Monitoring message size to break privacy
Current issues and proposed solutions

Alfredo Pironti
INRIA Paris-Rocquencourt, Paris, France
alfredo.pironti@inria.fr

January 15, 2014

Abstract

One of the Internet traffic features that can be easily collected by passive pervasive monitoring is the size of the exchanged messages, or the total bandwidth used by a conversation. Several works have showed that careful analysis of this data can break users' expected privacy, even for encrypted traffic. Despite this, little has been done in practice to hide message sizes, perhaps because deemed too inefficient or not a realistic threat.

In this short paper, we contextualize message size analysis in the wider pervasive monitoring scenario, which encompasses other powerful analysis techniques, and we re-state the severity of the privacy breach that message size analysis constitutes. We finally discuss proposals to fix this issue, considering practical aspects such as required developer awareness, ease of deployment, efficiency, and interaction with other countermeasures.

1 Introduction

As witnessed by recent awareness, pervasive monitoring is a reality, and it can be used to break expected users' privacy and confidentiality. One specific feature of pervasive monitoring is that it analyzes communication meta-data, rather than the communication payload itself, so that it can be applied to break encrypted communications as well. Typical targets of pervasive monitoring are popular application protocols such as HTTP or SMTP, their underlying security protocols such as TLS or SSH, as well as the base communication protocols such as TCP and IP.

Each protocol in the stack leaks some meta-data about the ongoing communication; even worse, the design of many of them does not even include security nor privacy aspects. Experience with the TLS protocol has showed that basic security properties can be plugged into existing applications almost transparently, with the notable exceptions of key management and computational overhead. However, a challenge remains in the protection of communication meta-data, and it is unclear yet whether this can be obtained purely as a pluggable aspect of the communication, or whether application involvement is necessary.

Furthermore, one can probably never achieve full meta-data indistinguishability without making the Internet unsustainably inefficient. Hence, it is important to identify a trade-off between meta-data obfuscation and efficiency, so that pervasive monitoring is thwarted with acceptable costs.

1.1 Brief discussion of meta-data availability

In a typical Internet communication, the source, destination and application protocols are extremely easy to collect for a passive attacker. In itself, this constitutes already a very valuable source of information, and probably constitutes one of the highest priorities to address. Attempts to hide this information include the Tor project. Notably, Tor requires specific client and routing software, but interoperates transparently with existing servers. While this is effective to thwart most basic attacks, careful traffic analysis can still identify target websites at the Tor-enabled communication source [9], and the plain communication between the Tor exit node and the server remains the weakest point.

Other meta-data are provided by the so-called side channels, which include at least timing, pattern and size of exchanged messages, as well as power consumption and acoustic cryptanalysis. Often these side channels are correlated, for example padding messages to hide their size may expose a new timing channel, if padding is processed quicker than application data.

According to the specific application, one side channel may be more exploitable than others, but in general trying to normalize different known side channels seems a reasonable target to achieve, to minimize the attack surface. Unfortunately, it turns out that some side channels are particularly tricky to eliminate. For example, execution time usually depends on specific hardware architectures, which may autonomously make unpredictable random choices to optimize execution speed, defeating cleverly designed constant time programming patterns.

In some other cases, some application cooperation is required in order to mitigate the side channel. For example, normalizing message patterns require all communicating parties (say, client and server) to adopt the same policy. In some cases, intermediate layers can help (such as Tor on the client side by sending messages of the same size at regular intervals), but with additional performance costs.

To sum up, given the current state of the art against pervasive monitoring, it seems that basic communication features such as endpoint identities are the most effective to exploit, and thus require immediate action. Nevertheless, once those are fixed, a wealth of communication meta-data remains available, that can be exploited with moderate resources and automated analysis. In the following, message size is discussed more deeply, along with some of its possible interactions with other side-channel communication features.

2 Exploiting message size analysis

In the following, we will assume that all communication payload is end-to-end encrypted (typically by using TLS), to provide its confidentiality and integrity. Nevertheless, we will show to what extent message size and pattern analysis can reveal some of the encrypted information.

In [2, 8], the size and number of objects returned by a web server following a client HTTP request are used to identify the client's requested URL. Hence, even if the request was sent over TLS, its content would not be protected against this kind of analysis. Even worse, when eavesdropping is performed at the server side, not even clients using Tor-like solutions can avoid profiling the served web pages.

In [1], the analysis of message size and direction is used to reveal users' browsing habits. For example, user's activity on an online health website is reconstructed by observing the encrypted communication pattern: in practice, the passive attacker gets to know whether the user searches for specific illnesses or doctors, or if she is adding a health record to her account. Similarly, with search engines, the encrypted searched keywords are revealed when the auto-completion feature is in place, by observing the size of the returned auto-completion lists, which is a function of the input string.

Bulletin boards are targeted in [3]. By correlating the size of encrypted documents uploaded to and downloaded from a bulletin board, an eavesdropper can track the flow of such documents. Additionally, if the bulletin board is publicly available, the attacker can further correlate the document flow with its content. A related attack tracks documents sent via mailing lists.

On this line of work, in [7] personal data shared between public social networks and private services such as email is correlated in order to de-anonymize the users of the private services. Typically, user names and profile pictures are shared between public social network profiles and private email services. An attacker first accesses the public social network to gather the victim's personal data and their size; then the attacker passively observes connections to the target private service, being able to tell when the victim user logs in, by observing the encrypted size of her personal data.

To sum up, the attacks above range from revealing the plaintext of requested URLs, to track users' behavior on target websites, to identify users who would otherwise expect their identity to be encrypted with the rest of the communication. Even finer attacks can be mounted by correlating different features, such as message sizes and their timing. Furthermore, such attacks become even more effective and powerful in the presence of a pervasive observer, which controls several sparse eavesdropping nodes.

3 Thwarting attacks based on message size analysis

What can be done to defeat or at least mitigate this kind of attacks? Naïvely, one would give all the Internet traffic the same shape. Unfortunately, the survey in [4] shows that either this is unsustainably inefficient, or any attempt to optimize it results in a substantial leak of the message sizes.

Other approaches try to randomly obfuscate the overall message size: some of them operate at the transport level, for instance by adding random padding to TLS fragments; others operate at the application level, for example by randomly splitting a single HTTP response in several overlapping fragments [5]. We claim that such countermeasures are fundamentally flawed for two reasons. First, they do not offer a deterministic level of protection: some sensitive parts of the plaintext may not be sufficiently protected in some runs of the communication, and crucially the application has no control over this. Second, random obfuscation is usually weak against repeated sampling: for example, if a user fills a log in form several times a day, or often refreshes a web page, few samples of the randomized communication are typically enough to recover the real size of the messages [7]. By comparison, encryption guarantees, at least to some extent, that all payload remains equally secret, independent of its position or the number of times it is encrypted.

If hiding the size of all messages effectively and efficiently seems unrealistic, pragmatically one can identify those parts of the message that are sensitive, and make sure that at least their size is effectively concealed. This constitutes a significant difference between classic encryption and side-channel mitigation. The former can be indiscriminately applied to the whole communication payload, and therefore can be plugged almost transparently. The latter instead is applied according to a semantic discrimination, and hence requires application cooperation in marking sensitive parts of the payload.

Application cooperation opens up a new series of challenges as well as possibilities. Here, a general discussion is presented; a more detailed and experimental analysis using TLS as a case study is given in [7].

One challenge is developer awareness: while security and privacy are often perceived as a necessary burden by developers, the awareness thereof can also lead to positive byproducts. Misconfigured security is the fifth most common web vulnerability according to OWASP: the more we allow developers to "plug security" without understanding it, the more misconfigured systems we will get. Developers who are required to assign sensitive levels to their application data are potentially more informed about the threat model and the consequences of an attack – and more pragmatically, they are accountable for the security choices they make.

On the positive side, application cooperation allows for precise and efficient countermeasures to message size analysis. The application can state which data are sensitive, and to what extent their size shall be concealed. Hence, one can rigorously measure the trade-off between message size indistinguishability and network congestion impact; interestingly, this trade-off can be controlled by the application over time, according to evolving privacy policies and network features.

Another challenge is the interaction between developers and the underlying security mechanisms. We claim that security mechanisms and protocols should expose a precise API to applications, to let the developers express their security requirements. This can range from function calls to identify sensitive data, to new language constructs to express constant time operations.

Finally, we discuss the layering of such message size analysis countermeasures. One may argue that, since the application needs to be aware anyway, one can directly implement countermeasures at the application level (say HTTP, SMTP). While this is certainly possible, and sometimes even desirable for very specific application scenarios, we claim that requiring each application to implement their own traffic analysis countermeasures in fact multiplies the chances that some of them will get it wrong; moreover, it may make the application itself distinguishable from other applications, by the very way it conceals message sizes.

We identify instead TLS as an amenable layer where to implement message size hiding, so that the wealth of applications that already rely on it can also benefit from the added message size analysis countermeasure. Technical details of this idea can be found in [6, 7]; here we stress that, even if TLS is the place where the countermeasure is implemented, still application cooperation is needed. In particular, TLS has to offer an enhanced API that allows the application to express its message size concealing requirements. Furthermore, we observe that the countermeasure works best when the application exchanges messages, as opposed to a (time sensitive) potentially unbound stream of bytes. While the size of sensitive parts of an unbound stream can be conceived, hiding the total size (or pattern) of a potentially unbounded stream is a task as hard as concealing the whole size of a communication.

Other approaches may try to embed such message size analysis countermeasures within lower layer protocols, such as IPSec. While we do not find any fundamental argument against this, we deem TLS more flexible and closely layered to the application, in such a way that makes TLS preferable to lower layer solutions.

References

[1] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *IEEE Symposium on Security and Privacy*, pages 191–206, 2010.

[2] Heyning Cheng and Ron Avnur. Traffic analysis of SSL encrypted web browsing, 1998.

[3] George Danezis. Traffic analysis of the HTTP protocol over TLS. Unpublished draft.

[4] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail. In *IEEE Symposium on Security and Privacy*, pages 332–346, 2012.

[5] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Network and Distributed Security Symposium*, 2011.

[6] Alfredo Pironti and Nikos Mavrogiannopoulos. Length hiding padding for the Transport Layer Security protocol. Internet Draft. Available at `http://tools.ietf.org/html/draft-pironti-tls-length-hiding-02`.

[7] Alfredo Pironti, Pierre-Yves Strub, and Karthikeyan Bhargavan. Identifying Website Users by TLS Traffic Analysis: New Attacks and Effective Countermeasures. Research Report RR-8067, INRIA, 2012.

[8] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy*, pages 19–30, 2002.

[9] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: a camouflage proxy for the Tor anonymity system. In *ACM Conference on Computer and Communications Security*, pages 109–120, 2012.