle \overset{pad_1^-}{\rightarrow} \langle(0,\mathtt{ab}),(0,\mathtt{ab})\rangle$	$\langle(0,\mathtt{a_b})\rangle \overset{pad_1^-}{\rightarrow} \langle(0,\mathtt{ab})\rangle \overset{ret^+}{\rightarrow} \langle(0,\mathtt{ab}),(0,\mathtt{ab})\rangle$

Table 3. Positive commutativity of Φ_{http} and Φ_{tcp}.

The biggest challenge is to prove the uniformity and reversibility of TCP transformations that contain TCP retransmission of a different payload. The problem is that the content of the bytes is different across different TCP segments. This ambiguity creates a difficulty to define ret^- because we must choose only one of the values.

Notice that there is no ambiguity in practice because the end host resolves the ambiguity. For example, most Linux kernels prefer the first byte they receive. In comparison, other operating systems (e.g., openBSD) prefer the last byte they got. This suggests that it is possible the define TCP retransmission in a way that preserves uniformity, according to the policy defined by a particular operating system. We leave this investigation for future work.

Modeling application-level transformations. Application-level transformations operate on the attack payload. For example, *FTP padding* [13, 24] adds benign commands before the malicious commands of an FTP attack. Since such transformations are similar to the transformations in Φ_{http}, we believe that their uniformity can be proved in a similar way.

Modeling network-level transformation. Network-level transformations (e.g., IP, UDP) change the way the attack is delivered; for example, IP transformations [21] might split IP packets. Such transformations are similar in nature to our TCP transformations and their uniformity proofs should be similar to the proofs for Φ_{tcp}.

9 Conclusion

NIDS testing is a challenging problem in intrusion detection. Experience has shown that many NIDS are evaded easily and frequently. We believe that a Φ-complete mutation algorithm can serve as the basis for a rigorous testing process, even when it is infeasible to test all possible mutations. To the best of our knowledge, we are the first to present such an algorithm.

Acknowledgments. We deeply thank Vinod Ganapathy and the anonymous referees for their useful comments that have helped us refine the concepts presented in this paper.

References

[1] D. Alessandri, editor. *Towards a Taxonomy of Intrusion Detection Systems and Attacks*. IBM Zurich Research Laboratory, Sep. 2001. Deliverable D3, Project MAFTIA IST-1999-11583, Available at `www.maftia.org`.

[2] F. Baader and T. Nipkow. ***Term Rewriting and All That***. Cambridge University Press, 1999.

[3] H. Barendregt. **The Lambda Calculus (Studies in Logic and the Foundations of Mathematics)**. North Holland, 1984.

[4] Y. Chevalier, R. Ksters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *IEEE Symp. on Logic in Computer Science*, Ottawa, Canada, June 2003.

[5] E. M. Clarke, S. Jha, and W. R. Marrero. Verifying security protocols with Brutus. *ACM TOPLAS*, **9** (4), Oct. 2000.

[6] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *IEEE Symp. on Logic in Computer Science*, Ottawa, Canada, June 2003.

[7] M. Dacier, editor. *Design of an Intrusion-Tolerant Intrusion Detection System*. IBM Zurich Research Laboratory, Aug. 2002. Deliverable D10, Project MAFTIA IST-1999-11583, Available at `www.maftia.org`.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *RFC 2616 - Hypertext Transfer Protocol*. The Internet Engineering Task Force, June 1999.

[9] C. Giovanni. Fun with packets: Designing a stick, Mar. 2001. Endeavor Systems.

[10] M. Handley and V. Paxson. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *USENIX Security Symposium*, Washington, DC, Aug. 2001.

[11] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, **110**, 1994.

[12] J. Hopcroft, R. Motwani, and J. Ullman. ***Introduction to Automata Theory, Languages, and Computation***. Addison-Wesley, 2 edition, 2001.

[13] R. Marti. THOR: A tool to test intrusion detection systems by variations of attacks. Master's thesis, Swiss Federal Institute of Technology, Mar. 2002.

[14] C. Meadows. The NRL protocol analysis tool: A position paper. In *IEEE Computer Security Foundations Workshop*, Franconia, NH, June 1991.

[15] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, **51**, 1991.

[16] MITRE Corporation. CVE: Common Vulnerabilities and Exposures. Available at `www.cve.mitre.org`.

[17] D. Mutz, G. Vigna, and R. A. Kemmerer. An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems. In *Annual Computer Security Applications Conference*, Las Vegas, NV, Dec. 2003.

[18] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, **31**(23/24), Dec. 1999.

[19] J. Postel. *RFC 793 - Transmission Control Protocol*. The Internet Engineering Task Force, Sept. 1981.

[20] T. H. Ptacek and T. N. Newsham. Custom attack simulation language (CASL). Available at `www.sockpuppet.org/tqbf/casl.html`.

[21] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical Report T2R-0Y6, Secure Networks, Inc., Calgary, AB, Canada, 1998.

[22] Rain Forest Puppy. A look at whisker's anti-IDS tactics – just how bad can we ruin a good thing?, Dec. 1999. Available at `www.wiretrip.net/rfp/txt/whiskerids.html`.

[23] M. Roesch. Snort: the Open Source Network Intrusion Detection System. Available at `www.snort.org`.

[24] S. Rubin, S. Jha, and B. P. Miller. Automatic generation and analysis of NIDS attacks. In *Annual Computer Security Applications Conference*, Tucson, AZ, Dec. 2004.

[25] S. Rubin, S. Jha, and B. P. Miller. Using attack mutation to test a high-end NIDS. *Information Security Bulletin*, Apr. 2005.

[26] N. Shankar. Proof search in the intuitionistic sequent calculus. In *Proceedings of 11th International Conference on Automated Deduction (CADE-11)*, Saratoga Springs, NY, June 1992.

[27] Sniphs. Snot, Jan. 2003. Available at `www.stolenshoes.net/sniph/index.html`.

[28] D. Song. Fragroute: a TCP/IP fragmenter, Apr. 2002. Available at `www.monkey.org/~dugsong/fragroute`.

[29] G. Vigna, W. Robertson, and D. Balzarotti. Testing network-based intrusion detection signatures using mutant exploits. In *ACM Conference on Computer and Communications Security*, Washington, DC, Oct. 2004.

A Formal Definition of Φ_{tcp}

The formal definition of Φ_{tcp} is given in Table 4. The predicates used in this table are given below.

Definition 8 (Fragmentation and defragmentation of a TCP segment)**.** *Let* $s = \{seq, payload\}$ *be a TCP segment,* pf *be a (possibly empty) prefix of* s.payload*, and* sf *be a (possibly empty) suffix of* s.payload *such that* size_of(pf)+size_of(sf) $\geq$ size_of(s.payload)*. Then, define* seg_frag$(s) = (s_1, s_2)$ *such that:*

1. s_1.payload = pf *and* s_2.payload = sf.
2. s_1.seq = s.seq *and* s_2.seq = s.seq+size_of(s.payload)-size_of(sf)*.*

TCP defragmentation is defined as the inverse operation of TCP fragmentation. That is, we say that a segment r is a defragmentation of s_1 and s_2 if *seg_frag*$(r) = (s_1, s_2)$.

Definition 9 (Retransmission of a TCP segment)**.** *Let* $s = \{seq, payload\}$ *be a TCP segment and* substr *be a (possibly empty) substring of* s.payload*.* retrans$(s) = r$ *such that* r.payload = substr *and* r.seq = s.seq + *(the index of the first character of* substr *in* s.payload*).*

B Proofs Mentioned in the Paper

Induction proof for Claim 3: A derivation with a single uniformity violation is of the form $\sigma \overset{R_1^-}{\Rightarrow} \cdot \overset{R_1^+}{\Rightarrow} \cdot \overset{S_1^-}{\Rightarrow} \cdot \overset{S_1^+}{\Rightarrow} \tau$. We use Claim 4 and convert this derivation into $\sigma \overset{R_1^-}{\Rightarrow} \cdot \overset{\hat{S}_1^-}{\Rightarrow} \cdot \overset{\hat{R}_1^+}{\Rightarrow} \cdot \overset{S_1^+}{\Rightarrow} \tau$.

Induction step: Consider Derivation 2 with n uniformity violations. First, we use the induction base and convert this derivation into:

$$\sigma \overset{R_1^-}{\Rightarrow} \cdot \overset{\hat{S}_1^-}{\Rightarrow} \cdot \overset{\hat{R}_1^+}{\Rightarrow} \cdot \overset{S_1^+}{\Rightarrow} \sigma_2 \overset{R_2^-}{\Rightarrow} \sigma_3 \overset{R_2^+}{\Rightarrow} \cdots \overset{R_n^-}{\Rightarrow} \cdot \overset{R_n^+}{\Rightarrow} \cdot \overset{S_n^-}{\Rightarrow} \cdot \overset{S_n^+}{\Rightarrow} \tau$$

The derivation above from σ_2 to τ has only $n-1$ uniformity violations. We use the induction hypothesis to get:

$$\sigma \overset{R_1^-}{\Rightarrow} \cdot \overset{\hat{S}_1^-}{\Rightarrow} \cdot \overset{\hat{R}_1^+}{\Rightarrow} \cdot \overset{S_1^+}{\Rightarrow} \sigma_2 \overset{R_2^-}{\Rightarrow} \sigma_3 \overset{\hat{S}_2^-}{\Rightarrow} \sigma_4 \cdots \overset{\hat{R}_n^-}{\Rightarrow} \cdot \overset{\hat{S}_n^-}{\Rightarrow} \cdot \overset{\hat{R}_2^+}{\Rightarrow} \cdot \overset{\hat{S}_2^+}{\Rightarrow} \cdots \overset{\hat{R}_n^+}{\Rightarrow} \cdot \overset{S_n^+}{\Rightarrow} \tau \quad (4)$$

We use a series of changes to "shift" $\overset{S_1^+}{\Rightarrow}$ into the right side of Derivation 4. Due to positive commutativity we change $\cdot \overset{S_1^+}{\Rightarrow} \sigma_2 \overset{R_2^-}{\Rightarrow} \sigma_3$ into $\cdot \overset{\hat{R}_2^-}{\Rightarrow} \sigma_2' \overset{\hat{S}_1^+}{\Rightarrow} \sigma_3$. We use uniformity of S and change $\sigma_2' \overset{S_1^+}{\Rightarrow} \sigma_3 \overset{\hat{S}_2^-}{\Rightarrow} \sigma_4$ into $\sigma_2' \overset{\hat{S}_2^-}{\Rightarrow} \sigma_3' \overset{\hat{S}_1^+}{\Rightarrow} \sigma_4$. We continue this process until we get the derivation:

$$\sigma \overset{R_1^-}{\Rightarrow} \cdot \overset{S_1^-}{\Rightarrow} \cdot \overset{R_1^+}{\Rightarrow} \cdot \overset{\hat{R}_2^-}{\Rightarrow} \cdot \overset{\hat{S}_2^-}{\Rightarrow} \cdot \ldots \overset{\hat{R}_n^-}{\Rightarrow} \cdot \overset{\hat{S}_n^-}{\Rightarrow} \cdot \overset{\hat{S}_1^+}{\Rightarrow} \cdot \overset{\hat{R}_2^+}{\Rightarrow} \cdot \overset{\hat{S}_2^+}{\Rightarrow} \cdots \overset{\hat{R}_n^+}{\Rightarrow} \cdot \overset{\hat{S}_n^+}{\Rightarrow} \tau \quad (5)$$

Finally, we use the same technique to shift $\overset{R_1^+}{\Rightarrow}$ into the right side of Derivation 5. □

Proof of Claim 5:

***frag*$^+$, *ret*$^+$.** Based on our *length* definition (Definition 5), fragmenting (or retransmitting) a TCP segment increases the sequence *length*, therefore increases the sequence complexity.

***swap*$^+$.** This rule swaps two segments such that they are delivered out of order. This operation increases the *disorder* of the sequence and therefore increases the sequence *complexity*. Formally, Let σ=$\langle s_1 \ldots p \ldots q \ldots s_n \rangle$.

Name	*pre-conditions*	*post-conditions*
frag$^+$	$\sigma = \langle s_1 \dots s_i \dots s_n \rangle$	$\tau = \langle s_1 \dots s_{i-1}, r_1, r_2, s_{i+1} \dots s_n \rangle$ *frag_seg*$(s_i) = (r_1, r_2)$
frag$^-$	$\sigma = \langle s_1 \dots s_i, s_{i+1} \dots s_n \rangle$	$\tau = \langle s_1 \dots s_{i-1}, r_1, s_{i+2} \dots s_n \rangle$ *frag_seg*$(r_1) = (s_i, s_{i+1})$
swap$^+$	$\sigma = \langle s_1 \dots s_i \dots s_j \dots s_n \rangle$ s_i.*seq* $< s_j$.*seq*	$\tau = \langle s_1, \dots s_j \dots s_i \dots s_n \rangle$
swap$^-$	$\sigma = \langle s_1, \dots s_i \dots s_j \dots s_n \rangle$ s_i.*seq* $> s_j$.*seq*	$\tau = \langle s_1 \dots s_j \dots s_i \dots s_n \rangle$
ret$^+$	$\sigma = \langle s_1, \dots, s_i, \dots s_n \rangle$	$\tau = \langle s_1, \dots, s_i, r, \dots, s_n \rangle$ $r =$ *retrans*(s_i)
ret$^-$	$\sigma = \langle s_1, \dots, s_i, r, \dots, s_n \rangle$ $r =$ *retrans*(s_i)	$\tau = \langle s_1, \dots, s_i, \dots s_n \rangle$

Table 4. Formal definitions of transformation rules. Each rule has the form $\frac{\sigma, pre(\sigma)}{\tau, post(\sigma,\tau)}$.

Assume $\sigma \xrightarrow{swap^+} \tau$ and $\tau = \langle s_1 \dots q \dots p \dots s_n \rangle$ such that $\sigma[j]{=}\tau[k]{=}p$ and $\sigma[k]{=}\tau[j]{=}q$. From the definition of *swap*$^+$ we know that p.*seq* $< q$.*seq* (Table 1). Note the following:

1. For all i such that $i{>}k$, *not_in_order*(σ, q, s_i) = 1 if and only if *not_in_order*$(\tau, q, s_i){=}1$.
2. For all i such that $i{>}k$, *not_in_order*(σ, p, s_i) = 1 if and only if *not_in_order*$(\tau, p, s_i){=}1$.
3. For all i such that $i{<}j$, *not_in_order*(σ, s_i, p) = 1 if and only if *not_in_order*$(\tau, s_i, p){=}1$.
4. For all i such that $i{<}j$, *not_in_order*(σ, s_i, q) = 1 if and only if *not_in_order*(τ, s_i, q) = 1.
5. For all i such that $j < i < k$:
   (a) Assume p.*seq* $<$ s_i.*seq* $<$ q.*seq*. Then *not_in_order*(σ, p, s_i) = 0 and *not_in_order*(σ, s_i, q) = 0. However, *not_in_order*(τ, s_i, p) = 1 and *not_in_order*(τ, q, s_i) = 1. This means that the swap operation contributes to the value of *disorder*(τ).
   (b) Assume p.*seq* $<$ q.*seq* $<$ s_i.*seq*. Then *not_in_order*(σ, p, s_i) = 0 and *not_in_order*(σ, s_i, q) = 1, but *not_in_order*(τ, s_i, p) = 1 and *not_in_order*(τ, q, s_i) = 0. This means that *disorder*(τ) is at least as *disorder*(σ).
   (c) Assume s_i.*seq* $<$ p.*seq* $<$ q.*seq*. Then *not_in_order*(σ, p, s_i) = 1 and *not_in_order*(σ, s_i, q) = 0, but *not_in_order*(τ, s_i, p) = 0 and *not_in_order*(τ, q, s_i) = 1. This means that *disorder*(τ) is at least as *disorder*(σ).
   (d) Note that other orderings of p, q and s_i are impossible because p.*seq* $< q$.*seq*.
6. Since we used *swap*$^+$, *not_in_order*$(\sigma, p, q) = 0$ but *not_in_order*$(\tau, q, p) = 1$.

Proofs that shrinking transformations reduce *complexity* are analogous to the proofs above. Hence, our *complexity* order is suitable for a uniform and reversible attack mutation system. □

From properties (a) to (f), we conclude that *disorder*$(\tau) \geq$ *disorder*$(\sigma) + 1$. Since the *swap*$^+$ transformation does not change the length of a stream, *complexity*$(\sigma) <$ *complexity*(τ) and $\sigma \prec \tau$.

C Adjusting Φ_{http}

The rules in Φ_{http} represent an HTTP attack as a single string while the rules in Φ_{tcp} represent an attack as a TCP sequence (Section 5.1). When we unify Φ_{http} and Φ_{tcp} we must use a single representation. Since our TCP representation contains the payload, it is natural to express the HTTP transformations in terms of TCP sequences. However, we need to preserve the properties of our HTTP and TCP rules:

1. We need to express how HTTP rules change the sequence numbers of a TCP sequence. Since our HTTP rules insert (or remove) bytes, they need to "shift" the bytes that follow the inserted bytes. For example, *url*$^+$ should update sequence numbers of TCP segments that are different than the segment in which *url*$^+$ encoded the byte:

$$\langle (0, \texttt{ab}), (2, \texttt{cd}), (4, \texttt{ef}) \rangle \xrightarrow{url^+} \langle (0, \texttt{a\%62}), (4, \texttt{cd}), (6, \texttt{ef}) \rangle$$

2. Recall that Φ_{tcp} does not support retransmission of different payload (Section 5.1). This means that when an HTTP rule insert, remove, or change a byte in the TCP stream it must do so for every copy of the byte in the stream. For example:

$$\langle (0, \texttt{abc}), (1, \texttt{bcd}), (4, \texttt{ef}) \rangle \xrightarrow{url^+} \langle (0, \texttt{a\%62c}), (1, \texttt{\%62cd}), (6, \texttt{ef}) \rangle$$

These two changes of our HTTP rules can be formally expressed as a simple procedure that traverses the segments of a TCP sequence and modify them as necessary.