A Taxonomy and Adversarial Model for Attacks against Network Log Anonymization

Justin King*
Systems & Tech. Group
IBM Rochester
jkking@us.ibm.com

Kiran Lakkaraju
Dept. of Computer Science
UIUC
klakkara@illinois.edu

Adam Slagell
NCSA
UIUC
slagell@illinois.edu

ABSTRACT

In recent years, it has become important for researchers, security incident responders and educators to share network logs, and many log anonymization tools and techniques have been put forth to sanitize this sensitive data source in order to enable more collaboration. Unfortunately, many new attacks have been created, in parallel, that try to exploit weaknesses in the anonymization process. In this paper, we present a taxonomy that relates similar kinds of attacks in a meaningful way. We also present a new adversarial model which we can map into the taxonomy by the types of attacks that can be perpetrated by a particular adversary. This has helped us to negotiate the trade-offs between data utility and trust, by giving us a way to specify the strength of an anonymization scheme as a measure of the types of adversaries it protects against.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: [Network Monitoring]; K.4.1 [**Public Policy Issues**]: [Privacy]; K.4.3 [**Organizational Impacts**]: [Computer-supported collaborative work]; K.6.m [**Miscellaneous**]: [Security]

General Terms

Security, Verification

Keywords

Adversarial Model, Anonymization, Network Logs, Taxonomy

1. INTRODUCTION

As many have argued, sharing of network traces, flow data and other logs is vitally important to research, industry and pedagogy [22, 16, 14]. The network and security research communities rely on large, diverse and non-synthetic [12] data sets for empirical studies. Incident responders need to share real logs to collaborate on investigations of attacks that cross organizational boundaries. Educators and those creating educational materials need example data for student projects and exercises.

*This work was performed when the author was at UIUC.

However, it has frequently been pointed out that these sorts of data are often very sensitive [22, 18, 23, 17]. There are security concerns about what information may be revealed regarding the data collector's network and systems, and there are legal questions about betraying the trust of the users whose private actions are being recorded [19].

Anonymization techniques have been developed to alleviate this conflict created by the need for security and trust on the one hand and high utility data sets on the other [14, 23, 25, 26, 13]. In fact, several tools have been created so that data owners can sanitize network data [20, 15, 9, 21]. While very useful, these tools alone do not solve the entire problem. One must know how to create anonymization policies that provide the necessary level of assurance—especially in lieu of many recently developed attacks against anonymization [8, 4, 5, 18]. These tools alone do nothing if one does not know how to use them intelligently.

As a corollary to this trade-off between information loss and trust, there are always at least two actors with opposing interests in the creation of an anonymization policy. The data owner and the people whose behavior the data describes want policies to protect their privacy and security interests. Even if the data does not directly reveal sensitive information about the data owner's network or services, Internet service providers and such do not want to violate the consumer's trust or their own privacy policies. On the other hand, the person(s) doing analysis of the data—whether it is a researcher or someone investigating a specific intrusion—needs data as close to the original as possible. Alterations can affect analysis, and they want these minimized. Therefore one party wants more anonymization of the data and the other wants less.

Balancing these two different needs requires the ability to specify constraints on which fields must and must not be anonymized, and with what algorithms such fields may be anonymized. In other work [6], we have developed a first order predicate logic in which these constraints may be specified to generate anonymization policies or test them for compliance. However, one must still come up with mechanisms to create intelligent constraints. In [10], we investigated the needs of the party performing the analysis so that they could specify minimal constraints on what can *not* be anonymized—trying to measure the utility of anonymized data. This work, instead, focuses on how one can specify minimal constraints on what must be anonymized to meet the minimum assurance level of the data owner.

A natural way to express one's security or trust requirements is in terms of the types of attacks or the type of adversary they must withstand. With this in mind, we have created a taxonomy of the attacks against network log anonymization based upon common attack preconditions. We have created a more thorough and mutually exclusive taxonomy than previous attempts which not only reflects all currently known attacks, but we believe it will be able to incorporate future attacks. Furthermore, since it is based upon attack preconditions, namely what information is necessary for the adversary to perform the attack, it readily translates into constraints for our predicate logic. Therefore, constraints can be easily derived to prevent a single attack, a class of attacks, or arbitrary collections of attacks through simple logical conjunctions of these statements.

The other natural way to express a level of assurance is to specify the type of adversary that must be protected against. Therefore, our second contribution in this paper is the development of an adversarial model based upon adversarial means and capabilities. Lastly, we tie together the adversarial model and attack taxonomy so one can translate between constraints based upon either adversaries or the classes of attacks which they can perpetrate.

This paper is organized as follows: Section 2 presents our taxonomy of attacks. Section 3 introduces the basics of our adversarial model, including the notation we have developed to express the model. Section 4 establishes our mapping between the taxonomy and adversarial model. Section 5 discusses related work, and section 6 presents future work and our conclusions.

2. ATTACK TAXONOMY

Classifying attacks against log anonymization is an early step towards a comprehensive study of the security of anonymization policies. If network owners can select classes of attacks that they wish to prevent, they can then ensure that their anonymization policies meet their security constraints, while allowing as much non-private information as possible to be revealed—thus increasing a data set's utility.

2.1 Motivations and Requirements

As described previously, we wish to provide network owners with a taxonomy of attacks, the classes of which they can select to prevent, rather than having to focus on individual attacks. We also wish to formally express relationships between attacks, allowing for expression of attack groupings in a logic about anonymization. This taxonomy must be complete (every known attack can be placed in at least one class) and mutually exclusive (no attack can be a member of more than one class). The classes must be fine-grained enough for network owners to select specific classes without seriously impacting the utility of a log. Finally, the classes must be tied together in a more concrete way than a description in natural language.

2.2 Methodology

Nineteen attacks from multiple sources [23, 15, 11, 2, 14, 4, 22, 7, 5, 24] were collected as representative of current attacks against log anonymization. These attacks varied significantly in both sophistication and mechanisms used. The attacks included all the attacks specifically mentioned both by Pang and Paxson [15] and by Slagell *et al.* [22], and so the attack sample is a strict superset of those considered for those previous taxonomies discussed in Section 5.

Each attack was analyzed for its pre-conditions. We chose to organize around pre-conditions because they capture the initial knowledge of the adversary and properties of the anonymized log. Every anonymization attack requires some initial knowledge and some specific log properties. By organizing around pre-conditions, we can identify the knowledge and log properties that are crucial to the execution of an anti-anonymization attack. Section 3 will demonstrate how we can take this initial knowledge and log properties into account when modeling an adversary and when determining whether a given adversary can carry out a particular attack.

A graph was constructed, with nodes for each attack under consideration, as well as for pre-conditions for each attack. In cases where two or more nodes shared a common pre-condition, that condition was represented with a single node. Edges were added to the graph connecting pre-conditions to the attacks they enabled. Attempts were then made to satisfactorily group attacks with common pre-conditions.

2.3 Results

The full graph using pre-conditions is illustrated in Figure 1. We note that a large number of the attacks (9 of 19) require some form of consistent IP pseudonyms in order to identify individual machines in the log, and an additional attack specifically requires prefix-preserving pseudonyms [23]. Because this pre-condition does not help us distinguish many attacks, we ignore it when generating our taxonomy. Given this, we now have six unconnected subgraphs that can be treated as classes in our taxonomy. Two of these subgraphs each involve a single attack/pre-condition pair, and as both of these attacks essentially involve potential cryptographic weaknesses in the anonymization functions themselves, we group them together into a single class.

Remarkably, this analysis reveals a *high-level* taxonomy virtually identical to those given by Slagell *et al* [22]. Several of these classes share no common pre-conditions, and the only pre-condition shared between classes is a consistent IP mapping (such as a prefix-preserving mapping [23]) that enables identification of individual machines in the log. The major classes in our taxonomy, separated in Figure 1 with black, dashed lines, are as follows:

- **Fingerprinting**: The process of matching attributes of an anonymized object against attributes of a known object to discover a mapping between anonymized and unanonymized objects.
- **Structure Recognition**: The act of recognizing structure between objects to use one mapping to discover multiple mappings between anonymized and unanonymized objects.
- **Known Mapping**: Exploiting the discovery of a mapping between unanonymized and anonymized data in one reference, to undo anonymization of that same data in multiple places.
- **Data Injection**: Injecting information to be logged with the purpose of later recognizing that data in an anonymized form.
- **Cryptographic**: Attacking the cryptographic primitives that form the basis for anonymization algorithms.

Ten of the nineteen attacks are classified as Fingerprint-

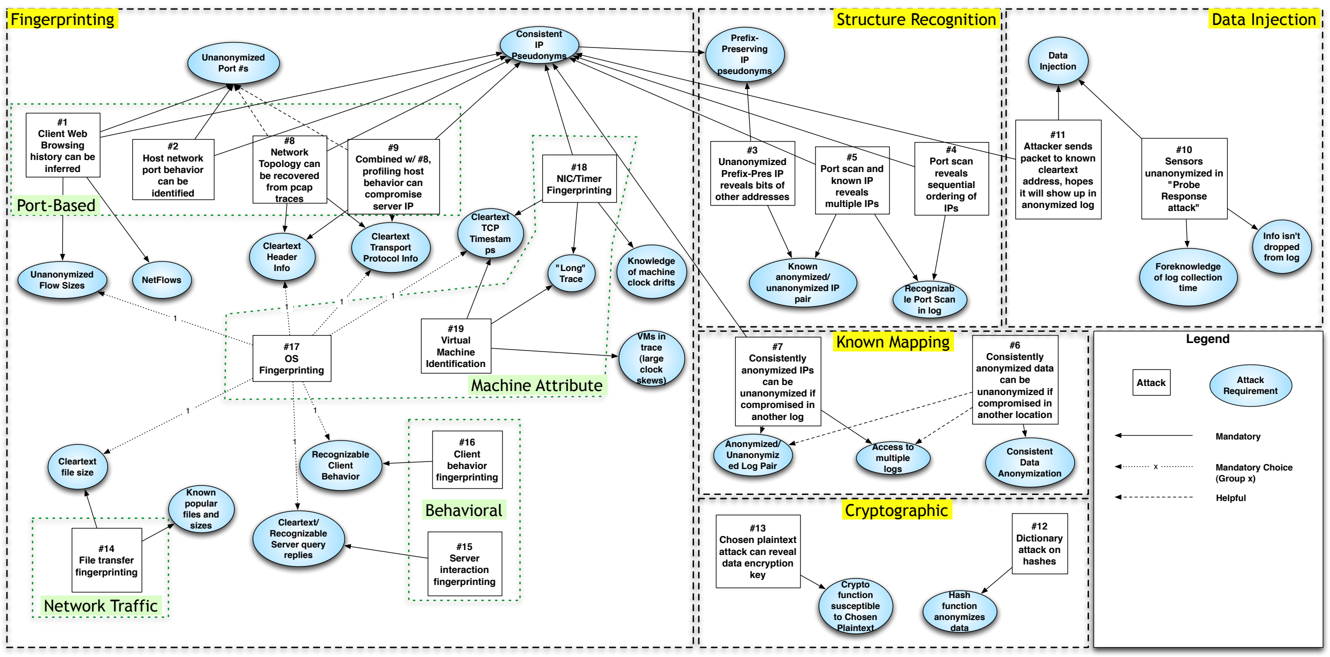
Fingerprinting
Unanonymized Port #s
Consistent IP Pseudonyms
#1 Client Web Browsing history can be inferred
#2 Host network port behavior can be identified
#8 Network Topology can be recovered from pcap traces
#9 Combined w/ #8, profiling host behavior can compromise server IP
Port-Based
#18 NIC/Timer Fingerprinting
Cleartext TCP Timestamps
Cleartext Header Info
Cleartext Transport Protocol Info
"Long" Trace
Knowledge of machine clock drifts
Unanonymized Flow Sizes
NetFlows
#17 OS Fingerprinting
#19 Virtual Machine Identification
VMs in trace (large clock skews)
Machine Attribute
Cleartext file size
Recognizable Client Behavior
#16 Client behavior fingerprinting
Known popular files and sizes
Cleartext/ Recognizable Server query replies
Behavioral
#15 Server interaction fingerprinting
#14 File transfer fingerprinting
Network Traffic
Structure Recognition
Prefix-Preserving IP pseudonyms
#3 Unanonymized Prefix-Pres IP reveals bits of other addresses
#5 Port scan and known IP reveals multiple IPs
#4 Port scan reveals sequential ordering of IPs
Known anonymized/ unanonymized IP pair
Recognizable Port Scan in log
Data Injection
Data Injection
#11 Attacker sends packet to known cleartext address, hopes it will show up in anonymized log
#10 Sensors unanonymized in "Probe Response attack"
Foreknowledge of log collection time
Info isn't dropped from log
Known Mapping
#7 Consistently anonymized IPs can be unanonymized if compromised in another log
#6 Consistently anonymized data can be unanonymized if compromised in another location
Anonymized/ Unanonymized Log Pair
Access to multiple logs
Consistent Data Anonymization
Cryptographic
#13 Chosen plaintext attack can reveal data encryption key
Crypto function susceptible to Chosen Plaintext
#12 Dictionary attack on hashes
Hash function anonymizes data
Legend
Attack
Attack Requirement
Mandatory
x
Mandatory Choice (Group x)
Helpful
1

Figure 1: Grouping of Anonymization Attacks by Pre-conditions

ing attacks. Our original criticism of the Slagell *et al* classification was that it was not fine-grained enough to allow for policy creation that provided a high level of both security and utility. Thus, we have attempted to create subclasses of the Fingerprinting attack class. We first analyzed common prerequisites between attacks. However, no simple approach—as taken to create the major classes—could be found. Instead, we decided to create subclasses using a case-by-case approach:

- Attacks #1, #2, #8, and #9 all relied on unanonymized port numbers (or on being able to extract the port numbers from statistical analysis [4]). We chose to call these attacks **Port-based Fingerprinting**: Fingerprinting by network ports or well-known services.
- Attacks #15 and #16, while not sharing any common pre-conditions, were based on a known machine/user behavior, and so we title these **Behavioral Fingerprinting**: Fingerprinting based on observing an entity's known behavior.
- Attacks #18 and #19 were simple to group together, as both rely on cleartext timestamps. We also decided to group OS Fingerprinting (#17) in this subclass, though we will discuss it further below. We call this **Machine Attribute Fingerprinting**: Fingerprinting based on (relatively) immutable attributes of a machine, such as hardware timer drift or operating system characteristics.
- Alone in this subclass is attack #14, File Transfer Fingerprinting, which is the only fingerprinting attack in our sample which attempts to fingerprint something *other* than a machine. For this reason, we created a separate subgroup for it, which we believe can be expanded with other examples. We titled this class **Network Traffic Fingerprinting**: Fingerprinting based on recognizing patterns inherent to network traffic such as known file transfer sizes.

Figure 2 illustrates the complete taxonomy.

OS Fingerprinting presents a particular problem for our classification. We feel that OS Fingerprinting is viewed best not as a single attack, but a group of attacks based on a post-condition. If we are structuring our taxonomy upon pre-conditions, though, it is more sensible to split OS Fingerprinting attacks among the other four subclasses—that is to say, there may be Port-based, Behavioral, Machine Attribute-based, and Network Traffic-based ways to perform OS fingerprinting. Thus, to stop *all* forms of OS fingerprinting, we need to consider each subclass in turn. We leave these considerations as future work.

2.4 Analysis of Post-conditions

We attempted to analyze attack post-conditions as well. However, a majority of the attacks we examined (10 of 19) were specifically aimed at identifying a machine or deanonymizing an IP address. The remaining nine attacks each have a unique attribute that they deanonymize. As most of these single attack/post-condition pairs seem to be unrelated, and the remaining attacks all focused on the goal of machine identification, we determined that post-condition analysis was not useful for our purposes. A more thorough discussion of our post-condition analysis is available in [6].

3. ADVERSARIAL MODEL

Our model consists of a number of elements, representing logs, log entities, log entity' properties and data, the adversary's knowledge set, patterns, and a collection of four adversarial means. We will explain each of these elements, and Section 3.7 will give a thorough example of how these elements can be combined to describe an adversary's capabilities.

3.1 Entities

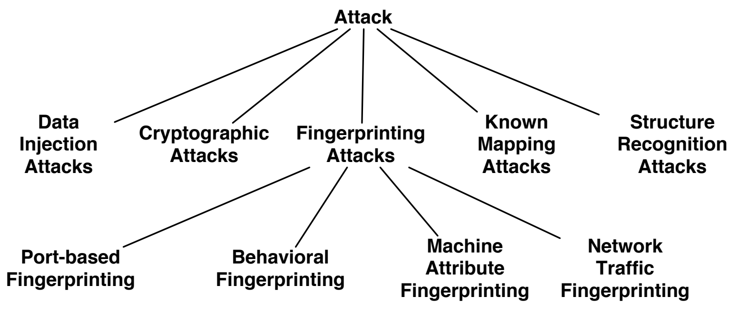
Attack
Data Injection Attacks
Cryptographic Attacks
Fingerprinting Attacks
Known Mapping Attacks
Structure Recognition Attacks
Port-based Fingerprinting
Behavioral Fingerprinting
Machine Attribute Fingerprinting
Network Traffic Fingerprinting

Figure 2: Taxonomy of Attacks Against Anonymization

An *entity* in our model is any item that an adversary may wish to discover information about. Example entities could be machines, users, or files sent over a network. An entity is essentially synonymous with an *object* in [3], though we may wish to discover information about entities that is not in a log, even when unanonymized. For example, we may wish to discover a machine's operating system, even though there is no such field in most log formats.

Entities have attributes, some of which may be known to an adversary and some of which may not be known. Entity attributes may be anonymized or unanonymized, and the adversary may wish to discover either form. We will develop notation for anonymized and unanonymized attributes in section 3.4.

An entity in our model will be denoted as $\mathcal{e}$, with an optional subscript that serves as a way to differentiate entities. For example, a "target" entity may be denoted as $\mathcal{e}_t$. Attributes will be denoted as the entity's name, followed by a period and the name of the attribute. For example, the target entity's unanonymized IP address may be denoted as $\mathcal{e}_t.\underline{ip\_addr}$.

3.2 Adversary Knowledge Set

The adversary's knowledge set is a collection of entities, and their known attributes. *The goal of the adversary is to uncover an entity's attributes, and to add them to its knowledge set.* Data must be in the adversary's knowledge set before it can be used to uncover more information. We consider adding information to the adversary's knowledge set, but we do not believe there is any reason to consider information loss from this set. We denote the adversary's knowledge set using the symbol $\mathcal{K}$, and we denote addition of information to the adversary's knowledge set using the '$\Rightarrow$' symbol. For example, $\mathcal{e}.\underline{ip\_addr} \Rightarrow \mathcal{K}$ means that the entity's unanonymized IP address has been added to the adversary's knowledge set.

The $Knows()$ Assertion
The $Knows()$ assertion lets us state that an adversary knows a specific piece of information. When used to construct attack specifications, the $Knows()$ will cause an attack to fail if it is not true, and thus, it is used to specify an attack's prerequisite knowledge. As an example, the statement $Knows(\mathcal{e}_t.\underline{ip\_addr})$ asserts that $\mathcal{e}_t.\underline{ip\_addr} \in \mathcal{K}$, that is, that the adversary knows the target entity's unanonymized IP address.

3.3 Logs

Abbreviation	Meaning
CP	Consistent Pseudonym
PP	Prefix Preserving IP [23]
BM	Black Marker [20]
RP	Random Permutation [20]

Table 1: Abbreviations for anonymization algorithms and classes of algorithms

Logs are the source of information about entities. When discussing a particular log, we assume that the adversary has some form of that log (either anonymized or unanonymized) in his possession. We will denote logs using the symbol $\mathcal{L}$. We may occasionally place a subscript after this for the few attacks in which multiple logs are involved.

3.4 Anonymized/Unanonymized Data

We will denote unanonymized data using an underline notation. For example, an unanonymized IP address may be denoted as $\underline{ip\_addr}$.

We will denote anonymized data using an $\overline{overline}$ notation, followed by a backslash, and then an abbreviation to note what algorithm (or class of algorithms) are used to anonymize. For example, an IP address anonymized using a consistent pseudonym algorithm (such as prefix-preserving IP anonymization [23] or random permutation [20]) may be denoted as $\overline{ip\_addr}\backslash CP$. A list of abbreviations for use in this model can be found in Table 1.

3.5 Patterns

Patterns are a way to match information in the adversary's knowledge set. Because of the large variety of computations matching information can involve, we choose not to formalize the construction of a pattern besides specifying its inputs and outputs. However, Coull's adversarial model [3] may make a good model for a pattern in many cases of fingerprinting or structure recognition attacks. While our model makes use of very unspecified patterns, we view this as being a powerful construct that can be broadly applied.

Patterns take a set of input attributes, which may be either anonymized or unanonymized. The key to these inputs is that they are *pre-conditions*, in that the pattern cannot be matched against if these inputs are not available. For example, a data injection pattern using unanonymized, obscure TCP options cannot be recognized if those options are anonymized (especially by a Black Marker algorithm). Likewise, a pattern relying on packet sizes cannot be applied to a NetFlow log, as it does not report individual packets, but rather, the size of the entire flow between hosts. Thus, if our goal is to stop the application of a pattern, we simply need to prevent the adversary from gaining knowledge of the pattern's pre-conditions.

Patterns are denoted using the symbol $\mathscr{P}$. We have formalized the pre-conditions by listing them in angle brackets after the name of the pattern. For example, if pattern $\mathscr{P}_{ex}$ has $\mathscr{e}.\overline{ip\_addr}\backslash CP$ and $\mathscr{e}.\underline{tcp\_options}$ are prerequisites, then the full notation will be $\mathscr{P}_{ex}\langle \mathscr{e}.\overline{ip\_addr}\backslash CP, \mathscr{e}.\underline{tcp\_options}\rangle$.

3.6 Adversarial Means

Below we consider four "means", or capabilities, of an adversary in our model. Each of these means is in relation to an attribute in a record (usually an attribute belonging to an entity of interest). In combination, these means (along with the $Knows()$ assertion) can describe any attack. The '$\rightarrow$' symbol represents that a means produces a given value. For example, $f(\ldots) \rightarrow g$ says that the means f, when successfully used, produces information g.

3.6.1 *The Observe() Means*

Form: $Observe(\mathscr{L}, \mathscr{e}.attr) \rightarrow \mathscr{e}.attr$

The $Observe()$ means represents knowledge that an adversary can directly learn from a log. This information may be either anonymized or unanonymized, as it is presented in the log in question. $Observe()$ cannot directly un-anonymize data, nor can it infer data not found in the log.

As an example, the act of observing an entity's IP address from a log that has not been anonymized can be represented as $Observe(\mathscr{e}_t.\underline{ip\_addr}) \rightarrow \mathscr{e}_t.\underline{ip\_addr} \Rightarrow \mathscr{K}$ (we note that some authors have considered this a simple attack [11]).

A second example, involving a port number anonymized using the black marker algorithm, is very similar:
$Observe(\mathscr{e}_t.\overline{port\_num}\backslash BM) \rightarrow \mathscr{e}_t.\overline{port\_num}\backslash BM \Rightarrow \mathscr{K}$.

3.6.2 *The Compromise() Means*

Form: $Compromise(\mathscr{e}.\overline{attr}\backslash alg) \rightarrow \mathscr{e}.\underline{attr}$

The $Compromise()$ means directly reveals the unanonymized version of an anonymized attribute, which can then be added to the adversary's knowledge set.

An example use of $Compromise()$ is an attack on a cryptographically weak anonymization algorithm, such as a dictionary attack against a hash of IPv4 addresses. There are only 2^{32} possible addresses (which can be even further constrained due to other knowledge available to the adversary), and constructing a dictionary of these addresses would take under an hour at the relatively modest rate of 1.2 million address hashes per second[1]. This attack can be represented as $Compromise(\mathscr{e}_t.\overline{ip\_addr}\backslash MD5) \rightarrow \underline{ip\_addr} \Rightarrow \mathscr{K}$.

We do not anticipate that most adversaries will have a $Compromise()$ ability for most or any attributes and that it primarily will serve to identify adversaries who can take advantage of weak anonymization algorithms. The attacks performed using the $Compromise()$ means will generally be those we have classified as *Cryptographic*.

3.6.3 *The Inject() Means*

[1]We ran an Openssl benchmark and found that it could do over 1.2 million md5 hashes per second on a single core Xeon at 3.00 GHz.

Form: $Inject(\mathscr{L}, \mathscr{P}\langle attr1, attr2, attr3, \ldots\rangle) \rightarrow \emptyset$

The $Inject()$ means injects a pattern (involving attributes $attr1, attr2, attr3, \ldots$) into a log for later recognition by *Match()*. It must be performed before any $Observe()$, *Compromise()*, or $Match()$ can be done to an individual log (as an injection must happen *while* the log is being collected, as opposed to the post-collection analysis performed by the other three means). The attacks constructed using an adversary's $Inject()$ means directly map to the class of attacks called "Data Injection" attacks in Section 2.

As an example, consider injecting with a pattern based upon a particular TCP option. If we can construct a pattern $\mathscr{P}_{inj}\langle tcp\_opt\rangle$ using this TCP option, we can denote its injection to log $\mathscr{L}_t$ by $Inject(\mathscr{L}_t, \mathscr{P}_{inj}\langle tcp\_opt\rangle)$.

3.6.4 *The Match() Means*

Form: $Match(\mathscr{e}, \mathscr{P}\langle attr1, attr2, attr3, \ldots\rangle) \rightarrow \mathscr{e}.attr$

The $Match()$ means uses a constructed pattern $\mathscr{P}$ to reveal an unknown attribute of entity $\mathscr{e}$. It relies on the parameter attributes $attr1, attr2, attr3, \ldots$, which may be anonymized or unanonymized, in finding a match and revealing the new attribute (which may also be anonymized or unanonymized). The $Match()$ means **is the primary means by which we expect adversaries to launch attacks**.

An example use of $Match()$ can be seen using a continuation of the TCP option pattern injected in our $Inject()$ means example. If we have an entity $\mathscr{e}_s$ which we are observing (to see if it is the same as our target entity $\mathscr{e}_t$ that injected the pattern), we can use the match means as follows: $Match(\mathscr{e}_s, \mathscr{P}_{inj}\langle tcp\_opt\rangle) \rightarrow \mathscr{e}_t.\overline{ip\_addr}\backslash CP \Rightarrow \mathscr{K}$. If $\mathscr{e}_s$'s TCP options match the pattern $\mathscr{P}_{inj}$, then we infer that $\mathscr{e}_s \equiv \mathscr{e}_t$, and we then know $\mathscr{e}_t$'s anonymized IP address.

3.7 Constructing an Example Attack

From the $Knows()$ assertion and the four means in section 3.6, (as well as a set of patterns), we may construct formal descriptions of various attacks against anonymization. Figure 3 demonstrates a data injection attack using obscure TCP options, which we will explore more thoroughly below. Construction of other attacks is quite similar, and further examples are available in the appendix, as well as in the first author's thesis [6].

The attack given in Figure 3 consists of a single $Knows()$ assertion, and four means. First, we assert that the adversary must know its own unanonymized IP address (that is, the IP address of the entity in the log that injects the data). Second, the adversary must have the means to $Inject()$ a pattern $\mathscr{P}_{inj}$ using TCP options into the log. Next, after obtaining the log, the adversary loops through all the anonymized entities. For each entity, the IP address (anonymized by an algorithm that produces consistent pseudonyms) and the TCP options used by the entity are observed ($Observe()$). Finally, we match the entity against the injected pattern. If the pattern matches ($Match()$) the entity we are examining, we know that the anonymized entity is the same as the adversary's entity, and thus we know the adversary's anonymized IP address.

4. MAPPING ADVERSARIES & ATTACKS

Our taxonomy and adversarial model are useful in and of

```
$Knows(e_{self}.\underline{ip\_addr})$
$Inject(\mathcal{L}, \mathcal{P}_{inj}\langle e_{self}.\underline{tcp\_options}\rangle)$
FOREACH $e \in \mathcal{L}$
    $Observe(\mathcal{L}, e.\overline{ip\_addr}\backslash CP) \Rightarrow \mathcal{K}$
    $Observe(\mathcal{L}, e.\underline{tcp\_options}) \Rightarrow \mathcal{K}$
    $Match(e, \mathcal{P}_{inj}\langle e_{self}.\underline{tcp\_options}\rangle) \rightarrow$
        $e_{self}.\overline{ip\_addr}\backslash CP \Rightarrow \mathcal{K}$
ENDFOR
```

Figure 3: Data Injection Attack Construction: Recover Adversary's Anonymized IP Address

themselves. However, their interrelationship is the key feature that lets network dataset owners develop a safe anonymization policy. This policy can be derived by hand from our model, though our other work [6] has shown how policies safe against a given set of attacks can be derived automatically. Regardless of the method of policy derivation, we seek to show the relationship between our taxonomy and model as a way of selecting the threats to anonymization that concern us.

4.1 Mapping adversaries to attacks

An adversary can perpetrate any attack for which it has the prerequisite knowledge and means. Formally, consider an adversary χ with a set of means and knowledge M_χ. An adversary can perpetrate any attack that can be constructed from the elements of M_χ. In other words, an attack α that requires means and knowledge μ_α can be perpetrated by χ if and only if $\mu_\alpha \subseteq M_\chi$.

Since any attack that can be perpetrated by an adversary has a means and knowledge set that is a subset of the adversary's means and knowledge set (that is, $\mu_\alpha \subseteq M_\chi$), we can explore all possible combinations of the means and knowledge of the adversary to find all attacks possible, given an adversary. Specifically, since the power set $P(M_\chi)$ contains all subsets of M_χ, we know that every attack that χ can perpetrate is a member of $P(M_\chi)$. In the case that $|M_\chi|$ is finite, we can even enumerate all the elements of $P(M_\chi)$, and thus all of χ's possible attacks.

However, note that many (if not most) of the elements of $P(M_\chi)$ are not actually attacks. For example, an attack that does not have at least one $Observe()$ means does not read any information from a log, and therefore is not an attack on log anonymization. Second, only $Compromise()$ and $Match()$ actually reveal new information, and thus an attack must contain at least one $Compromise()$ or $Match()$ means. Finally, note that patterns, $Compromise()$, and $Match()$ all rely on specific information that has been read from a log, and thus, to construct an attack using any of these, we must also include their $Observe()$ counterparts. This limits the set of potential attacks that we can construct. Because of this, if we call our mapping of adversaries to attacks γ, then we might say $\gamma(\chi) \subsetneq P(M_\chi)$.

4.2 Deriving adversaries from attacks

We can also establish a reverse mapping, that is, γ^{-1} from an attack to the set of adversaries that can perpetrate it. While this adversary set will generally be uncountably infinite (as there are an uncountably infinite number of patterns we may use for a $Match()$), we can discuss a *minimal adversary* that can perpetrate α simply as the adversary with exactly the means and knowledge such that α can be constructed. Thus, a minimal adversary χ_{min} for attack α is simply the adversary with means and knowledge $M_{\chi_{min}} = \mu_\alpha$.

Given a *set* of attacks A, we can construct a minimal adversary χ_{min} by simply taking the union of every means and knowledge set required for the attacks in A, formally,

$$A \subseteq \gamma(\chi_{min}) \iff \bigcup_{\alpha \in A} \mu_\alpha \equiv M_{\chi_{min}}$$

Note that $A \subseteq \gamma(\chi_{min})$. This is because it is entirely possible for the union of two or more attacks' prerequisite means and knowledge sets to combine and allow for an additional attack. The new attack must have all of its prerequisites in the union of the original attacks' means and knowledge sets. In some cases (for example, attacks #8 and #9 in our taxonomy), an attack α_1 may have a means and knowledge set that is a superset of another attack α_2's means and knowledge set, that is, $\mu_{\alpha_2} \subset \mu_{\alpha_1}$, and in this rare case, the minimal adversary capable of perpetrating α_1 will also be able to carry out α_2. We believe that this will only occur with *highly similar* attacks.

4.3 Example Mappings

Consider an adversary χ, who can inject data into a log as it is being collected, and then recover the log in anonymized form. However, assume that the log is anonymized such that only pseudonyms remain for the IP addresses, and all other fields except the timestamps and TCP options are removed. If the adversary is familiar with anonymization attack literature, he or she may be able to use the attacks described in some papers to reproduce the methods used for the attacks. Assume that the adversary knows the target entity's unanonymized IP address, clock drift, TCP options, and operating system. A reasonable representation of χ's means and knowledge sets are:

$$M_\chi = \{Observe(\mathcal{L}, e.\overline{ip\_addr}\backslash CP), Observe(\mathcal{L}, r.\underline{timestamp}) \\ Observe(\mathcal{L}, e.\underline{tcp\_options})\ Match(e.\underline{timestamps[]}, \mathcal{P}_{td}), \\ Match(e.\underline{timer\_drift}, \mathcal{P}_t\langle e_t.\underline{timer\_drift}\rangle), \\ Match(e, \mathcal{P}_{inj}\langle e_t.\underline{tcp\_options}\rangle), Inject(\mathcal{L}, \mathcal{P}_{td}), \\ Inject(\mathcal{L}, \mathcal{P}_{inj}\langle e_t.\underline{tcp\_options}\rangle)\}$$

$$\mathcal{K}_\chi = \{e_t.\underline{ip\_addr}, e_t.\underline{timer\_drift}, e_t.\underline{tcp\_options}, e_t.\underline{os}\}$$

By examining Figure 3, we can see that all the necessary knowledge and means are present to launch the TCP options injection attack—we call this α_{inj}. We can also see from Figure 5 that we have the means for Kohno *et al*'s hardware timer drift attack [7], which we call α_{tim}. α_{inj} and α_{tim} are members of $\gamma(\chi)$, and thus, both are possible for χ to perpetrate.

Our example χ above is not the minimal adversary that can perpetrate both attacks: the means $Inject(\mathcal{L}, \mathcal{P}_{td})$ and the knowledge $e_t.\underline{os}$ are unnecessary for either attack (neither are present in $\mu_{\alpha_{inj}}$ or $\mu_{\alpha_{tim}}$). However, these extras may make other attacks possible for χ. For example, it may be possible to use $Inject(\mathcal{L}, \mathcal{P}_{td})$ to inject a timing pattern into the log based on the adversary's known unique timer drift.

In order to protect against the attacks that we known χ can perform, we need to change the anonymization policy, which changes the available $Observe()$ means. For example, we could anonymize IP addresses with a black marker algorithm, making traffic from various entities much harder (or impossible) to distinguish. We could also anonymize timestamps and TCP options.

5. RELATED WORK

5.1 Prior Taxonomies

Pang and Paxson [15] were the first to suggest an attack classification in the domain of log anonymization. They described four basic types of attacks. However, these classifications were neither complete nor mutually exclusive [22]. Furthermore, the classifications are too broad. The plethora of attacks that fall within any one class makes elimination of all attacks in a class, while feasible, a serious detriment to the utility of a log.

Slagell *et al*'s classification [22] introduces two additional categories of attacks not considered by Pang and Paxson, and reorganizes the previous, sometimes-overlapping categories into three mutually exclusive sets. This reclassification addresses the first two problems of Pang and Paxson's taxonomy, but it is still too coarse. We require a more fine-grained taxonomy to useful correlate with our adversarial model.

5.2 Prior Adversarial Models

Coull et al. recently developed their own adversarial model in conjunction with an entropy measurement [3]. While this model is convincing from an information theoretic perspective, it gives no hard guarantees on specific, known attacks—even ignoring all active attacks. We view Coull et al.'s model and the associated metrics as an additional tool that compliments our own adversarial model. An analogous contrast would be between signature and anomaly based intrusion detection.

The inspiration for our model comes from Avione's generic model for attacking secure RFID tags [1]. Avione's model consists of three channels: "forward", "backward", and "memory". The adversary also has a set of five "means", or capabilities: *Query(), Send(), Execute, Execute*()*, and *Reveal()*. Our model has two (implicit) channels. The Forward channel is data injected into the log, and the Backward channel is the log itself. We have no need for an equivalent to the memory channel. Our means roughly map to theirs as follows: *Query()* $\iff$ *Observe()*, *Send()* $\iff$ *Inject()*, *Execute()* $\iff$ *Match()*, and *Reveal()* $\iff$ *Compromise()*.

6. CONCLUSIONS AND FUTURE WORK

We have presented a new taxonomy and adversarial model to describe attacks against network log anonymization. This taxonomy is the first to explore and classify a large variety of attacks, and it is the first to classify by a rigorous, empirical evaluation of attack properties. Our adversarial model is capable of capturing all currently known anti-anonymization attacks, and we believe it will be able to incorporate new attacks as they are discovered. Furthermore, we are able to map an adversary constructed with a specific means and knowledge set into a class of potential attacks (and vice versa).

The impact is to provide a methodology that allows organizations to create safer anonymization policies for existing log sanitizers. By allowing data owners to make more informed decisions, they can more easily balance the trade-offs between utility and trust, thus allowing for more collaboration. Hence, this work compliments our other work in this area evaluating how anonymization affects utility and developing a flexible anonymization framework [10, 20].

Like any new taxonomy, its value will only be determined in the future by those who have found it useful towards their goals. We have found it useful as it maps so nicely into a first order predicate logic that can express constraints on what cannot be anonymized for utility and what must be anonymized to prevent specific attacks or classes of attacks [6]. In the future, we plan on using this predicate logic along with our anonymization framework to see if they meet specified utility and trust requirements.

7. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation (NSF) under Award No. CNS 0524643. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF.

8. APPENDIX

```
$Observe(\mathcal{L}, e_t.\overline{ip\_addr}\backslash MD5) \Rightarrow \mathcal{K}$
$Compromise(e_t.\overline{ip\_addr}\backslash MD5) \rightarrow e_t.\underline{ip\_addr} \Rightarrow \mathcal{K}$
```

Figure 4: Cryptographic Attack Construction - Dictionary attack on MD5-hashed IPv4 Addresses

```
$Knows(e_t.\underline{ip\_addr})$
$Knows(e_t.\underline{timer\_drift})$
FOREACH $e \in \mathcal{L}$
   $Observe(\mathcal{L}, e.\overline{ip\_addr}\backslash CP) \Rightarrow \mathcal{K}$
   FOREACH record $r \in e$
      $Observe(\mathcal{L}, r.\underline{timestamp}) \rightarrow e.\underline{timestamps[]} \Rightarrow \mathcal{K}$
   ENDFOR
   $Match(e.\underline{timestamps[]}, \mathcal{P}_{td}) \rightarrow e.\underline{timer\_drift} \Rightarrow \mathcal{K}$
   $Match(e.\underline{timer\_drift}, \mathcal{P}_t\langle e_t.\underline{timer\_drift}\rangle) \rightarrow$
      $e_t.\overline{ip\_addr}\backslash CP \Rightarrow \mathcal{K}$
ENDFOR
```

Figure 5: Machine Attribute Fingerprinting Attack Construction - Hardware Timer Drift

```
$Observe(\mathcal{L}, e_t.\overline{ip\_addr}\backslash CP) \Rightarrow \mathcal{K}$
FOREACH record $r \in e_t$
   $Observe(r.\underline{timestamp}) \Rightarrow \mathcal{K}$
   $Observe(r.\underline{port}) \Rightarrow \mathcal{K}$
   $Observe(r.\underline{flow\_size}) \Rightarrow \mathcal{K}$
   $Match(r, \mathcal{P}_{sessions}\langle r.\underline{timestamp}\rangle) \rightarrow$
      $e_t.\underline{sessions[]} \Rightarrow \mathcal{K}$
ENDFOR
FOREACH session $s \in ent_t.\overline{sessions[]}$
   $Match(s, \mathcal{P}_{websites}\langle s.\underline{port}, \overline{s.\underline{flow\_sizes}}\rangle) \rightarrow$
      $s.\underline{website} \Rightarrow \mathcal{K}$
ENDFOR
```

Figure 6: Port-based Fingerprinting Attack Construction: Web Browsing Activity Identification

9. REFERENCES

[1] Avoine, G., "Adversary Model for Radio Frequency Identification," *Cryptology ePrint Archive*, Report 2005/049, 2005.

[2] Bethencourt, J., Franklin, J., Vernon, M., "Mapping Internet Sensors with Probe Response Attacks," 14th *USENIX Security Symposium*, Aug., 2005.

[3] Coull, S., Wright, C. V., Keromytis, A., Monrose, F., and Reiter, M., "Taming the Devil: Techniques for Evaluating Anonymized Network Data,", 15th *Network and Distributed System Security Symposium (NDSS '08)*, Feb., 2008.

[4] Coull, S., Wright, C. V., Monrose, F., Collins, M., and Reiter, M., "Playing Devil's Advocate: Inferring Sensitive Information from Anonymized Network Traces," 14th *Network and Distributed System Security Symposium (NDSS '07)*, Feb., 2007.

[5] Coull, S., Collins, M., Wright, C. V., Monrose, F., and Reiter, M., "On Web Browsing Privacy in Anonymized NetFlows," 16th *USENIX Security Symposium*, Aug., 2007.

[6] King, J., "A Taxonomy, Model, and Method for Secure Network Log Anonymization," *Master's Thesis*, University of Illinois at Urbana-Champaign, Apr., 2008.

[7] Kohno, T., Broido, A., and Claffy, K. C., "Remote Physical Device Fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, Apr., 2005.

[8] Koukis, D., Antonatos, S., and Anagnostakis, K., "On the Privacy Risks of Publishing Anonymized IP Network Traces," 10th *IFIP Open Conference on Communications and Multimedia Security (CMS '06)*, Oct., 2006.

[9] Koukis, D., Antonatos, S., Antoniades, D., Markatos, E., and Trimintzios, P., "A Generic Anonymization Framework for Network Traffic," *IEEE International Conference on Communications (ICC '06)*, Jun., 2006.

[10] Lakkaraju, K. and Slagell. A., "Evaluating the Utility of Anonymized Network Traces for Intrusion Detection," 4th SecureComm Conference, Sep., 2008.

[11] Lincoln, P., Porras, P., and Shmatikov, V., "Privacy-preserving Sharing and Correction of Security Alerts," 13th *USENIX Security Symposium*, Aug., 2004.

[12] Mc Hugh, J., "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory," *ACM Transactions on Information and System Security*, Vol. 3(4), Nov., 2000.

[13] Øverlier, L., Brekne, T., and Årnes, A., "Non-expanding Transaction Specific Pseudonymization for IP Traffic Monitoring," 4th *International Conference on Cryptology and Network Security (CANS '05)*, Dec., 2005.

[14] Pang, R., Allman, M., Paxson, V., and Lee, J., "The Devil and Packet Trace Anonymization," *SIGCOMM Computing Communications Review*, Vol. 36(1):29–38, 2006.

[15] Pang, R. Paxson, V., "A High-level Programming Environment for Packet Trace Anonymization and Transformation," *ACM Conference of the Special Interest Group on Data Communication (SIGCOMM '03)*, Aug., 2003.

[16] Porras, P., and Shmatikov, V., "Large-scale Collection and Sanitization of Network Security Data: Risks and Challenges," *Workshop on New Security Paradigms (NSPW '06)*, Sep., 2006.

[17] Ramaswamy, R., and Wolf, T., "High-Speed Prefix-Preserving IP Address Anonymization for Passive Measurement Systems," *Networking, IEEE/ACM Transactions on Networking*, Vol. 15(1), Jan., 2007.

[18] Ribeiro, B., Chen, W., Miklau, G., and Towsley, D., "Analyzing Privacy in Enterprise Packet Trace Anonymization," 15th *Network and Distributed System Security Symposium (NDSS '08)*, Feb., 2008.

[19] Sicker, D., Ohm, P., and Grunwald, D., "Legal Issues Surrounding Monitoring during Network Research," *Internet Measurement Conference (IMC '07)*, Jun., 2007.

[20] Slagell, A., Lakkaraju, K., and Luo, K., "FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs," 20th *Large Installation System Administration Conference LISA '06)*, Dec., 2006.

[21] Slagell, A., Li, Y., and Luo, K., "Sharing Network Logs for Computer Forensics: A New tool for the Anonymization of NetFlow Records," *First Computer Network Forensics Research Workshop*, held in conjunction with IEEE SecureComm, Sep., 2005.

[22] Slagell, A., and Yurcik, W., "Sharing Computer Network Logs for Security and Privacy: A Motivation for New Methodologies of Anonymization," *First International Workshop on the Value of Security through Collaboration (SECOVAL '05)*, Sep., 2005.

[23] Xu, J., Fan, J., Ammar, M., and Moon, S., "Prefix-preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme," 10th *IEEE International Conference on Network Protocols (ICNP '02)*, Nov., 2002.

[24] Zalewski, M., and Stearns, W., "Passive OS Fingerprinting Tool," www.stearns.org/p0f/README, viewed Aug. 1, 2008.

[25] Zhang, Q. and Li, X., "An IP Address Anonymization Scheme with Multiple Access Levels," *Advances in Data Communications and Wireless Networks (ICOIN '06)*, Jan., 2006.

[26] Zhang, Q., Wang, J., and Li, X., "On the Design of Fast Prefix-Preserving IP Address Anonymization Scheme," *International Conference on Information and Communications Security (ICICS '07)*, Dec., 2007.