
# USENIX Security '24 Artifact Appendix: SpotProxy: Rediscovering the Cloud for Censorship Circumvention

Patrick Tser Jern Kon  Sina Kamali[‡]  Jinyu Pei[†]
Diogo Barradas[‡]  Ang Chen  Micah Sherr[⋆]  Moti Yung[◇,○]
*University of Michigan*  [‡]*University of Waterloo*  [†]*Rice University*
[⋆]*Georgetown University*  [○]*Columbia University*  [◇]*Google*

## A Artifact Appendix

### A.1 Abstract

SpotProxy is a censorship resistance system that uses cost-effective and high-churn cloud instances to maximize the circumvention utility of cloud-hosted proxies. To achieve this, SpotProxy designs a circumvention infrastructure that constantly searches for cheaper VMs and refreshes the fleet for anti-blocking. We adapt Wireguard and Snowflake for use with SpotProxy, and demonstrate that our active migration mechanism allows clients to seamlessly move between proxies without degrading their performance or disrupting existing connections. We show that SpotProxy leads to significant cost savings, and that SpotProxy's rejuvenation mechanism enables proxies to be replenished frequently with new addresses. This artifact evaluation aims to show that the SpotProxy system is accessible and functional.

### A.2 Description & Requirements

#### A.2.1 Security, privacy, and ethical concerns

The artifact does not pose any particular risk to the evaluators' machines, data privacy, etc., or raise any other notable ethical concerns.

#### A.2.2 How to access

All artifacts can be accessed centrally through our GitHub repository at https://github.com/spotproxy-project/spotproxy. All instructions are provided in the README file within the repository. The stable link for SpotProxy can be found at `https://github.com/spotproxy-project/spotproxy/tree/1abbfe79ef8358afe81c0db491e4ec528a42773f`.

#### A.2.3 Hardware dependencies

None.

#### A.2.4 Software dependencies

SpotProxy has been evaluated on x86-64 Ubuntu 22.04 VMs. It requires access to an AWS account with the ability to create EC2 VMs and provision Elastic IPs (default AWS-assigned quotas should be sufficient for this artifact evaluation). It also requires numerous dependencies (e.g., django) that can be installed by following the instructions provided in Sec. A.3.1.

#### A.2.5 Benchmarks

Our historical AWS SpotVM cost analysis depends on a publicly available dataset hosted by a researcher at https://github.com/ericpauley/aws-spot-price-history. We describe our usage of it in more detail within experiment E2 in Sec. A.4.2.

### A.3 Set-up

#### A.3.1 Installation

Clone our repository and follow the installation steps in `docs/INSTALLATION.md`. A variety of installation procedures will be performed, including: configuring access to AWS CLI, installing Python dependencies, and building the artifact.

#### A.3.2 Basic Test

Follow the steps in `docs/INSTALLATION.md#basic-test`. to run a basic test. This validates if the AWS CLI was configured correctly and if the required packages were installed.

### A.4 Evaluation workflow

We list all our claims and the experiments we performed to support them. For the Artifacts functional badge, we provide minimal working examples for all our system components in Sec. A.4.2.

### A.4.1 Major Claims

**(C1):** SpotProxy continuously searches for the cheapest instances via cost arbitrage. This is proven by experiment (E1) which demonstrates the functionality of cost arbitrage, and experiment (E2) which demonstrates the efficacy of cost arbitrage through historical data.

**(C2):** Both of SpotProxy's rejuvenation mechanisms function and are capable of refreshing the fleet. This is proven by experiment (E3) where we perform actual rejuvenations on the cloud.

**(C3):** SpotProxy's active migration works on both Snowflake and Wireguard with little additional overhead. This is proven by experiment (E4) where we perform periodic migration over cloud-hosted proxies.

**(C4):** SpotProxy is capable of facilitating circumvention efforts against two main types of simulated censor attacks. This is proven by experiment (E5) where we simulate the effects of these attacks against periodic rejuvenation/relocation by SpotProxy.

### A.4.2 Experiments

We now provide minimal working examples for each of our major claims.

**(E1):** *[Cost arbitrage] [a few minutes]: creates 1 instance of the cheapest VM currently available.*
**How to:** *Follow the steps provided in* `docs/instance_manager_setup.md`.
**Preparation:** *This assumes that the installation step in Sec. A.3.1 has already been executed.*
**Results:** *The script will return the instance ID that was created. Note that while the script continuously checks for cheaper instances, we only perform this once in this experiment due to unpredictable cost fluctuations. Instead, we defer to experiment E2 to measure the effects of continuous cost arbitrage.*

**(E2):** *[Cost arbitrage efficacy] [a few minutes]: analyzes AWS SpotVM historical pricing data to determine the cost savings afforded by SpotProxy's cost arbitrage mechanism.*
**How to:** *Follow the steps provided in* `docs/historical_cost_analysis.md`.
**Preparation:** *This assumes that the installation step in Sec. A.3.1 has already been executed.*
**Results:** *The script will return a JSON file containing a month-by-month breakdown of various metrics including cost arbitrage intervals throughout the month. The source data is extracted from the dataset mentioned in Sec. A.2.5.*

**(E3):** *[Rejuvenation functionality test] [a few minutes]: performs instance and live IP rejuvenation periodically until the script is stopped.*
**How to:** *Follow the steps provided in* `docs/instance_manager_setup.md`.
**Preparation:** *This assumes that the installation step in Sec. A.3.1 has already been executed.*
**Results:** *The script will return the instance/NIC IDs details produced in each rejuvenation period.*

**(E4):** *[Active migration functionality test] [a few minutes]: this is essentially a minimal working example that incorporates our cost arbitrage and rejuvenation functionalities presented in the earlier experiments, but with the addition of clients connected to the system, and using active migration to maintain seamless connectivity despite rejuvenation.*
**How to:** *Follow the steps provided in* `docs/controller_setup.md` *to instantiate the controller. Then, instantiate the instance manager at* `docs/instance_manager_setup.md` *, and the client at* `docs/client_setup.md`. *Our minimal working example assumes the use of a Wireguard implementation throughout.*
**Preparation:** *This assumes that the installation step in Sec. A.3.1 has already been executed.*
**Results:** *This test will produce various logs including instance/NIC details, client-to-proxy assignments, and client access logs.*

**(E5):** *[Circumvention efficacy test] [a few minutes]: simulates the connectivity effects of two censor attacks against periodic rejuvenation/relocation by SpotProxy, across time.*
**How to:** *Follow the steps provided in* `docs/circumvention_efficacy_setup.md`.
**Preparation:** *This assumes that the installation step in Sec. A.3.1 has already been executed.*
**Results:** *This script will produce the connectivity ratios across time for a specific censoring agent ratio.*

## A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2024/.