Structure Preserving Anonymization of Router Configuration Data[*]

David A. Maltz, Jibin Zhan, Geoffrey Xie, Hui Zhang
{dmaltz,jibin,geoffxie,hzhang}@cs.cmu.edu
Carnegie Mellon University

Gísli Hjálmtýsson[†] Albert Greenberg, Jennifer Rexford
{gisli,albert,jrex}@research.att.com
AT&T Labs–Research

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, that mean 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 more than 7600 routers in 31 networks. Through example analysis, we demonstrate that the anonymized data retains the key properties of the network design. We believe that applying our single-blind methodology to a large number of production networks from different sources would be of tremendous value to both the research and operations communities.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]

[*]This research was sponsored by the NSF under ITR awards ANI-0085920, ANI-0331653, and ANI-0114014. 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 AT&T, NSF, or the U.S. government.
[†]Also at Reykjavík University

General Terms

Measurement, Design, Management

Keywords

Data anonymization, router configuration, security

1. INTRODUCTION

By far the best source of design information available today for an IP network is the running configuration files associated with its routers.[1] Each of these files, known as "configs", contains the complete set of commands used to define the behavior of a single router. Access to the router configuration files of production networks would bring tremendous benefits to a wide group of networking researchers. For example, an accurate 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 the exact same reasons that make them valuable for research. They reveal internal details of the network design, and expose business secrets such the owner's organizational structure and clientele. Further, they expose potentially embarrassing configuration mistakes and security vulnerabilities that could be remotely exploited. Only if the link between the configuration files and the identity of the network owner is severed can it be guaranteed that any information learned from the configurations cannot be exploited against the owner. Therefore, before public access to the configs of production networks becomes feasible, a method must be invented to anonymize the data. However, the anonymization must preserve relationships within the information to retain the data's value for networking researchers.

In this paper, we present a detailed formulation of this configuration anonymization problem. We qualify the two

[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.

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 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 more than 7655 routers in 31 backbone and enterprise networks.

2. 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 ASs (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

```
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
```

Figure 1: Excerpts of a router configuration file.

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.

3. CHALLENGES

In this section, we 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.

3.1 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 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.

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

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 4.

3.2 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).

Public AS Numbers must be hashed: 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.

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

4. 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.

4.1 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 [2]: this anonymizes the names of class-maps, route-maps, and any other strings that could hold privileged information. Simple integers are generally not anonymized.

4.2 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 discovered a set of 28 rules[4] 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.

4.3 Anonymizing IP Addresses

Two of the best prefix preserving IP address anonymization schemes are due to Xu [4] and Minshall[5]. Xu's has the property that very little state must be shared to consistently

[4]More details available in the technical report [3].

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 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`. We configured it to be class-preserving, and modified it so all "special" IP addresses (e.g., netmasks, multicast) are passed through unchanged. Doing so 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 have proven maintains the structure-preserving property of the algorithm.

4.4 Anonymizing AS 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,[5] 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 6.

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.

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

We anonymize regular expressions involving digit wildcards and ranges by leveraging automata theory [6]. 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.

4.5 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.

5. 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. The anonymization process will independently anonymize the AS numbers and the IP prefixes, however, allowing a reader to determine that routes to external networks are being dropped via two different mechanisms, but not that the mechanisms both target the same AS. The anonymizer today supports wide classes of useful analysis. If the anonymizer is provided with the "well known" external information on which the implicit relationship is based, it can be extended to preserve these relationships as well.

6. 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.

6.1 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 iteration closes quickly, requiring fewer than 5 iterations 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.[6]

6.2 Attacks on the IP Address Anonymization

Hypothetical attacks have been proposed [7] 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 via counting up how many subnets of different sizes (/30s, /29s, /28s, etc.) appear in the anonymized configs. To determine the identity of the physical network that the configs belong to, he could then send probe packets into candidate physical networks attempting to measure how many subnets of different sizes each candidate contains from the ICMP Reply or backscatter 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 physical 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. The remaining question that we will experimentally evaluate in future work 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.

6.3 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 [8]. 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

[6]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.

useful as fingerprints. It seems likely that peering structure can be used to fingerprint backbone networks, but not edge networks — both because they 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 the fingerprinting attacks like the ones mentioned above 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. 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.

7. 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.

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 plan to establish a *single-blind methodology* for working with private network data through a website portal. Network owners could download the configuration anonymization tools from the portal via third-party web traffic anonymizers, and upload their anonymized configurations after taking whatever additional steps they felt necessary to verify the anonymization. Researchers with accounts on the portal could then be given access to the data, communicating comments to the anonymous network owners through a blinding function of the portal and the third-party web traffic/email anonymizers.

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.

8. REFERENCES

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

[3] D. A. Maltz, J. Zhan, G. Xie, H. Zhang, G. Hjalmtysson, A. Greenberg, and J. Rexford, "Structure preserving anonymization of router configuration data," Tech. Rep. CMU-CS-04-149, Carnegie Mellon University, 2004.

[4] 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.

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

[7] 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.

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