Structure Preserving Anonymization of Router Configuration Data

David A. Maltz, Jibin Zhan, Gísli Hjálmtýsson, Albert Greenberg, Jennifer Rexford, Geoffrey G. Xie, Hui Zhang

***Abstract*—A repository of router configuration files from production networks would provide the research community with a treasure trove of data about network topologies, routing designs, and security policies. However, configuration files have been largely unobtainable precisely because they provide detailed information that could be exploited by competitors and attackers. This paper describes a method for anonymizing router configuration files by removing all information that connects the data to the identity of the originating network, while still preserving the structure of information that makes the data valuable to networking researchers.**

Anonymizing configuration files has unusual requirements, including preserving relationships between elements of data, anonymizing regular expressions, and robustly coping with more than 200 versions of the configuration language. Conventional tools and techniques are poorly suited to the problem. Our anonymization method has been validated with a major carrier, earning unprivileged researchers access to the configuration files of thousands of routers in hundreds of networks. Through example analysis, we demonstrate that the anonymized data retains the key properties of the network design. The paper sets out techniques that could be used in an attempt to break the anonymization and concludes the method is most applicable to enterprise networks. When applied to backbone networks, which are few in number and many of whose properties can be publicly measured, the anonymization might be broken by fingerprinting techniques described in this paper.

***Index Terms*—Data anonymization, router configuration**

I. Introduction

By far the best source of design information available today for an IP network is the set of configuration files running on its routers.[1] Each of these files, known as a "config", contains the complete set of commands used to define the behavior of that single router. Taken together over all the routers, the set of configs for a network define the overall behavior of the network. Access to the config sets for production networks would bring tremendous benefits to a wide group of networking researchers. For example, an accurate IP-level network topology can typically be directly derived from the configs. The parameters governing the intricate interactions among routing protocols and policies that could only be estimated otherwise are explicit in the configuration files, making it possible to develop more precise analysis techniques for evaluating essential network properties such as the robustness of the routing design [1].

However, configs are held as closely-guarded secrets for some of the same reasons that make them valuable for research. They reveal internal details of the network design, and potentially expose business secrets such as the owner's organizational structure and clientele. They show where a company has resources and capacity, and where its network bottlenecks are. Further, they may expose potentially embarrassing configuration mistakes and security vulnerabilities that could be remotely exploited. Without a high assurance that sensitive information will not leak, a network owner will hesitate to grant access to its configs to anyone outside its staff.

Thus, an important question arises: How can config sets be sanitized to avoid leaking sensitive information? In some scenarios, e.g., for controlling the scope of access by a *trusted* group, it may be sufficient to remove just specific types of information from the configs, e.g., the identity of the network's customers, or the geographical locations of its facilities. In this paper, we investigate the feasibility of *anonymizing a config set* – hiding the identity of owning organization itself. Such anonymization provides the most general form of protection because severing the link between the configuration files and the identity of the network owner means that any information learned from the configurations cannot be exploited against the owner. Fortunately, for many networks the technical engineering details of the network and its routing design, the parts most interesting to the research community, are not considered business secrets or competitive advantages of the owning organization. This means the processing of the config sets need not necessarily strip out the information with value to networking researchers, if a technique can be developed that anonymizes the owner of the files while retaining this information.

We present a detailed formulation of the problem of anonymizing router configuration file sets. We qualify the two equally important but often competing requirements – *owner-identity anonymization* and *relationship preservation* – and outline a methodology to validate that they are met. We identify key challenges in developing an acceptable anonymization method and consider potential attacks against it. Guided by this formulation, we have crafted a first working method for

This research was sponsored by the NSF under awards ANI-0085920, ANI-0331653, ANI-0114014, and CNS-0721574. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Microsoft, CMU, AT&T, NSF, or the U.S. government.

[1]Ultimately, we believe that researchers should not need to work at the level of the configs themselves, but with a higher-level representation that abstracts away the idiosyncrasies of particular configuration languages and exposes the critical information. However, developing such a data model is an extremely difficult task, one that must be driven and validated by examples of how configurations are used in real networks. We see our work as the first logical stepping stone to the creation of a high-level representation of configuration data.

```
1   hostname cr1.lax.foo.com
2   !
3   banner motd ^C
4     FooNet contact xxx@foo.com
5     Access strictly prohibited!
6   ^C
7   !
8   interface Ethernet0
9    description Foo Corp’s LAX Main St offices
10   ip address 1.1.1.1 255.255.255.0
11  !
12  interface Serial1/0.5 point-to-point
13   description cr1.sfo-Serial3/0.2
14   ip address 66.253.32.85 255.255.255.252
15  !
16  router bgp 1111
17   redistribute rip
18   neighbor 66.253.160.68 remote-as 701
19   neighbor 66.253.160.68 route-map UUNET-import in
20   neighbor 66.253.160.68 route-map UUNET-export out
21  !
22  route-map UUNET-import deny 10
23   match as-path 50
24   match community 100
25  route-map UUNET-import permit 20
26  route-map UUNET-export permit 10
27   match ip address 143
28   set community 701:1234
29  !
30  access-list 143 permit 1.1.1.0 0.0.0.255
31  ip community-list 100 permit 701:7[1-5]..
32  ip as-path access-list 50 permit (_1239_|_70[2-5]_)
33  !
34  router rip
35   network 1.0.0.0
```

Fig. 1. Excerpts of a router configuration file.

config anonymization[2].

Anonymizing configs is challenging for several reasons: First, there are numerous ways in which configs can leak information that would allow an attacker to break the anonymization. For example, public AS numbers and IP addresses can be easily connected with the owner. Even the number and location of peering points to other networks that can be gleaned from configs might uniquely identify a network. Second, there is no consistent grammar for the configuration language, so conventional compiler tools and techniques are poorly suited to the problem. Third, the anonymization needs to support a diverse set of research goals. Fourth, the anonymization process must be fully automated to avoid human errors and gain the acceptance of network operators.

The anonymization method described in this paper makes an important step towards overcoming these challenges. It has been validated with a major carrier, earning unprivileged researchers access to the configuration files of thousands of routers in hundreds of enterprise and backbone networks, and used by three other organizations in sharing their config sets. A complete implementation of the anonymizer is publicly available and open to improvement by the community [2].

II. The Nature of Configuration Files

Figure 1 shows command lines like those found in a pre-anonymized configuration file. Typical configs in production networks vary from 50 to 10,000 lines — in our dataset of 7655 routers, the 25th percentile was 183 lines and 90th percentile was 1123 lines.

Lines 8–14 define two interfaces and assign them IP addresses, with free text comments used to indicate where these interfaces connect. Line 16 defines a BGP process and configures it as a speaker for the public Autonomous System Number (ASN) 1111. Lines 18–20 declare an EBGP session with a router at 66.253.160.68, presumably inside the UUNET network as the remote AS has UUNET’s ASN (701). Lines 22–28 define the route-maps used by BGP in terms of the access-lists defined in lines 30–32. Line 30 selects IP addresses matching 1.1.1/24. Line 31 uses a regular expression to match any BGP community attribute value coming from UUNET (701) between 7100 and 7599, and line 32 uses another regular expression to match any AS path that contains AS 1239, or one of UUNET’s non-US ASes (702-705).

The config illustrates several common relationships between information elements. The **uses** relationship between the BGP process in line 19 and the routing policy definition in lines 22–25 is established by the name “UUNET-import”. The RIP routing protocol in line 35 is configured to run over the interface in line 8 by the **subnet contains** relationship between the prefix 1.0.0.0/8 and the address 1.1.1.1.

Anonymizing this configuration requires removing or transforming: (1) the comments; (2) the owner’s public AS number (here 1111) (3) the publicly routable IP addresses (e.g., 1.1.1/24), all of which directly identify Foo Corp; and (4) all data about external peers (e.g., neighbor IP addresses, AS numbers, route-map names, community attributes), which while (probably) innocuous individually could build a picture identifying Foo Corp.

III. Challenges

In this section, we first define the problem of config set anonymization. We then provide more detail about the specific challenges that we had to address while developing a working method of config anonymization. The difficulties of anonymizing configs can be broken into two broad classes of challenges. First is finding all the elements of a configuration that can leak identity information. Second is anonymizing each component so that the relationships between information in the configs are preserved.

A. *The Problem of Config Anonymization*

In config set anonymization, the configuration files from a set of routers in a network that belongs to an organization are processed using the anonymization method described in this paper and then made publicly available. The goal of anonymization is to establish two properties for the anonymized config set. First, minimize the confidence with which an attacker can label a publicly released config set with the identify of the organization that owns the routers. Second, each anonymized config set must be structurally and functionally identical to the organization’s network, such that if a network were to be created with similar hardware and configured using the information in the anonymized config set, it would behave as the real network does.

[2]We have implemented our approach for Cisco IOS, but we believe the techniques are directly applicable to JunOS and other router configuration languages as well.

As a caveat to the second property, anonymization can preserve only those aspects of the network that are explicit in the text of the configs. For example, it cannot preserve the latency between routers in the configs since latency is not explicitly described in the configs.

We assume the attacker has the following capabilities. It has access to any public information or attributes for a large number of networks. It is capable of sending probe traffic across the public Internet from a wide variety of location, and so has knowledge of attributes discoverable by external observation (e.g., RocketFuel [3]). The attacker also has access to a large number of anonymized configuration sets, which may or may not overlap with the set of networks for which the attacker has public information.

B. Finding Elements to Anonymize

At first impression, it might seem that parsing the configuration is the simplest way to find the elements of a config that must be anonymized. However, attributes of the underlying grammar make existing compiler tools poorly suited for the task.

No explicit grammar is available: While somewhat surprising an explicit and complete grammar does *not* appear to be publicly available. Moreover, small, but syntactically significant changes occur between Cisco Internet Operating System (IOS) versions and each type of device supports slightly different commands. All but the most trivial networks have routers running different versions of IOS (the routers in our dataset run over 200 different IOS versions). Consequently, even a complete grammar for a particular version would typically not be applicable for all routers in a study — not even within a single network.

Grammar is poorly suited for standard compiler tools: The language interpreted by the Cisco Command Line Interface (CLI) is described in manuals by a regular expression grammar, and thus in principle is of relatively low complexity. However, in contrast to the grammar of programming languages, IOS supports a huge set of commands,[3] each specified as a separate grammar rule, and it recognizes a very large set of keywords that appear in different orders depending on the command. Inconsistencies and ambiguities abound. For example, sometimes parameters are positional and sometimes attribute-value pairs; other commands allow multiple values for some parameters. Even *space* is not consistently a separator. These specifics furthermore depend on the particular IOS version, resulting in all combinations and variations potentially appearing in a single network.

Ensuring completeness is difficult: The huge number of distinct commands not only make the CLI language problematic for traditional compiler tools, but would also make it very challenging to ensure correct anonymization through annotation of the complete grammar. Even if the complete grammar were successfully annotated, the effort would bring questionable value, as only a small fraction of the commands are of interest for the study of IP networks. This fact highlights a key advantage of our approach, as our anonymization operates across commands mostly without grammatical or semantic discrimination, as explained in Section IV.

[3]Over 3000 commands for Authorization, Authentication and Accounting (aaa) alone.

C. Relationship Preserving Anonymization

Each element of a configuration that is altered to hide the identity of the owner must be anonymized in a way that preserves the relationship between elements, even when not all relationships are known at anonymization time. Even several known relationships are particularly challenging to maintain.

Preserving the structure of addresses: Configuration files make extensive use of the **subnet contains** relationship to associate elements of the configuration (e.g., the RIP routing protocol in line 35 and the interface in line 10), so the relationship must be preserved by anonymization.

There are also restrictions on how addresses are anonymized. Some addresses used in configuration files have special meanings and must not be modified at all, e.g., netmasks in lines 14 and 30 (255.255.255.252 and 0.0.0.255). Also, older commands, such as those for configuring RIP and EIGRP, implicitly assume classful IP addresses, so the mapping must also be class preserving: mapping addresses with class A prefixes to addresses with another class A prefix. Additionally, it improves human readability in the post-anonymization configs if subnet addresses (i.e., addresses with a host part of all zeros such as 128.2.0.0) are mapped to other subnet addresses (e.g., 135.9.0.0).

Hashing public AS numbers: Although most integers found in configuration files do not leak information, AS numbers can. Anonymizing individual AS numbers with a random permutation is trivial, but they can also be referenced by regular expressions, as shown in lines 31–32 of Figure 1, which then must be rewritten to reflect the permuted values.

Maintaining referential integrity: All identifiers must be anonymized in a consistent manner so that, for example, the **uses** relationship between the routing policy statement at line 19 and the policy definition at lines 22–25 created by the shared identifier "UUNET-import" is maintained.

IV. Anonymization Method

We first describe our general approach, which anonymizes most parts of the configuration files, and then explain in detail how particularly troublesome or important aspects of the configurations are handled.

A. Basic Method

Being unable to know *a priori* which strings can leak information about the identity of the network owner, the most conservative approach is to cryptographically hash *every* string that is not known to be innocuous. A *pass-list* of "unprivileged" tokens was created by building a web-walker that string scraped the Cisco IOS command reference guides. In theory, most Cisco keywords will appear somewhere in the guides, and non-keywords used in the guides are so common they cannot leak information. All non-numeric tokens found in the configurations are checked against this pass-list, and any tokens not found are hashed using SHA1 digests [4]: this anonymizes the names of class-maps, route-maps, and any other strings that could hold privileged information. All IP addresses are hashed using a modified version of the tcpdpriv algorithm [5]. Our version of the algorithm preserves the important properties of IP address relationships that are fundamental to the network design and routing logic, such as the classes of the IP addresses in addition to longest prefix matching. Simple integers are generally not anonymized.

B. Handling Expressions Requiring Context

While our goal is to avoid creating anonymization rules that depend on context so that the anonymizer is robust against different versions of IOS, there are situations which require context to handle properly. In these situations, we add *rules* to the anonymizer written using regular expressions that establish context. In practice, we have developed a set of 28 rules that is sufficient for anonymizing the 200-plus IOS versions we have tested them on.

We use two rules to segment all words in the configs into tokens before consulting the pass-list, so identifiers like `ethernet0/0` become a string "ethernet" that matches against the pass-list and a non-alphabetic remainder "0/0" that doesn't need anonymization. Without this step, the string "Ethernet0/0" would not have been found in the pass-list and would have been hashed, destroying valuable information about the interface type.

Although all "unsafe" words in comments would be hashed by our basic method, the arrangement of pass-list words in comments can still leak information. For example, "global" and "crossing" are both in the pass-list, but the string "global crossing" in a comment must be anonymized, as it is the name of a major ISP. Since there is no means short of human inspection to reliably find these leaks, we use three rules to strip out all comments, including multi-line comments like the banner in lines 3–6 of Figure 1. Among a dataset of 173 networks, an average of 1.5% of the words were found to be comments and removed (90th percentile 6%).

An additional four rules are needed to anonymize miscellaneous information, including phone numbers in dialer strings, and so on.

C. Anonymizing IP Addresses

Two of the best prefix preserving IP address anonymization schemes are due to Xu [6] and Minshall[5]. Xu's has the property that very little state must be shared to consistently map addresses, making it amenable to parallelization, while Minshall's requires a data-structure to store the mapping as it is created.

However, anonymizing configs requires that the IP anonymization scheme has the properties discussed earlier, such as being class-preserving and subnet-address-preserving. We have found that using a data-structure-based mapping scheme makes it easier to implement these requirements. By controlling how new entries are added to the data-structure, we can shape the mapping to have the needed properties while

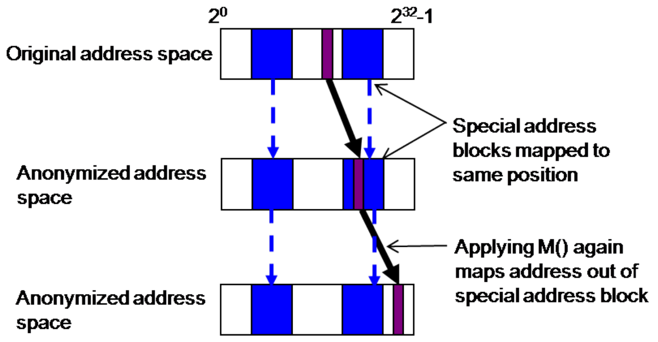

2^0
2^{32}-1
Original address space
Special address blocks mapped to same position
Anonymized address space
Applying M() again maps address out of special address block
Anonymized address space

Fig. 2. Example of a collision while anonymizing IP addresses, caused by the need to use the identity function to anonymize "special" blocks of IP addresses, such as private address space and multicast addresses. Our solution is to recursively map the address until there is no collision.

maintaining as much of the randomness needed for security as possible.

We use an extended version of Minshall's original "-a50" scheme as taken from `tcpdpriv`. Minshall's original algorithm is structure-preserving, by which we mean that it is a mapping function $M(a)$ that maps every IP address a to a randomly chosen address while mantaining the property that for every address b that shares a prefix of length l with a, $M(a)$ and $M(b)$ also share a prefix of length l. To make Minshall's original algorithm class-preserving as well as structure-preseving, we modify it so all "special" IP addresses (e.g., private address space [7], netmasks, multicast addresses) are passed through unchanged. This is challenging, as creating an anonymization function that uses the identity function for mapping part of the address space and $M()$ for mapping other parts of the address space requires dealing with collisions that occur when the algorithm maps a non-special address a into an address s that falls within the range of special addresses. When such collisions occur, we recursively map s until there is no collision, which we prove below maintains the structure-preserving property of the algorithm.

The handling of collisions is illustrated in Figure 2, where "special" IP address blocks are shown as being mapped from the original address space to the anonymized address space using the identity function. Since a normal IP address a is mapped to a randomly selected portion of the address space, there is the potential the post-image of a, $M(a)$, will be inside a block of special addresses, resulting in a collision. However, while random, the mapping function implemented by Minshall's algorithm is both 1-to-1 and on-to, so a is the only address mapping to $M(a)$. This means a is also the only pre-image that will map to $M(M(a))$, and we can use that value as the anonymized version of a. Subsequent collisions can be resolved by another applicaiton of the $M()$ — this will eventually result in a mapped address that does not fall into a special address block unless $M()$ itself contains a cycle that maps a special address block into sequence of special address blocks that ultimately map back to the initial special address block. This situation can be dealt with by restarting the entire anonymization with a different seed to create a different $M()$. Since the function M() is structure preserving, M(M(...)) is also structure preserving.

D. Anonymizing Autonomous System Numbers

The space of Autonomous System Numbers (ASNs) is divided into public and private ranges, 1-64512 and 64513-65536 respectively. Public ASNs need to be anonymized because they are globally unique and the mapping between public ASN and network owner can be obtained from many sources.

There are no semantics and no relationships embedded in public ASNs,[4] so a random permutation can be used to anonymize them. Since private ASNs are not globally unique and do not leak identity information about the networks, they are not anonymized.

There are two major challenges in anonymizing ASNs. First is to correctly identify every appearance of an ASN in the configuration file. For example, an ASN can appear inside a BGP community attribute. ASNs can also appear in regular expressions that are used in routing policies related to AS-path attributes of BGP routes (line 32). A list of 12 rules is used to locate all the ASNs and ASN regular expressions in the configuration files — this is the most fragile part of our method since ASNs are syntactically indistinguishable from simple integers. Strategies for coping with errors are discussed in Section VII.

The second challenge in anonymizing ASNs is dealing with ASNs that do not explicitly appear in the text of the configs, but are accepted by regular expressions that do appear in the configs. For example, `70[1-3]` accepts ASN 701, 702, and 703. If this regexp appeared in a pre-anonymization config, it would need to be rewritten so that the post-anonymization version accepts whichever ASNs 701, 702, and 703 are mapped to by the random permutation. The use of digit wildcards and ranges in regexps dealing with public ASNs is quite rare, appearing in two of 31 networks studied, because there is little structure among public ASNs for the regexps to exploit. Even among private ASNs, where the network designer is free to impose structure, only 3 of 31 networks use ranges in regexps dealing with private ASNs. Although rare, we feel these cases must still be handled properly. The use of alternation in regexps (e.g., `(_701|1|1239)_.*`) is very common, appearing in 10 networks, but can be easily handled by anonymizing each ASN individually.

We anonymize regular expressions involving digit wildcards and ranges by leveraging automata theory [8]. Using that terminology, the set of ASNs a regexp accepts is called the *language* accepted by the regexp. Since there are only 2^{16} ASNs in BGPv4, we can find the language accepted by the regexp by simply applying the regexp to a list of all 2^{16} ASNs and seeing which it accepts. If the accepted language includes only private ASNs, which do not need anonymization, no changes are required to the regexp. If there are public ASNs in the accepted language, these are all anonymized and the challenge becomes computing a regexp that will accept this new language. Currently, we construct a regexp that is the alternation of all ASNs in the language. For example `70[1-3]`, becomes `701|702|703` and then we anonymize 701, 702 and 703 individually. The resulting regexps could be very long, but this is not a problem when anonymized configs are primarily analyzed by software tools. We could use known polynomial-time algorithms for constructing the minimum finite automata (FA) that accepts the new language and then convert this FA back into a regexp, but we have not had need for this functionality.

[4]An exception is UUNET, which owns the contiguous range of ASNs from 701–705.

E. Anonymizing BGP Community Attributes

BGP community attributes are usually represented by two integers, written as `701:1234`, where the first integer (701) is an ASN and the second (1234) is an ordinary integer (for an example, see line 28 in Figure 1). Community attributes are normally used to inform a directly connected BGP peer how routes carrying the attribute should be handled.

The ASN part of an attribute is located and anonymized as discussed above. To be conservative, we must assume that even the integer part of the attributes used by each network are publicly known and sufficiently distinctive to identify the network owner, so the integer part of community attributes must also be anonymized. This represents a loss of information, but we have chosen to favor anonymity over information wherever such trade-offs must be made.

Like AS numbers, community attributes can appear in regexps (e.g., line 31 in Figure 1), and are anonymized using the same method as AS numbers. Five of the 31 networks used regexps involving communities, but only two networks used regexps with range expressions.

V. Implementation

We implemented the anonymization method described above primarily in `perl` to maximize the portability of the system and to leverage `perl`'s excellent regular expression and text file processing capabilities. For performance reasons, the IP address anonymization code is implemented in C, based directly on Minshall's code. Our current system only works for IPv4 addresses, but the principles on which the system is based apply equally to IPv6 addresses. Pseudocode for the implementation is shown in Figure 3.

```
set of subnets S = ()
ForEach configuration file f {
   S = S + Extract-Subnets(f)
}

PrepareIPAddressAnonymizationFunction(S)

ForEach configuration file f {
   ForEach line l {
      RemoveComments
      AnonymizeIPAddresses
      TokenizeLineAndAnonymizeTokens
      FindAndAnonymizeASNs
   }
}
```

Fig. 3. Pseudocode for configuration anonymizer.

Anonymization begins by identifying all the IP subnets used in any configuration file - this list of subnets will be used later to ensure that any subnet address appearing in a configuration file (e.g., 1.0.0.0) will map to an address ending in zeros in the anonymized configuration file.

The system then initializes the IP address anonymization function. Minshall's algorithm works by mapping the 32 bits in an IP address $a_{31}a_{30}...a_0$ to randomly chosen bits $a'_{31}a'_{30}...a'_0$. However, it records the choices it makes so that any address b whose n most significant bits are the same as a's is mapped to $a'_{31}a'_{30}...a'_{31-n+1}b'_{31-n}...b'_0$, where bits $b'_{31-n}...b'_0$ are randomly chosen but the first n bits are the same as those chosen for a. We make the algorithm preserve classful address space by initializing it to map most significant bits 0 to 0 (i.e., class A to class A), 10 to 10 (i.e., class B to class B), 110 to 110 (i.e., class C to class C), etc.. We make it preserve the zeros in the host part of a subnet address by iterating through the list of extracted subnets S, sorted so that shortest netmasks (i.e., larger subnets) are first. For each subnet with n bits of host address, $a_{31}a_{30}...a_{n+1}a_n...a_0$ where the bits $a_n...a_0$ are all zero, we insert a mapping to $a'_{31}a'_{30}...a'_{n+1}a_n...a_0$ where the a' are chosen randomly and the rest of the bits are still 0.

The heart of the configuration anonymizer processes configs line by line, using state variables to remove multi-line comments.

Each line is matched against regular expressions that identify IP addresses and any associated netmask. Any IP addresses found are anonymized using Minshall's algorithm initialized as described above. Each line is then tokenized, and any token not found in the pass list created by string scraping the IOS manual pages is hashed using SHA1.

TABLE I
EXAMPLE REGULAR EXPRESSIONS FOR IDENTIFYING IOS COMMANDS THAT INVOLVE AS NUMBERS.

```
# find basic ASN
 'router\s+bgp'
 'bgp\s+confederation\s+peers'
 'neighbor\\s+$ipAddrPatt\\s+remote-as'
 'neighbor\s+\w+\s+remote-as'
 'set\s+as-path\s+[\S]+'
# find community strings that might include ASNs
 'set\s+community',
 'set\s+extcommunity\s+\w+',
 'ip\s+community-list\s+\d+',
# find regular expressions that might involve ASNs
 'ip\s+as-path\s+access-list\s+\d+\s+(?:permit|deny)'
```

Finally, a set of regular expressions are run against the line to identify uses of integers that might reveal identifying information, such as ASNs or regular expressions involving ASNs. Any matches are handled as described in Section IV-D. Examples of these regular expressions are shown in Table I — in each case, the ASN would follow the regular expression shown. The complete list is available in the source code [2].

VI. Validation of Accuracy

Anonymization of configuration files is potentially a lossy process. To validate that information relevant to network researchers is surviving the anonymization process unchanged, we use end-to-end tests that compare attributes of the configs pre- and post-anonymization. We developed two suites of tests that a colleague with access to the unanonymized configuration files runs over both the anonymized and unanonymized configurations and then checks for differences in the output.

The first suite of tests verifies that independent characteristics of the configurations are being preserved by comparing properties such as: (a) the number of BGP speakers; (b) the number of interfaces; and (c) the structure of the address space (i.e., number of subnets of each size).

The second suite of tests consists of running our tools to reverse engineer the routing design [1] of a network and comparing the extracted designs. Extracting the routing design makes an excellent test case, as it depends on many aspects of the configuration files being consistent inside each file and across all the files in the network, including physical topology, routing protocol configuration, routing process adjacencies, routing policies, and address space utilization.

While our tests have given us great confidence that our anonymizer implementation preserves information related to routing design, it is possible that other aspects of the configs we have not tested are being altered. As more research is conducted using anonymized configs, we expect the number of tests in the validation suite to increase.

In general, the anonymizer is capable of preserving any relationship between configuration data elements of which it is programmed to be aware. However, the potential exists for there to be implicit relationships between elements of the configuration data that are unknown to the anonymizer, and so are not preserved during the anonymization. For example, it might be "well known" that all addresses used by AS number X have prefix Y. A network designer could conceivably configure some router in his or her network to drop all routes from AS X and other routers to drop all routes to destinations with prefix Y. Using this external information and the unanonymized configurations, it would be possible to determine these two different configurations express the same intent and achieve the same effect. By default, the anonymization process will independently anonymize the AS numbers and IP prefixes, not allowing a reader to infer that the two mechanisms target the same AS. To preserve such implicit relationships, it is necessary to extend the anonymizer to maintain a database of "well-known" external information and actively look for the relationships."

VII. Potential Vulnerabilities

There are two general ways in which the anonymization provided by our approach can be attacked. First, textual information accidentally left inside a post-anonymization configuration file could identify the owner of the network. Second, it might be possible to analyze the configuration files to determine a set of network characteristics that are so unusual they form a unique "fingerprint" of the network. If these characteristics can be measured externally via the public Internet, then a search of all known networks could be made looking for a fingerprint that matches the fingerprint of the configs.

A. Textual Attack Based on Unanonymized Strings

It is very unlikely a textual attack could succeed against the strings in an anonymized configuration file, as we take the extremely conservative approach of stripping all comments from the configs and hashing all strings except those known to be innocuous with the cryptographically secure SHA1 hash (salted with a secret chosen by the network owner). However, it is possible that a non-string that carries identity information could escape the rules we use to find and anonymize them. AS numbers have been the greatest threat, as they are simple integers.

Our best defense against textual attacks is an iterative methodology. After anonymizing configs, we highlight for a human operator lines that seem likely to leak information (usually a tiny fraction of the configs). Lines they believe are dangerous are used to add more rules to the anonymizer. Our experience is that the iterations converge quickly. For example, fewer than 5 iterations were required over 3 months to anonymize 4.3 million lines of configuration from 7655 routers running more than 200 different IOS versions. As an example of a leak-highlighting method, the anonymizer can record all AS numbers it sees before hashing them, and then grep out all lines from the anonymized configs that still include any of those numbers.[5]

B. Attacks on the IP Address Anonymization

Hypothetical [9] and experimental [10] attacks have been proposed on the `tcpdpriv` algorithm on which our IP address anonymization is based. Fortunately, they use the frequency with which addresses appear in a dynamic packet trace — information that is not available from anonymized static configuration files.

However, because the IP address anonymization is structure preserving, the number of subnets of different sizes is the same in pre- and post-anonymization configs. This means an attacker could construct a fingerprint of a network by counting up how many subnets of different sizes (/30s, /29s, /28s, etc.) appear in the anonymized configs. To determine the identity of the network to which the configs belong, he could then send probe packets into candidate networks attempting to measure how many subnets of different sizes each candidate contains from the ICMP Reply or back-scatter packets received. Conceivably this could be done by "pinging" every consecutive address in the address blocks announced by the candidate network in BGP, and using heuristics such as "most subnets have hosts clustered at the lower end of the subnet's address range" to guess where subnet boundaries must lie.

Although remotely determining the address space fingerprint of a real network seems extremely challenging (or impossible in the case of networks behind firewalls or not reachable from the Internet), for this security analysis we will assume it is possible. An open research question is whether address space usage fingerprints are sufficiently unique to enable the identification of networks. Should large numbers of networks have roughly the same fingerprint, the risks of this attack succeeding will be quite low.

C. Attacks Based on Style, Template, or Function

Like all code, there is a large degree of freedom in constructing a config (which is partially what makes their analysis so interesting to the research community). However, this freedom admits stylistic variation that could be used as a fingerprint, such that a well-known style could be used to identify the authors or their organization. Many configs are derived partially from templates, and some organizations make these templates publicly available. Further, some backbone networks offer unique services for their customers and these services are publicly marketed. It would be possible to identify these networks from their configs by fingerprinting the configurations needed to implement the unique service.

D. Attacks Based on Network Composition

Many networks consist of devices of different types running different versions of the operating system software. The distribution of device makers, device types and software versions could serve as a fingerprint, and device vendors might have sufficient data to de-anonymize the configs sets of its customers.

E. Attacks Based on Network Topology and Peering

Although we independently hash the AS numbers that identify the peers of an anonymized network, anonymized configs accurately represent the number of routers at which the anonymized network peers with other networks, and the number of peering sessions that terminate on each of those routers. This peering structure could serve as one form of fingerprint that could be checked against maps made using the RocketFuel techniques [3]. However, there are many side-door peerings between real backbone networks that RocketFuel and RouteViews do not see, so it is an open experimental question for future work to determine if there is enough entropy in the peering structures to make them useful as fingerprints.

It seems likely that peering structure can be used to fingerprint backbone networks, but not edge networks. This is because edge networks have fewer points of attachment to the backbone and because they do not generally provide transit so their peering structure cannot be measured via RocketFuel. Also, edge networks often have firewalls that drop unsolicited probes, such as traceroutes, and so their internal topology cannot be measured from outside.

Summarizing these vulnerabilities, until such time as the actual risks of fingerprinting attacks can be established, we cannot conclude that our method securely anonymizes backbone networks. However, for the many networks which cannot be externally fingerprinted, either because they use firewalls or are not reachable over the public Internet, this method appears reasonably secure against external attackers. The remaining concern is that an *insider attack*, where the probing/fingerprinting is launched from a host in the target network, could potentially succeed. However, 10 of 31 networks we examined use internal compartmentalization that would also defeat insider attacks by preventing a user at a single host inside the network from computing a fingerprint of the entire network. For example, some networks use NATs to divide up the network into smaller pieces, some use routing policy to prevent reachability between portions of the network, and others drop traceroutes and other probe traffic.

[5]This has worked well on the configs we have tried it on, although it would work poorly for Genuity customers as Genuity's AS number (AS 1) will appear in many unrelated config lines.

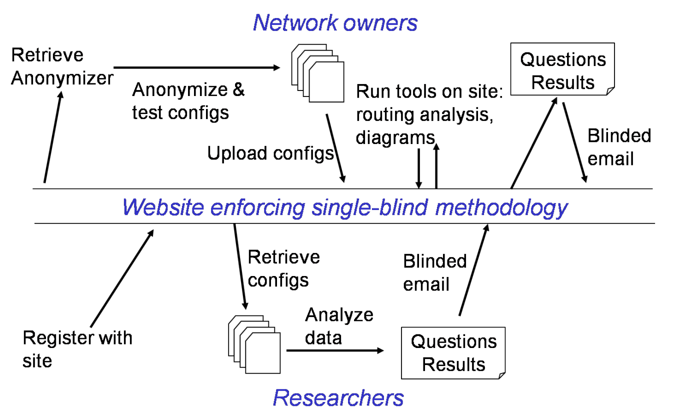


Fig. 4. Design for a clearinghouse that uses the anonymization techniques from this paper to enable researchers to access configurations via a single-blind methodology.

VIII. Towards a Clearinghouse of Configuration Data

The motivation for our work is to create a means by which network owners will feel comfortable making their configuration data available to the research community.

Using the ability to anonymize router configuration files, we propose it is time to establish a *single-blind methodology* for working with private network data through a website portal. By single-blined, we mean that the portal website serves to keep the identity of network owners hidden from researchers accessing the configuration files, while still allowing communication between researchers and network owners. Since network owners are the parties requiring the most incentive, we propose giving them the option of restricting which researchers can access their configurations. We suggest the design shown in Figure 4 that, in effect, automates the methodology we have successfully used in gaining access to enterprise network configurations [1].

Network owners begin participating by downloading the configuration anonymization tools from the portal (via third-party web traffic anonymizers if desired), and uploading their anonymized configurations after taking whatever additional steps they felt necessary to verify the anonymization. As the research community develops useful analysis tools, these tools will be added to the website so that they can be easily run on uploaded configuration sets.

The identities of Researchers wishing to access configuration will be verified in the same fashion as that used by PlanetLab [11] and Emulab [12]. Researchers with accounts on the portal will request access to configuration sets and be able to download them once given permission by the network owners — this allows the owners to retain a measure of control over the scope of distribution of their configs. As researchers analyze the configurations, they are likely to have questions about the configurations. The website will enable blinded communication between the anonymous network owners and the researchers and third-party web traffic/email anonymizers. In our experience, our questions were generally of the form "is this the intended behavior, or is this a bug in the network design?" so hopefully communication from the researchers will be valued and answered by the network owners, although this cannot be assured.

The motivation for researchers to use the site is clear - access to previously unavailable configuration files. Motivating network owners to upload configurations will be harder. We hope that by making interesting tools available on the website and offering the networks the potential of "free" consulting from experienced researchers we can create sufficient value that network owners will upload their configurations. To bootstrap the system, we have found some network owners who would like to share their configurations to "aid the greater good", but need help preparing them. An easily accessible web portal might be sufficient to get these owners to share.

We recognize that ultimately trust will be incremental. Some researchers may be given access to raw configs, others to anonymized configs via a single-blind methodology, and others only to higher-level abstractions of the data. Part of our current research aims to define such higher-level abstractions, and we rely heavily on data extracted from anonymized configs to ensure our abstractions can express the diversity found in real networks.

IX. Related Work

There is a large body of work on anonymizing packet traces [13], [14], [15], however ours is the first we know of to anonymize router configuration files. Both domains share the problem of anonymizing IP addresses, but anonymization of router configuration files must also anonymize the text of the files while still preserving the structure of relationships among the entities in the files. Backstrom and colleagues [16] consider the problem of anonymizing social networks while maintaining their structure, and argue anonymization is easily broken. However, their attacks require knowledge of parts of the graph. As we describe in Section VII, it will probably be impossible to anonymize backbone networks because too much of their external peering structure is publicly known. This should not be a problem for enterprise networks.

Our hope that the address space fingerprint of a network will be insufficient to uniquely identify any single network once there is a large enough body of anonymized networks is an example of k-anonymity as defined by Sweeney [17], and the techniques proposed by Coull and colleagues [18] could be used to measure the entropy of network fingerprints. Until such time as a large body of configuration sets exists, the anonymity of the configurations will rely on the difficulties of computing a fingerprint for enterprise networks, which are generally compartmentalized and not externally probe-able.

As an alternative to the anonymization and distribution of configuration files, Dwork proposes an interactive system whereby researchers pose questions to an analysis system that holds the configurations files [19]. While provably-strong privacy guarantees can be made about interactive systems, it is not clear that the types of questions of interest to networking researchers can be expressed in these systems.

X. Summary and Future Work

In this paper we make two contributions.

First, we have formulated the key issues of the configuration anonymization problem, including the requirements for an acceptable anonymization method, major areas of challenges, a methodology for validating anonymized data, and potential security vulnerabilities. The formulation exposes essential trade-offs between anonymization and information preservation, and can serve as a basis for further discussions by the research community leading to refined solutions.

Second, we provide a working solution for configuration anonymization that meets the formulated requirements. It has been validated with a major carrier, earning unprivileged researchers access to the configuration files for dozens of networks.

While both technical and organizational challenges remain to be overcome in the creation of network configuration data sets accessible to the research community, we are excited by the new areas of research such data sets could open up — areas with impacts in both networking research and network operations. Our work on the anonymization of configurations is intended as a first step in generating momentum towards this goal.

REFERENCES

[1] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg, "Routing design in operational networks: A look from the inside," in *Proc. ACM SIGCOMM*, August 2004.

[2] D. A. Maltz and J. Zhan, "Source code for router configuration anonymizer." https://sourceforge.net/projects/config-anon/.

[3] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with RocketFuel," in *Proc. ACM SIGCOMM*, August 2002.

[4] D. Eastlake, 3rd and P. Jones, *RFC 3174 - US Secure Hash Algorithm 1 (SHA1)*, 2001. Available from http://www.ietf.org/.

[5] G. Minshall, "tcpdpriv - remove private information from a tcpdump -w file." Software distribution available from http://ita.ee.lbl.gov/-html/contrib/tcpdpriv.html, 1997.

[6] J. Xu, J. Fan, M. Ammar, and S. B. Moon, "Prefix preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme," in *Proc. International Conference on Network Protocols*, October 2002.

[7] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets." RFC 1918 (Best Current Practice), February 1996.

[8] J. C. Martin, *Introduction to Languages and the Theory of Computation*. McGraw-Hill, 1991.

[9] T. Ylonen, "Thoughts on how to mount an attack on tcpdpriv's "-a50" option...." Web White Paper available from http://ita.ee.lbl.gov/-html/contrib/attack50/attack50.html.

[10] B. Ribeiro, W. Chen, G. Miklau, and D. Towsley, "Analyzing privacy in enterprise packet trace anonymization," in *NDSS*, 2008.

[11] L. L. Peterson, A. C. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences building PlanetLab," in *OSDI*, pp. 351–366, 2006.

[12] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *OSDI*, pp. 255–270, December 2002.

[13] R. Pang and V. Paxson., "A high-level programming environment for packet trace anonymization and transformation," in *SIGCOMM*, 2003.

[14] M. Peuhkuri, "A method to compress and anonymize packet traces," in *ACM Internet Measurement Workshop*, 2001.

[15] J. Xu, J. Fan, M. Amar, and S. B. Moon, "On the design and performance of prefix-preserving IP traffic trace anonymization," in *Proc. Internet Measurement Workshop*, 2001.

[16] L. Backstrom, C. Dwork, and J. M. Kleinberg, "Wherefore art thou r3579x? anonymized social networks, hidden patterns, and structural steganography," in *WWW*, 2007.

[17] L. Sweeney, "k-anonymity: a model for protecting privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems,*, vol. 10, no. 5, pp. 557–570, 2002.

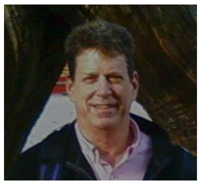
[18] S. Coull, C. Wright, A. Keromytis, F. Monrose, and M. Reiter, "Taming the devil: Techniques for evaluating anonymized network data," in *NDSS*, 2008.

[19] C. Dwork, "Differential privacy," in *33rd International Colloquium on Automata, Languages and Programming-ICALP*, 2006.

David A. Maltz is in the Networking Research Group of Microsoft Research where he designs new architectures for data center and enterprise networks and investigates techniques for managing IT infrastructure. He received his Ph.D. from CMU in 2001 and his S.B. from MIT in 1993. He founded a traffic engineering startup and was a post-doc at CMU.

Jibin Zhan is the Manager for Data Management and Analysis at Conviva Networks, which is redefining the Internet video experience. He has worked as a researcher and program manager at Carnegie Mellon University and Turin Networks.

Gísli Hjálmtýsson is CEO and co-founder of Thule Investments ehf. Prior to that he was Dean of Computer Science and Engineering at Reykjavik University. Dr. Hjálmtýsson joined AT&T Research at Bell Laboratories in 1995, later AT&T Research at AT&T Shannon Laboratories. Dr. Hjálmtýsson has a Ph.D. in Computer Science from University of California at Santa Barbara. He has held positions on the faculty of Columbia University, University of Iceland and at Reykjavk University. In 1993 Dr. Hjálmtýsson was a visiting research scientist at Telecom Research Laboratories, in Melbourne Australia for six months. He has been a co-recipient on significant grants from EC, NSF, DARPA, with leading researchers from MIT, Dartmouth College, Carnegie Mellon University, Princeton and more.



Albert Greenberg is an ACM Fellow, and a Principal Researcher at Microsoft, which he joined in Jan 2007. At Microsoft, he is working on data center networking, enterprise network management, and monitoring. From 1983 to 2007, Albert worked at AT&T Bell Labs, where he was named an AT&T Fellow and was awarded AT&T's Science and Technology Medal., and where he worked on packet and flow measurement and analysis, traffic matrix inference, anomaly detection, configuration management, IP/MPLS control plane monitoring, MPLS/GMPLS control and management, IP traffic and network engineering, IP fault management and troubleshooting, new route control architectures, database and systems applications, network security, scheduling, wireless and satellite networks, massively parallel computation, and parallel simulation.

Jennifer Rexford (S'89, M'96, SM'01) is a Professor in the Computer Science department at Princeton University. From 1996-2004, she was a member of the Network Management and Performance department at AT &T Labs–Research. Jennifer is co-author of *Web Protocols and Practice* (Addison-Wesley, May 2001). Jennifer served as the chair of ACM SIGCOMM from 2003 to 2007, and serves on the CRA Board of Directors. She received her BSE degree in electrical engineering from Princeton University in 1991, and her MSE and PhD degrees in computer science and electrical engineering from the University of Michigan in 1993 and 1996, respectively. She was the winner of ACM's Grace Murray Hopper Award for outstanding young computer professional of the year for 2004.

Geoffrey G. Xie is a Professor in the Computer Science department at U.S. Naval Postgraduate School. He received his BS degree in computer science from Fudan University, China, his MS in computer science and his MA in mathematics from Bowling Green State University, Ohio, and his PhD in computer sciences from the University of Texas, Austin. His current research interests include clean slate design of IP control plane, static analysis of network configuration, routing glue logic, underwater acoustic networks, and abstraction-driven design and analysis of enterprise networks.

Hui Zhang is the co-founder and Chief Scientist of Conviva Inc. and a Professor in the Computer Science Department at the School of Computer Science at Carnegie Mellon University. He won the National Science Foundation CAREER Award in 1996, the Alfred Sloan Fellowship in 2000, and was elected a fellow of ACM in 2006. During 2000 - 2003, he served as the Chief Technical Officer of Turin Networks.