Free and Open
COMMUNICATIONS
https://foci.community on the Internet

Crowdsourcing the Discovery of Server-side Censorship Evasion Strategies

Nhi Tran
University of Maryland

Kevin Bock
University of Maryland

Dave Levin
University of Maryland

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit `https://creativecommons.org/licenses/by/4.0/` or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Free and Open Communications on the Internet 2023(1), 88-90

© 2023 Copyright held by the owner/author(s).

Crowdsourcing the Discovery of Server-side Censorship Evasion Strategies

Nhi Tran, Kevin Bock, Dave Levin
University of Maryland

1 Introduction

A recent advance in the field of censorship evasion is that of *server-side* censorship evasion, in which a server manipulates packets during a TCP three-way handshake in order to bypass censorship on the client's behalf [1]. The packet manipulations exploit weaknesses in how censors track or tear-down connections so that they are unable to censor a client's forbidden request. Server-side evasion strategies are particularly appealing because they do not require clients to take *any* additional action or install additional software; indeed, the client avoids censorship even if they did not realize they were being censored in the first place. This results in easier and safer deployment, and it has been adopted by multiple censorship evasion tools [7].

A limitation of server-side evasion is the difficulty in discovering the evasion strategies in the first place. In the past, researchers studied censorship systems and handcrafted packet manipulation strategies that could bypass the censor [3, 4, 8]. More recently, researchers have proposed multiple solutions towards automating the discovery of these packet manipulation strategies, including Geneva [2], *Alembic* [5], and SymTCP [9]. All of these require client-side instrumentation for their training phase: even when packet manipulation strategies are deployed exclusively at the server, each tool still requires the control of a client during training to issue requests through a censor. This poses a significant limitation: it is challenging for researchers to get access to machines in every network of interest.

In this extended abstract, we sketch a design for how we can automatically discover censorship evasion strategies without having direct control over the clients that participate in training. We propose a distributed training model in which users can visit a website with an unmodified web browser, and, after providing their informed consent, contribute to discovering censorship evasion strategies. The idea is that users' browsers can generate forbidden requests to a server under our control, and we can make use of those interactions with the censor to discover censorship evasion strategies. By allowing many users under a given censor to contribute to a single training pool, we can distribute the burden of training and gain broader test coverage for strategies. We describe our distributed training model as an extension of Geneva.

2 Background

Geneva [2] is a genetic algorithm that discovers packet manipulation strategies to bypass censorship, through the process of evolution over a series of discrete generations. The censorship evasion strategies that Geneva discovers only need to be run on one side of the connection: either client-side or server-side. Training Geneva requires both a client and a server whose communication crosses a censor. In server-side training, the client merely sends unmodified requests that trigger censorship; all of the packet manipulations occur server-side.

While it trains, Geneva maintains a *population pool* of strategies, wherein each strategy is a series of modifications to perform on network traffic. It evaluates each strategy using a fitness score, and those with the highest fitness are more likely to survive to the next generation, at which point they may also undergo mutations. This process repeats iteratively, logging any successful evasion strategies. At first, Geneva's initial population pool of strategies generally performs very poorly, but as the worst strategies are eliminated, Geneva explores the space of potential strategies that do not break the TCP connection.

3 Design

Put simply, our tool allows any willing user to use their browser as the client in performing Geneva's server-side training.

User Experience From the user's perspective, they visit a website in an unmodified browser with minimal requirements (JavaScript), from within a network that the user suspects is experiencing censorship. Before conducting any experiments, the user is presented with an informed consent form describing the experiment as well as the potential benefits and risks.[1] If the user consents, then the browser automatically coordinates with a server under our control and runs the experiments;

[1]OONI has an easy-to-understand description of risks [6], which we believe can serve as a good starting point for our informed consent.

during this, the webpage informs the user that the experiment is underway, showing them the specific domains being used in its testing, and including a button that allows them to stop the experiment at any time. If the server is unable to trigger or detect censorship, then it terminates the experiment and informs the user. When the training is complete, the webpage debriefs the user, providing a detailed summary of what transpired (number of requests, which domains), and the project's contact information.

Despite such minimal user interaction and no downloads whatsoever, we posit that this will suffice for allowing us to discover new censorship evasion strategies in a wide range of new networks.

Geneva **training** The servers we control will run many instances of Geneva's genetic algorithm and host the public facing website. Specifically, each server will run many *training pools*; each pool is a population of strategies that Geneva's genetic algorithm is evolving. The server maintains at least one training pool for each censorship system we want to train against; in our initial version, we plan for only a few population pools and to start by targeting individual countries with centralized censorship systems (such as China or Iran). As users are available to contribute to each training pool, the server instructs their browser to make forbidden requests to isolated ports on which the server runs select strategies from the pool. By monitoring the success or failure of each strategy, the server can evolve each training pool in isolation for each censor. As we are constrained by the types of requests that a browser can generate and direct at our server, we are limited to training against HTTP and HTTPS censorship.

Protecting Users The most important consideration of this design is protecting users. First, we require that all users provide informed consent before participating in training. Our system will explain what their browser will do in clear, concise language, (internationalized to different languages) and the system will require that users give consent by typing a message that says they understand and agree.

Even with informed consent, some users may not accurately conceptualize the risk of participating in training. To reduce the number of times each user will trigger censorship, the system will impose limitations on how frequently the user can generate forbidden requests.

It is also critical that the system protects user privacy. We will collect the minimal amount of information on users as possible. We will not store any source IP addresses or any identifying information about their browser; instead, we will store only AS-level information alongside randomized, short-lived client identifiers. Fortunately, this coarse-grained information suffices for server-side evasion strategies.

Identifying the Censor If multiple users from different countries train in the same population pool, clients may face difficulties in triggering censorship, as different countries have different censorship systems put into place. To avoid these difficulties, the server must identify which censor a user is coming from. The server will start with identifying the user's ASN and determine if that ASN has already been tested to trigger censorship. If the server does not know how to trigger censorship for that ASN, the user cannot participate in the training session at that time. However, if the server does know how, the user will then be allowed to participate in the training session and send requests to server, testing if the censor will behave in the way that we expect it to.

Triggering Censorship In order to evaluate censorship evasion strategies during training, we must be able to trigger censorship destined to our server from a browser. Compared to other networking tools, browsers are more limited in the types of requests they can generate. For example, we cannot independently control the `Host:` header of HTTP requests or the Server Name Indication (SNI) field of TLS requests, which commonly store the destination domain of a request. We propose two paths forward to trigger censorship.

First, we plan to use *keyword-based censorship*. This would allow us to trigger censorship by including forbidden keywords in requests. Some censors have been known to censor specific keywords when included in HTTP parameters, including the GFW [1,2]. This gives us an easy mechanism to trigger HTTP censorship, but it has at least two limitations. *First*, it will not allow us to trigger censorship everywhere, because not all censors employ keyword-based censorship. *Second*, the mechanisms by which a country censors keyword-based censorship (and thus the strategies that evade them) do not necessarily apply to other forms of censorship.

Our second planned approach for triggering censorship is to *abuse overblocking*. Researchers have identified that some censors use regular expressions to block certain domain names, and sometimes, those regular expressions are too broad. For example, a country looking to block *torproject.org* with a regular expression might accidentally also block *mentorproject.org*. We propose registering domains that contain other censored domains and pointing them back to our server. In this way, we can legitimately address requests to a censored domain and have it still be routed to our server.

4 Conclusion

Server-side censorship evasion strategies are compelling because they do not require any client-side changes whatsoever. In this extended abstract, we have sketched the design of a system that will allow us to discover new server-side strategies with unmodified clients, as well. Central to our design is the respect and protection of users; to this end, we believe that informed consent is of paramount importance. We welcome feedback on this work-in-progress tool.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Micah Sherr, for their helpful feedback. This work was supported in part by NSF award CNS-1943240.

References

[1] Kevin Bock, George Hughey, Louis-Henri Merino, Tania Arya, Daniel Liscinsky, Regina Pogosian, and Dave Levin. Come as You Are: Helping Unmodified Clients Bypass Censorship with Server-side Evasion. In *ACM SIGCOMM*, 2020.

[2] Kevin Bock, George Hughey, Xiao Qiang, and Dave Levin. Geneva: Evolving Censorship Evasion. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[3] Sheharbano Khattak, Mobin Javed, Philip D. Anderson, and Vern Paxson. Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2013.

[4] Fangfan Li, Abbas Razaghpanah, Arash Molavi Kakhki, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. lib.erate, (n): A library for exposing (traffic-classification) rules and avoiding them efficiently. In *ACM Internet Measurement Conference (IMC)*, 2017.

[5] Soo-Jin Moon, Jeffrey Helt, Yifei Yuan, Yves Bieri, Sujata Banerjee, Vyas Sekar, Wenfei Wu, Mihalis Yannakakis, and Ying Zhang. Alembic: Automated Model Inference for Stateful Network Functions. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2019.

[6] OONI. Risks: Things you should know before running OONI Probe. `https://ooni.org/about/risks/`.

[7] Psiphon Packet Manipulation: Packetman. `https://pkg.go.dev/github.com/Psiphon-Labs/psiphon-tunnel-core/psiphon/common/packetman`.

[8] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V. Krishnamurthy. Your State is Not Mine: A Closer Look at Evading Stateful Internet Censorship. In *ACM Internet Measurement Conference (IMC)*, 2017.

[9] Zhongjie Wang, Shitong Zhu, Yue Cao, Zhiyun Qian, Chengyu Song, Srikanth V. Krishnamurthy, Kevin S. Chan, and Tracy D. Braun. SymTCP: Eluding Stateful Deep Packet Inspection with Automated Discrepancy Discovery. In *Network and Distributed System Security Symposium (NDSS)*, 2020.