Evaluating the Efficacy of Gaussian Padding on Website Fingerprinting Attacks

Master thesis by Johannes Leupold
Date of submission: 01.09.2021

1. Review: Jean-Paul Degabriele
2. Review: Vukasin Karadzic
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department

IT Security

Cryptography and Network
Security

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Johannes Leupold, die vorliegende Masterarbeit gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 01.09.2021

J. Leupold

Abstract

Abstract

Contents

1 Introduction

This is the introduction.

2 Background Material

2.1 Theoretical Setting

Website Fingerprinting (WF) is a field of study focusing on data leaked from anonymization techniques and whether it is still sufficient to conclude which websites the user visited. Furthermore, a goal of Website Fingerprinting study is the evaluation of various countermeasures that limit the leaked data and thus thwart attempts to uncover the websites a certain user was visiting.

2.1.1 Threat Model

The Website Fingerprinting adversary is assumed to have access to the encrypted network traffic of the victim. However, he is not able to decrypt the traffic, or parts of it, so he can only draw conclusions based on metadata of the encrypted packets. In particular, the adversary observes

- packet length (full packet size in bytes),
- direction (whether the packet goes from Client $\rightarrow$ Server, or vice versa), and
- timing (amount of time between the first packet and the current packet).

This information also gives way to various derived measures, such as overall bandwidth consumed, overall time, or burst[1] length/size.

Before the adversary begins with the attack, he can create a database of websites the victim may visit and collect so-called *packet traces* for the websites. A packet trace is a sequence $\mathcal{T} = \langle p_1, p_2, ..., p_n \rangle$ of tuples $p_i = (t_i, l_i, d_i) \in \mathbb{N} \times \{52, \ldots, MTU\} \times \{\uparrow, \downarrow\}, i = 1, \ldots, n,$

[1] A *burst* is a contiguous sequence of packets going in the same direction

where t_i, l_i and d_i denote the time, length and direction for a single observed packet, respectively. The packet length can never be smaller than 52 bytes, as that is the size of the smallest possible TCP/IP packet, the ACK packet. The maximum packet size MTU (also called *Maximum Transmission Unit*) is dependent on the used network technologies. For Ethernet, the MTU is 1500 bytes, while for WiFi (IEEE 802.11) it is 2312 bytes. Note that the adversary is assumed to collect his database of website traces under the same technical preconditions (network technologies) that his victim uses to access them. When carrying out the attack, the adversary observes packet traces for particular page loads of the victim. He may then use any algorithm and his prepared database to try and deduce which websites out of the pre-selected set the victim visited. When a WF countermeasure is in place, the adversary is assumed to be aware of it such that he can collect his trace database under the same countermeasure.

2.1.2 Attacks and Countermeasures

To successfully make conclusions on the websites a victim visited, the attacker needs to classify the packet traces he observed to be an instance of one of the websites in his database. He may achieve this by randomly guessing the instance's class, but this naive method can't be expected to yield good results for higher numbers of websites. Therefore, the adversary may employ a Machine Learning technique, consisting of a feature extraction[2] and a classification algorithm. A classification algorithm is trained using a number of examples from all classes including the class labels (the *training set*). In the following, an overview of common classification algorithms is given.

Nearest Neighbours The Nearest Neighbours classifier (also called NN) is a simple algorithm that assigns to each instance the class of its nearest neighbor according to some distance metric. It may be extended to k-Nearest-Neighbours (kNN) by considering the nearest k neighbors and taking the majority vote of their classes. One may also apply weighting to the neighbors based on the distance from the point to be classified.

Naive Bayes The Naive Bayes classifier tries to estimate probability distributions for the attribute values conditioned on all classes from the training data. It estimates the probability distributions $p(C)$ and $p((f_1, ..., f_k) \mid C)$ from the data for all classes C and attribute values $f_1, ..., f_k$. To classify a new example e $(g_1, ..., g_k)$, Bayes' rule is

[2] *Feature Extraction* refers to transforming a certain observation into a number of (mostly numerical) attributes, called *features*

used to calculate $p(C \mid (g_1, ..., g_k))$ for each class C and the class yielding the highest probability is returned. The method used to estimate the probability distributions from the data can be chosen according to the requirements of the problem. Common examples include simply fitting a normal distribution using a maximum likelihood approach, and Kernel Density Estimation (see below).

Kernel Density Estimation (KDE) The term ”Kernel Density Estimation” refers to a method used to estimate the underlying probability density function a certain set of samples was drawn from. To this end, individual functions, called *kernels* are centered at each sample. To evaluate the density function at a certain point, all individual kernels are evaluated and summed up. Equation 2.1 shows the general formula with n being the number of training samples, k being the kernel function and h being a parameter called *bandwidth* that controls smoothing.

$$p(x) = \frac{1}{nh} \sum_{i=1}^{n} k \left(\frac{x - x_i}{h} \right) \tag{2.1}$$

A common choice for k is the standard normal density function.

Support Vector Machine (SVM) The SVM is a sophisticated classification algorithm trying learn a hyperplane to separate the examples of different classes in the feature space. This is achieved by selecting the hyperplane that yields the largest separation between the classes, i.e. the one that maximizes the distance to the nearest points of each class. Support Vector Machines are well suited for working with high-dimensional feature spaces and may use a non-linear transformation of data points into an implicit higher-dimensional feature space. This transformation is called a *kernel* (not to be confused with the kernel used in Kernel Density Estimation). [1, 5]

A defense against WF attacks aims at disguising the identifying patterns in the packet traces the attacker can observe. To this end, packets lengths may be artificially changed, they may be split up into multiple packets, or the timing might be changed. There exist three main classes of defenses, while, naturally, some countermeasures might not fit one particular class, but exhibit characteristics of multiple classes.

Padding Padding schemes are among the simplest countermeasures. They add certain amounts of dummy data to each packet up to the MTU. Padding schemes may be *deterministic* or *randomized*, with deterministic schemes always returning the same output trace when presented the same input trace, while this isn’t the case for randomized padding. When using this type of defense, packets can only grow, but never shrink. Also, packet counts and timing are not affected.

Noise Defenses of this class try to disguise the visited web page by deliberately adding noise to the transmission. This can be done for example by loading a second randomly chosen web site in parallel to the requested site [10] or randomizing the order of HTTP requests in the web browser [11]. Countermeasures using approach may affect packet timing and order and insert new packets into the trace.

Morphing Morphing countermeasures are the most complex defenses, trying to hide the actual distribution of packet sizes. They may simply try to make all websites look equal, or at least similar, [8, 3, 2] or to make a certain web site look like a different one [12]. Packet traces may be changed in many ways when using a morphing defense, like adding extra packets, growing or splitting packets, or changing timing.

2.2 Gaussian Padding

Gaussian padding is a specific form of randomized padding. The amount of dummy data added to a packet is drawn from a rounded normal distribution, as opposed to Uniform Padding, where padding sizes are drawn according to the uniform distribution. Furthermore, the tails of the distribution are clipped on both sides, as padding can never take negative values and packets may never become bigger than the MTU. The distribution is parameterized with the desired mean padding size, becoming the mean μ of the normal distribution. The standard deviation σ can be chosen such that the truncated tail corresponds to as little probability mass as desired. For our purpose, choosing $\sigma = \frac{\mu}{3}$ is sufficient, as in this case only approximately 0.1% of the probability mass is allocated to negative values. Figure 2.1 shows an example distribution function for a truncated rounded normal distribution.

Sampling from a rounded normal distribution is straight-forward by simply sampling from a continuous normal distribution and rounding the value. The truncation can be achieved by rejecting and resampling values that fall out of the desired interval, or by using the method described by Hülsing et al. in [9], which allows to sample from a rounded Gaussian in constant time, thus mitigating cache timing attacks on the sampling algorithm. While these are out of scope under the threat model of WF, this fact is of particular interest for different fields of research, such as Lattice-based cryptography. [9]

When applying Gaussian padding, the amount of data added to each packet may either be sampled from the distribution independently for each individual packet or once for every session (i.e. page load), leading to *packet-random* Gaussian padding and *session-random* Gaussian padding, respectively.

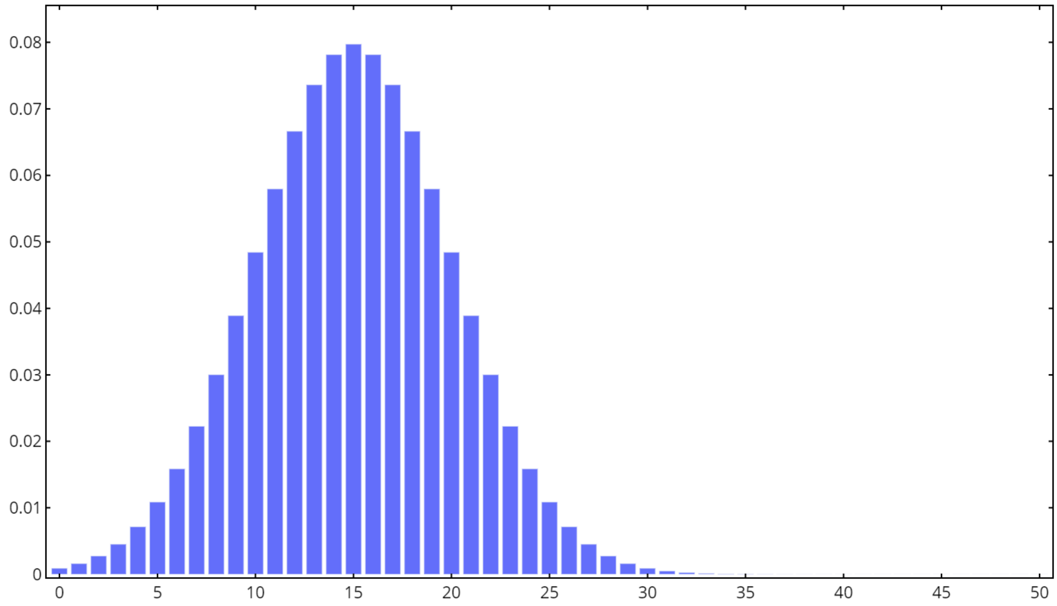

Figure 2.1: An example for a rounded normal distribution, truncated to the left at 0 with a mean padding size of $\mu = 15$ and a standard deviation of $\sigma = 5$

2.3 Security Bound Estimation According to Cherubin [4]

When evaluating WF attacks and defenses, the traditional approach is to empirically measure their performance on a previously collected data set of packet traces. However, such empirical evaluations are susceptible to noise and fail to produce provable statements on the security of a certain defense. While a defense may perform particularly well against the attacks it is evaluated on, it may just as well horribly fail on a different attack that wasn't considered yet. To address this problem and further advance research on provable security evaluation of WF defenses, Cherubin introduces a novel method centered around the Bayes error in [4].

The *Bayes error* is defined as the minimum probability of error that any classifier can commit given a joint probability distribution on the features and class labels. Intuitively speaking, the Bayes error corresponds to the area of uncertainty in the probability distribution, i.e.

the area where the conditional probabilities $p(x \mid c_i)$ and $p(x \mid c_j)$ overlap[3] for any two classes $c_i \neq c_j$ and features x.

To formalize, let $\mathcal{X}$ be the feature space and $\mathcal{C}$ the set of all class labels. Let $p(x, c)$ be a joint probability distribution on $\mathcal{X} \times \mathcal{C}$. Then for the set of classifiers $\mathcal{F} = \{f \mid f : \mathcal{X} \to \mathcal{C}\}$ with R_f being the error of a classifier $f \in \mathcal{F}$ according to p, the Bayes error R^* is defined as

$$R^* = \min_{f \in \mathcal{F}} R_f \tag{2.2}$$

This minimum error is achieved by the *Bayes classifier* $f^* \in \mathcal{F}$ that assigns to each example x the class label c maximizing the probability $p(c \mid x)$ [7].

$$f^*(x) = \arg\max_{c \in \mathcal{C}} p(c \mid x) \tag{2.3}$$

$$R^* = R_{f^*} \tag{2.4}$$

As the true joint probability distribution $p(x, c)$ is typically not known in practice, the exact value of the Bayes error can't be calculated. However, there are well-known methods to estimate a lower bound for the Bayes error. Cherubin uses an estimate based on the NN classifier (see section 2.1.2) first presented by Cover and Hart [6] deriving a lower bound $\widehat{R}^*$ for the Bayes error R^*

$$\widehat{R}^* = \frac{L-1}{L}\left(1 - \sqrt{1 - \frac{L}{L-1}\widehat{R}_{NN}}\right) \leq R^* \tag{2.5}$$

where $\widehat{R}_{NN}$ is the empirical error of the NN classifier on a data set containing $L = |\mathcal{C}|$ classes

[3] That is, both probabilities are non-zero

3 Prior Work

4 Experimental Methodology

4.1 Trace Data

4.2 Evaluating Attack Performance

4.3 Error Bound Estimation

5 Results

5.1 Empirical Performance of Gaussian Padding

5.2 Estimated Security Bounds

6 Discussion

7 Conclusion

Bibliography

[1] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. “A training algorithm for optimal margin classifiers”. In: *Proceedings of the fifth annual workshop on Computational learning theory - COLT ’92*. ACM Press, 1992. DOI: 10.1145/130385.130401.

[2] Xiang Cai, Rishab Nithyanand, and Rob Johnson. “CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense”. In: *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, Nov. 2014. DOI: 10.1145/2665943.2665949.

[3] Xiang Cai et al. “A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Nov. 2014. DOI: 10.1145/2660267.2660362.

[4] Giovanni Cherubin. “Bayes, not Naïve: Security Bounds on Website Fingerprinting Defenses”. In: *Proceedings on Privacy Enhancing Technologies* 2017.4 (Oct. 2017), pp. 215–231. DOI: 10.1515/popets-2017-0046. URL: https://petsymposium.org/2017/papers/issue4/paper50-2017-4-source.pdf.

[5] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. DOI: 10.1007/bf00994018.

[6] T. Cover and P. Hart. “Nearest neighbor pattern classification”. In: *IEEE Transactions on Information Theory* 13.1 (Jan. 1967), pp. 21–27. DOI: 10.1109/tit.1967.1053964.

[7] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley John + Sons, Oct. 1, 2000. ISBN: 0471056693. URL: https://www.ebook.de/de/product/3244086/richard_o_duda_peter_e_hart_david_g_stork_pattern_classification.html.

[8] Kevin P. Dyer et al. “Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail”. In: *2012 IEEE Symposium on Security and Privacy*. IEEE, May 2012. DOI: 10.1109/sp.2012.28. URL: https://cise.ufl.edu/~teshrim/tmAnotherLook.pdf.

[9] Andreas Hülsing, Tanja Lange, and Kit Smeets. “Rounded Gaussians: Fast and Secure Constant-Time Sampling for Lattice-Based Crypto”. In: *Public-Key Cryptography – PKC 2018*. Springer International Publishing, 2018, pp. 728–757. DOI: 10.1007/978-3-319-76581-5_25.

[10] Andriy Panchenko et al. “Website Fingerprinting in Onion Routing Based Anonymization Networks”. In: *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society - WPES ’11*. ACM Press, 2011. DOI: 10.1145/2046556.2046570. URL: https://dl.acm.org/doi/pdf/10.1145/2046556.2046570.

[11] Mike Perry. *Experimental Defense for Website Traffic Fingerprinting*. Sept. 4, 2011. URL: https://blog.torproject.org/experimental-defense-website-traffic-fingerprinting (visited on 08/11/2021).

[12] Charles Wright, Scott Coull, and Fabian Monrose. “Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis”. In: *Proceedings of the 16th annual Network and Distributed System Security Symposium - NDSS ’09*. NDSS, Jan. 2009. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/wright.pdf.

List of Figures

List of Tables