Provable Network Activity for Protecting Users Against False Accusation

Panagiotis Papadopoulos[1(✉)], Elias Athanasopoulos[2], Eleni Kosta[4], George Siganos[3], Angelos D. Keromytis[5], and Evangelos P. Markatos[1]

[1] FORTH - Institute of Computer Science, Heraklion, Greece
panpap@ics.forth.gr
[2] Vrije Universiteit Amsterdam, Amsterdam, Netherlands
[3] Qatar Computing Research Institute, HBKU, Doha, Qatar
[4] Tilburg Law School, Tilburg University, Tilburg, Netherlands
[5] Columbia University, New York, USA

Abstract. With the proliferation of the World Wide Web, data traces that correspond to users' network activity can be collected by several Internet actors, including (i) web sites, (ii) smartphone apps, and even (iii) Internet Service Providers. Given that the collection and storage of these data are beyond the control of the end user, these data traces can be easily manipulated, if not, tampered with. The result of such manipulated digital traces can be severe: Innocent users can be shamed or even wrongfully accused of carrying out illegal transactions.

To eliminate these potential accusations on innocent users, we introduce *Provable Network Activity (PNA)*: a framework with which the ISPs can give the end users control of their stored traces. The framework guarantees that the information collected for the end users is accurate and will remain accurate for as long as it is stored. Our implementation and preliminary evaluation suggest that PNA is fast, easy to deploy, and introduces zero network latency to the end clients.

1 Introduction

Over the past few years the degree to which computers are involved in evidence collection for crime and security investigations has profoundly changed. A couple of decades ago there were few cases that could be associated with cyber crime. Nowadays, there is hardly ever a case that does not involve a computer or a smartphone. Data from such devices are so important that even the FBI [5] or the government's prosecutors [8] want to get access to them almost at any cost.

Unfortunately, despite their importance, data collected by our digital infrastructure today are not necessarily trustworthy. Indeed, malicious attackers or even rogue insiders may tamper with the process of collecting and/or storing data and may add *fake* data about individual users. Such fake data may later be used to ridicule and slander users, or worse, accuse them of questionable or

© IFIP International Federation for Information Processing 2016
Published by Springer International Publishing Switzerland 2016. All Rights Reserved
S. Foresti and J. Lopez (Eds.): WISTP 2016, LNCS 9895, pp. 254–261, 2016.
DOI: 10.1007/978-3-319-45931-8_17

even illegal activities. Although fake digital evidence usually does not stand in a court of law, it is often enough to send people to jail [7].

In this paper we propose that data collection processes should actively involve the users concerned. We advocate that the users should not only give their consent in all data collections, but should also make sure that *the collected data are accurate* and will always *remain accurate* for as long as they are stored. To do so, we involve the user in every step of the process: (i) users collect, (ii) users double check, (iii) users sign, and finally, (iv) users encrypt the collected data.

Our proposal is Provable Network Activity (PNA): an architecture that enables both clients and ISPs to collaborate in the data collection process and make sure that the collected data are accurate. PNA works as follows:

- A client who wants to gain control of her traffic and data, and ensure their authenticity can simply register to the service by providing her ISP with her public key. Then, she installs a software responsible for signing all outgoing traffic, before it is sent to the ISP, with her private key[1].
- If the ISP receives traffic from the client *without* receiving the corresponding signature, the traffic is blocked[2].
- If the ISP receives a valid signature, it keeps a record of the access in its log.

2 Threat Model

In this paper we focus on protecting innocent users from false accusations based on fake network logs. We assume that the user's access logs have been collected by a local ISP (or similar service). We also assume that, at some point in time (either during the collection process or later), the logs were tampered with, maybe by intruders, or even by insiders.

More precisely, the setup is composed by three different entities: (i) the user U, (ii) the last-mile ISP I, and (iii) an web site s hosting offensive or even illegal content. We make the following assumptions:

- user U never accessed web site s (via ISP I).
- although ISP I is considered generally trusted, its data logs may be tampered with to show that U has accessed s [10].

3 Provable Network Activity

Traditional data collection has been done by the ISPs without consulting the users at all. That is, ISPs collected data about the users *without the users being able to verify that the collected data are correct* and that the data will remain correct for as long as they are stored.

We advocate that this approach should change. Data should not be collected without the user being able to verify that the data are (and will remain) correct.

[1] The exact details of the key management process is beyond the scope of this paper.
[2] Note that this service works on an *opt-in* basis.

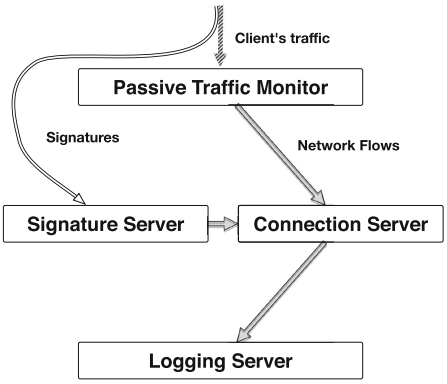

Fig. 1. Overview of the PNA components and their interconnections on the ISP side.

To bring the user back in the game, PNA proposes that the collected data (i) should be reviewed by a user's application (ii) should be signed by the user, and (iii) should be encrypted by the user. Any attempts to tamper with the data will stumble upon the signing and encryption process which becomes impossible (or obviously detectable) without the user's cryptographic keys.

3.1 Client Side

Key Management. The client is responsible for generating a public-private key pair, and is responsible for the security of her private key[3]. The client at any time can replace her public-private key pair, but it is her responsibility to store the old private key and update the public key provided to the provider.

Daemon. A daemon runs in the client's computer and passively monitors all outgoing traffic. For each outgoing flow a signature is computed. The signature is computed on the tuple (destination IP address, port number, protocol, timestamp) by first using SHA and then encrypting with the user's private key. The daemon then sends the tuple plus the signature of the tuple to the ISP's server[4]. This signature is a proof generated by the client that the client *did make* this network connection. If the ISP does not receive such a proof, it will not allow the connection to be established.

3.2 ISP Side

For each client connection (e.g. network flow), the ISP expects to receive a signature from the client. If the signature is received, the connection is allowed

[3] For simplicity, we describe our approach as if the user has one device, but it can be easily generalized to include several devices and users, as well.

[4] Note, that we do not use the source IP/port, since usually NAT/Proxies overwrite the source part [4].

to proceed and a log record is generated. If the signature is not received, or is not valid, the connection is not allowed to proceed.

The infrastructure on the ISP side consists of the following entities: (i) a *Passive Traffic Monitor*, (ii) a *Signature Server* that is interacting with the client-side, (iii) a *Connection Server* that maintains the connections and enforces policy, and (iv) a *Logging Server* that provides persistence in a privacy-preserving way.

- **Passive Traffic Monitor.** The ISP passively monitors all traffic to identify new flows. For each new flow identified, the monitor creates a new record to characterize the flow, and sends the record to the Connection Server that handles the policy.
- **Signature Server.** The Signature Server acts as a gateway to the system. It receives signatures from the client side and forwards them to the Connection Server.
- **Connection Server.** The Connection Server collects information (i) from the Passive Traffic Monitor and (ii) from the Signature Server, and makes sure that for each flow received by the Passive Traffic Monitor a corresponding signature is received from the Signature Server. If a signature is not received (or is not valid) for a given flow, the flow is blocked/terminated.
- **Logging Server.** The Logging Server is responsible for storing all signatures for each subscriber.

3.3 Privacy-Preserving Logging

The approach described so far keeps a record only of the IP addresses really accessed by the client:

- If an attacker tries to add a record to the log, this record will just not have the client's signature, and therefore it will be easily singled out as fake.
- If the client tries to access an IP address without providing a signed record for this access, the ISP will clock/reset the access.

Therefore, the log will be an always accurate record of the user's activity. Unfortunately, keeping the log as described so far may seriously violate the user's privacy. Indeed, if the log is leaked, then all the user's accesses will be easily visible. Encrypting the log with an ISP's key does not help either. A malicious attacker may accuse the client of illegal activity just to force the client to reveal the entire log in order to prove that the illegal activity is not there. Such an approach will help the user demonstrate her innocence, at the price of her privacy. It seems therefore that there is a dilemma the user needs to choose from: *either* demonstrate her innocence *or* protect her privacy. In this paper we believe that this is a *false dilemma*. We believe we can do both: i.e. *both* demonstrate the client's innocence *and* protect the client's privacy at the same time. To do so, instead of keeping each signed record, we keep a *hash* of it. We use bcrypt [9] in order to hash all network signatures.

4 Implementation

4.1 PNA Prototype

We implemented PNA in Linux using `libpcap` a building block for capturing network traffic, and OpenSSL a module that provides all cryptographic operations. The implementation was rather straightforward: all software is written in C++ totalling around 2,000 lines of code.

Client-Side Implementation. A Linux-based host runs the code for the client: a DSL subscriber. This host runs the daemon software as we described in Sect. 3.1. The daemon initially connects to a predefined port of a Linux-based server, which is run by the ISP. In parallel, the daemon monitors all TCP/UDP traffic and captures (i) all TCP packets having the `TCP-SYN` flag on and (ii) all UDP packets. For each TCP-SYN packet and for each UDP packet that starts a new flow, the client daemon generates a signature of the packet with the user's private key and sends it to the server using the established connection.

ISP-side Implementation. At ISP side, the server runs the passive traffic monitor, as we described in Sect. 3.2, for capturing all incoming traffic and infer (i) new TCP flows (by inspecting the `TCP-SYN` flag), and (ii) new UDP flows. The monitor uses Redis, an open source in-memory key-value store for maintaining all identified UDP flows. In addition, it also runs the signature module, which listens to a predefined port for incoming signatures by the client. The server verifies each incoming signature, by decrypting it using the user's public key. If the signature is valid, it is forwarded to the logging server and it is stored using bcrypt [9]. Otherwise, (i) if the signature is not valid, or (ii) if the signature is not sent by the client, or (iii) if the signature arrives too late, the connection is terminated. TCP/IP connections area terminated by an explicit `TCP-RST` to the user's host. For UDP connections, the flow is just blocked.

For the asymmetric encryption/decryption, the RSA algorithm is used with `PKCS1` padding as provided by OpenSSL). The length of the keys is 2048 bits. For hashing before encrypting we use `SHA512` as provided by OpenSSL. For bcrypt we use the implementation as provided by the original paper [9].

5 Evaluation

5.1 Client Side

The cryptographic overhead is the only *computational* overhead we impose on the client side. More specifically, the client needs to sign every new flow and send this signature to the ISP in time no more than T. To quantify the latency imposed by this required cryptographic operations, we perform 10,000 private-key encryptions for a testing tuple and we measure the average latency. In this experiment we use OpenSSL's RSA implementation with keys of length of 2048

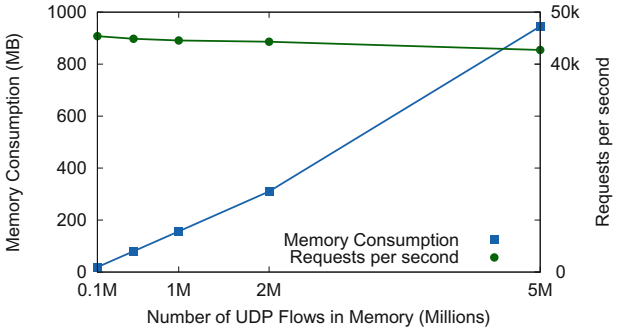

Fig. 2. Passive monitor performance. We see that even for 5 million flows, the server requires less that 1 GB of memory.

bits in two devices: a Linux desktop and a Mac OS laptop. In both cases, the overhead per encryption was less than 10 ms. This means that a typical computer can handle more than a hundred of new flows (i.e. new signatures) per second.

5.2 ISP Side

Passive Traffic Monitor (PTM). As we described in previous section, PTM is responsible for identifying new network flows. While this operation is straightforward when the flows are TCP, in the case of UDP flows it is not that trivial. For each UDP packet, a query needs to be sent to the key-value store to check if this UDP flow has been encountered in the past. Given this complexity, we focus on the UDP flows identification, which evidently is the most expensive part of this component.

To measure the performance of PTM, we simulate traffic sent from client to ISP, containing various number UDP flows (ranging from 100 K to 5 M). The UDP packets of these flows are distributed using a long-tail distribution (80 % of the packets are distributed to 20 % of the flows). In Fig. 2, we can see the results, where it seems that our server is able to process more than 40 K reqs/sec.

In the same experiment, we measure the memory requirements of our system. In Fig. 2, we can see that, as expected, the memory consumption demands grow linearly with the number of UDP flows. Indeed, we see that even if we have as many as 5M UDP flows we probably need less than 1 GB of memory to store all data. Of course, this is a trivial amount of memory for contemporary servers.

Logging Server. This is the component responsible for securely storing the signatures in a privacy-preserving fashion. The main bottleneck in the logging server is the computation of the hash key using bcrypt. To measure this computation overhead we stress a desktop equipped with one Quad-core Intel processor at 2.4G to identify how many bcrypt operations per second can achieve and in Fig. 3 we plot the results. As we see, we can have more than 2000 bcrypt operations per second, and consequently, we can process more than 2000 new flows

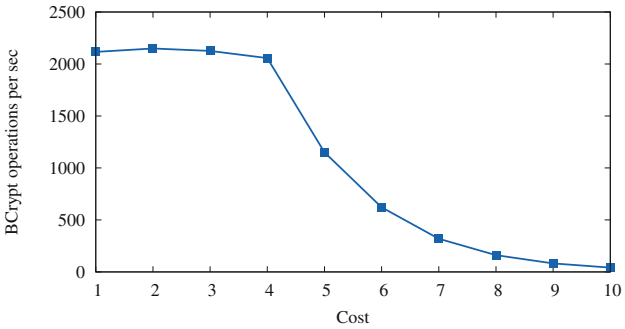

Fig. 3. Bcrypt performance as a function of the value of *cost*. Up to cost = 4, the single CPU can perform more than 2,000 operations per second.

per second. Additionally, recall that it is an off-line operation, and thus does not increase the typical packet routing operation of the ISP.

It is worth mentioning at this point, that bcrypt wastes computational resources on purpose for prohibiting massive in-parallel cracking of the generated hash products and resist dictionary attacks. To achieve that, bcrypt uses *cost* parameter. In our experiment, we explore how this parameter affects the overall calculation performance by testing various values of *cost*. As we can see, up to cost = 4, the single CPU can still perform more than 2,000 operations per second. As a result, even when using a cost value able to sky-rocket the amount of work required by an attacker to reveal the network logs, it is not capable of degrading the system's performance.

6 Related Work

Pretty Good Packet Authentication (PGPA) [6]) is a system that hashes all user's outgoing traffic and stores it to a device. The device must be placed between client and ISP: either towards the user's side, i.e. in her household, or it can be hosted by the ISP. Packet Attestation (PA) [2] suggests installing special monitors, independent of the current Internet's routing infrastructure, in every Autonomous System (AS). These monitors hash and collect all user traffic, and they can attest if the traffic is authentic under a particular case, where a user is accused for receiving offensive traffic. Clue [1] attempts to bring the notion of physical evidence, such as DNA, in the digital world. More precisely, Clue uses Group Signatures [3] for delivering a scheme, where each Internet packet can be attributed to its original source. All these systems propose major reconstruction of fundamental parts of today's Internet. We agree with a large fraction of these proposals and we believe that a clean-slate design will terminate a rich collection of problems that we exhibit today. However, deployment is not always trivial. Changing fundamental Internet-core concepts, such as routing or addressing, is really hard.

7 Conclusion

In this paper we designed, implemented, and evaluated PNA, a framework which gives end users control of their own traces as they recorded by their last-mile ISP. PNA empowers users with the ability to double check the data collected about them and to make sure that these collected data are correct and *will remain correct for as long as they are stored* in the ISP's logs. To do so, users explicitly sign each network flow. The ISP makes sure that each network flow is accompanied by the associated signature. If a signature is not received, the ISP blocks the flow. To ensure the correctness of the log, the ISP stores *both* the network flow records *end* their associated signatures. In this way, tampering with the stored data is extremely difficult: even if attackers manage to add records to the log, they will not be able to add the corresponding signatures. Our preliminary implementation and evaluation shows that our system does not impose any extra latency to the end user and has low computational requirements.

Acknowledgements. This work was supported in part by the project GCC, funded by the Prevention of and Fight against Crime Programme of the European Commission – Directorate-General Home Affairs under Grant Agreement HOME/2011/ISEC/AG/INT/4000002166. This project has received funding from the European Unions Horizon 2020 research and innovation programme under the Marie Skodowska-Curie grant agreement No 316808. This publication reflects the views only of the authors, and the European Commission cannot be held responsible for any use which may be made of the information contained therein.

References

1. Afanasyev, M., Kohno, T., Ma, J., Murphy, N., Savage, S., Snoeren, A.C., Voelker, G.M.: Privacy-preserving network forensics. Commun. ACM **54**, 78–87 (2011)
2. Haeberlen, A., Fonseca, P., Rodrigues, R., Druschel, P.: Fighting cybercrime with packet attestation (2011). http://repository.upenn.edu/cgi/viewcontent.cgi?article=1652&context=cis_papers
3. Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991)
4. Clayton, R.: Mobile internet access data retention (not!) (2012). http://www.lightbluetouchpaper.org/2010/01/14/mobile-internet-access-data-retention-not/
5. Digital trends staff: Apple vs. the FBI: a complete timeline of the war over tech encryption (2016). http://www.digitaltrends.com/mobile/apple-encryption-court-order-news/
6. Haeberlen, A., Rodrigues, R., Gummadi, K., Druschel, P.: Pretty good packet authentication. In: Proceedings of the Fourth Workshop HotDep 2008 (2008)
7. India News: Techie jailed due to Airtel mistake (2012). http://twocircles.net/node/25440
8. Kravets, D.: Twitter reluctantly coughs up occupy protesters data (2012). https://www.wired.com/2012/09/twitter-occupy-data/
9. Provos, N., Mazières, D.: A future-adaptive password scheme. In: ATEC 1999. http://dl.acm.org/citation.cfm?id=1268708.1268740
10. Stolfo, S.J., Bellovin, S.M., Keromytis, A.D., Hershkop, S., Smith, S.W., Sinclair, S. (eds.): Insider Attack and Cyber Security - Beyond the Hacker. Advances in Information Security, vol. 39. Springer, US (2008)