GET /out: Automated Discovery of Application-Layer Censorship Evasion Strategies

Michael Harrity, Kevin Bock, Frederick Sell, and Dave Levin, *University of Maryland*

https://www.usenix.org/conference/usenixsecurity22/presentation/harrity

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

In this paper, we present the first techniques to automate the discovery of new censorship evasion techniques purely in the application layer. We present a general solution and apply it specifically to HTTP and DNS censorship in China, India, and Kazakhstan. Our automated techniques discovered a total of 77 unique evasion strategies for HTTP and 9 for DNS, all of which require only application-layer modifications, making them easier to incorporate into apps and deploy. We analyze these strategies and shed new light into the inner workings of the censors. We find that the success of application-layer strategies can depend heavily on the type and version of the destination server. Surprisingly, a large class of our evasion strategies exploit instances in which censors are more RFC-compliant than popular application servers.

For the purposes of this submission, our artifacts are (1) the strategies we present in the paper and (2) the code used to implement them. We developed our fuzzer by building off of the open-source Geneva project (`https://github.com/Kkevsterrr/geneva`), but our code has not yet merged into that repository publicly. Therefore, we have provided the full modified codebase to assist in the evaluation.

For this artifact evaluation, we demonstrate how the reader can evaluate (1) that our strategies can generate modified requests; (2) that our strategies can evade censorship. Optionally, the evaluator can test for themselves that our tool can fuzz HTTP requests.

A.2 Artifact check-list (meta-information)

- **Algorithm: A new algorithm for bypassing censorship with modifications to application-layer data.**
- **Program: The program—an extension to our prior work, Geneva—that implements the algorithm. Specifically, we are asking you to evaluate the engine that runs the strategies that our algorithm discovered, not to run the genetic algorithm that *found* the strategies.**
- **Security, privacy, and ethical concerns:**
- **Metrics: Whether or not it is able to access otherwise restricted content.**
- **Output: HTTP output (from running `curl`).**
- **Experiments: Re-run several of our censorship-evading strategies.**
- **How much disk space required (approximately)?: Very little (megabytes); a free-tier Ubuntu AWS instance would suffice.**
- **How much time is needed to prepare workflow (approximately)?: Minutes (set up an Ubuntu VM, install, and run).**
- **How much time is needed to complete experiments (approximately)?: Minutes, including setup.**
- **Publicly available (explicitly provide evolving version reference)?:** Yes
- **Code licenses (if publicly available)?:** BSD 3-Clause "New" or "Revised" License
- **Archived (explicitly provide DOI or stable reference)?:** `https://zenodo.org/record/6692160`

A.3 Description

A.3.1 How to access

The code is available at `https://zenodo.org/record/6692160`

We developed our fuzzer by building off of the open-source Geneva project (https://github.com/Kkevsterrr/geneva), but our code has not yet merged into that repository publicly. Therefore, we have provided the full modified codebase to assist in the evaluation.

Documentation for Geneva is available at `https://geneva.readthedocs.io/en/latest/`. To ease the burden on the artifact evaluators, we have provided just the required steps to evaluate our artifact below.

A.3.2 Software dependencies

See the dependency installation in the installation setup below (some basic python libraries).

A.3.3 Security, privacy, and ethical concerns

As this is accessing a server that we control, we do not anticipate any security, privacy, or ethical concerns. However, we do suggest that the evaluator run from a machine inside of a country that does not prosecute censorship circumvention (e.g., from within the United States).

A.4 Installation

1. **Setup a machine:** To test our artifacts, we recommend using Ubuntu 18.04, and although it may be possible to reproduce our results using a virtual machine, it is ideal if the machine has a public IP address and is not behind a NAT, to avoid any potential interference from your host or home network.

   We recommend setting up a free-tier Amazon EC2 machine with Ubuntu 18.04. Once your machine is up and running, transfer our artifact submission to it.

2. **Install dependencies:** Next, install the dependencies for Geneva:

   ```
   # cd geneva/
   # sudo apt-get install build-essential python-dev
   libnetfilter-queue-dev libffi-dev libssl-dev
   iptables python3-pip
   ...
   # python3 -m pip install -r requirements.txt
   ...
   ```

3. **Test strategies:** To validate our strategies, reviewers can test (1) that our code generates the strategies it says it does and (2) that these strategies are actually effective at evading censorship.

In our paper, we present over 85 strategies, and tested these across 8 servers against 3 different censoring regimes, using vantage points we obtained in those locations. Unfortunately, for security reasons, we cannot give evaluators direct access to these vantage points.

To make evaluation easier, we have set up an HTTP server running Apache 2.4.6 in Kazakhstan with `www.youporn.com` as the required Host header (we have provided the IP address and port of that server in our submission). Kazakhstan operates censorship bidirectionally (forbidden requests sent into the country are censored in the same way as requests leaving the country), which enables evaluators to trigger HTTP censorship remotely to our vantage point. If evaluators wish to test the rest of our strategies to the other censored countries or our DNS strategies, we can offer advice and work with the evaluators as to how to best purchase vantage points in those locations and test the remaining strategies.

With the server we set up, an evaluator can safely test a sample of our strategies to this IP address and verify that they do evade censorship. To maintain reviewer anonymity, we will discard this server's logs. In the below guide, we have removed our server's IP address; wherever you see `<ip>`, please replace with the IP address we provided in HotCrp

First, we will verify that you can reach our server. Start by `curl`ing to our server. Note that Since `curl` will not set a `Host` header by default, you should see a 403 Forbidden response:

```
$ curl <ip>:8000
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access /
on this server.</p>
</body></html>
```

Second, we will verify that you can experience censorship to our vantage point. Start `tcpdump` to our IP address in the background, and make a request with `www.youporn.com` in the `Host` header. You should see that censorship occurs (the censor null-routes our request), and the request is retransmitted:

```
$ sudo tcpdump -f -n 'host <ip>' &
$ curl -H "Host: www.youporn.com" <ip>:8000
03:31:11.724898 IP 172.172.172.172.38520 > <ip>.8000: Flags [S], seq
2244093827, win 62727, options [mss 8961,sackOK,TS val 585883751 ecr
0,nop,wscale 7], length 0
03:31:11.840224 IP <ip>.8000 > 172.172.172.172.38520: Flags [S.], seq
1501790452, ack 2244093828, win 65160, options [mss 1460,sackOK,TS val
2426661480 ecr 585883751,nop,wscale 7], length 0
03:31:11.840266 IP 172.172.172.172.38520 > <ip>.8000: Flags [.], ack 1,
win 491, options [nop,nop,TS val 585883867 ecr 2426661480], length 0
03:31:11.840609 IP 172.172.172.172.38520 > <ip>.8000: Flags [P.], seq
1:80, ack 1, win 491, options [nop,nop,TS val 585883867 ecr
2426661480],
length 79
03:31:12.219434 IP 172.172.172.172.38520 > <ip>.8000: Flags [P.], seq
1:80, ack 1, win 491, options [nop,nop,TS val 585884246 ecr
2426661480],
length 79
03:31:12.571446 IP 172.172.172.172.38520 > <ip>.8000: Flags [P.], seq
1:80, ack 1, win 491, options [nop,nop,TS val 585884598 ecr
2426661480],
length 79
03:31:13.275434 IP 172.172.172.172.38520 > <ip>.8000: Flags [P.], seq
1:80, ack 1, win 491, options [nop,nop,TS val 585885302 ecr
2426661480],
length 79
```

Shut down the `tcpdump` before continuing.

A.5 Experiment workflow

Next, we can evaluate that Geneva can implement the strategies in our paper. Since our paper reported over 85 strategies, to reduce the burden on the evaluators, we have sampled 3 strategies that work with this server and evade censorship in Kazakhstan.

- Strategy 1 (Long Request):

  ```
  [HTTPRequest:host:*]-insert{%20:start:value:1600}-|
  ```

- Strategy 2 (Host Header Whitespace):

  ```
  [HTTPRequest:host:*]-insert{%20:end:value:1}-|
  ```

- Strategy 3 (Request Line Whitespace):

  ```
  [HTTPRequest:method:*]- insert{%0A:start:value:1}-| \/
  ```

To use Geneva to test a strategy, we use Geneva's `--test-type` flag to invoke our HTTP fitness function (called `application_http`), and can test strategies with the following:

```
$ python3 evolve.py --test-type application_http --log debug --port 8000
--server <ip> --headers Host:www.youporn.com --eval-only "<STRATEGY HERE>"
```

Explanation of flags:

- `--test-type`: The fitness function plugin to invoke, in this case, `application_http`
- `--log`: the log level to use (debug, so the evaluator can see the strategy running and the exact request it sends on the wire)
- `--port`: the port the server is listening on (8000 for our server)
- `--server`: the server to test with; supply our IP address.
- `--eval-only`: this is a Geneva flag instructing it to evaluate a single strategy and then exit. Provide the strategy to test here.

A.6 Evaluation and expected results

Our primary claim is that these modifications to application-layer data permit evasion of censorship. You can use the above steps to run individual strategies (either the ones we selected, or any of the ones from the paper: you should be able to simply copy-paste them where it says "`STRATEGY HERE`") and should see something to the effect of the following:

```
$ sudo python3 evolve.py --test-type application_http --log debug
--port 8000 --server <ip> --headers Host:www.youporn.com --eval-only
"[HTTPRequest:host:*]-insert{%20:end:value:1}-|"
2022-06-22 20:51:13 DEBUG:Launching strategy evolution: --test-type
application_http --log debug --port 8000 --server <ip> --headers
Host:www.youporn.com --eval-only [HTTPRequest:host:*]-
insert{%20:end:value:1}-|
2022-06-22 20:51:13 INFO:Logging results to trials/2022-06-
22_20:51:13/logs
2022-06-22 20:51:14 DEBUG:Beginning evaluation in plugin
2022-06-22 20:51:14 DEBUG:Now entered evaluate of
ApplicationHTTPPlugin.
2022-06-22 20:51:14 DEBUG:Only using port 8000
2022-06-22 20:51:14 DEBUG:Starting with request: b'GET /
HTTP/1.1\r\nHost:www.youporn.com\r\n\r\n'
2022-06-22 20:51:14
DEBUG:-------------------------------------------
2022-06-22 20:51:14 DEBUG:Running individual:
[HTTPRequest:host:]-
insert{%20:end:value:1}-| \/
2022-06-22 20:51:14 DEBUG: + out action tree triggered:
[HTTPRequest:host:]-insert{%20:end:value:1}-|
2022-06-22 20:51:14 DEBUG:Inserting value: |%20| into
the end of the
variable header_value, 1 times, in the header
Host:www.youporn.com
2022-06-22 20:51:14 DEBUG:Shuffling headers...
2022-06-22 20:51:14 DEBUG:New request string: b'GET /
HTTP/1.1\r\nHost:www.youporn.com \r\n\r\n'
2022-06-22 20:51:14 DEBUG:Connecting to url <ip> with
port 8000
2022-06-22 20:51:14 DEBUG:Response data
HTTP/1.1 200 OK
Date: Thu, 23 Jun 2022 03:51:14 GMT
Server: Apache/2.4.6 (Unix)
Last-Modified: Thu, 23 Jun 2022 03:12:30 GMT
ETag: "2d-5e214d2fe4577"
Accept-Ranges: bytes
Content-Length: 45
Content-Type: text/html

<html><body><h1>It works!</h1></body></html>

2022-06-22 20:51:14
DEBUG:===========================================
2022-06-22 20:51:14 DEBUG:EVADED the censor! Had
response line: HTTP/1.1
200 OK
2022-06-22 20:51:14
DEBUG:===========================================
2022-06-22 20:51:14 DEBUG:Punishing for complexity: 1
2022-06-22 20:51:14 DEBUG:New request is 1 bytes longer
than original.
Punishing -10 fitness.
2022-06-22 20:51:14 DEBUG:Individual
[HTTPRequest:host:]-
insert{%20:end:value:1}-| \/ ran with a fitness of: 316
2022-06-22 20:51:14 INFO:[316]
2022-06-22 20:51:14 INFO:Trial 0: success! (fitness
= 316)
2022-06-22 20:51:14 INFO:Overall 1/1 = 100%
2022-06-22 20:51:14 INFO:Exiting eval-only.
```

In the above output, you can see that the strategy was implemented on the outgoing request: one space (`%20`) was added to the outbound request at the end of the value of the `Host` header (`'GET / HTTP/1.1\r\nHost:www.youporn.com \r\n\r\n'`). You may use `tcpdump` to verify that these bytes are transmitted on the wire.

You can next see that this strategy evaded censorship, and the server was able to properly respond. You can also see that the fitness function correctly detected that it evaded censorship, and awarded a positive fitness value accordingly. Repeat this step for the three strategies we provided, and you can verify that each correctly implements the strategy it purports to on the wire and that those strategies successfully evade censorship.

A.7 Version

Based on the LaTeX template for Artifact Evaluation V20220119.