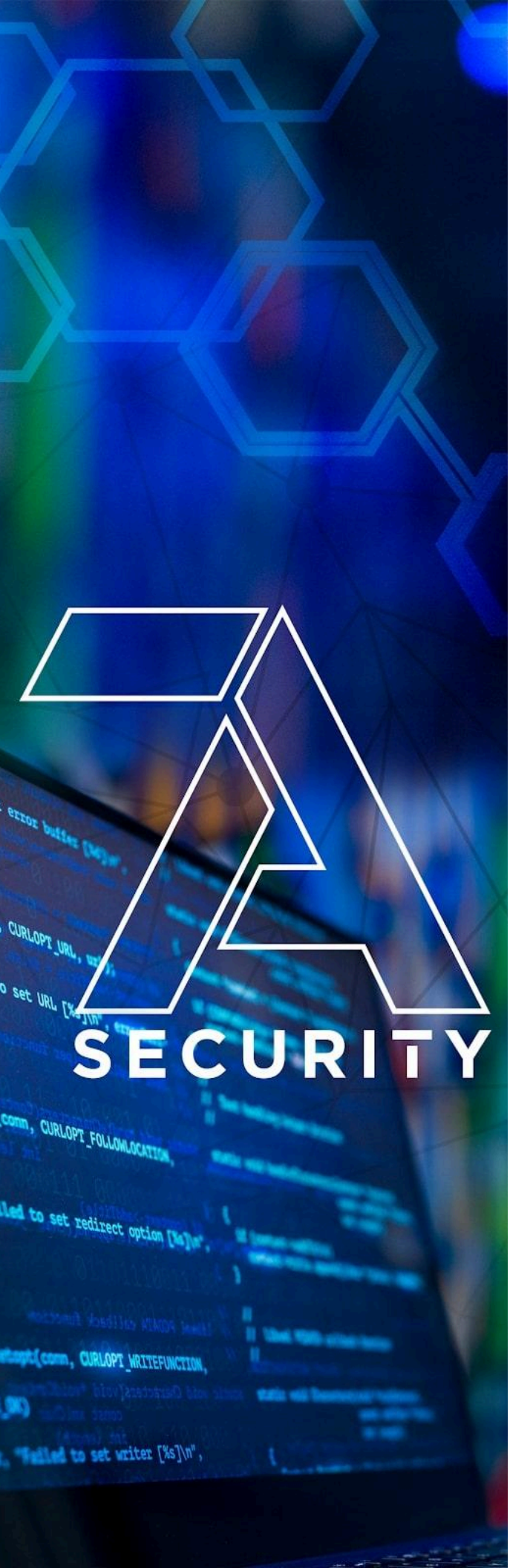


Ground Truth Test Targets:

Website, Servers & Clients
Fuzzing & Test Case Creation
Server Config Audit
Cloud Config Audit
Supply Chain Implementation
Threat Model Documentation
Server & Client Privacy Audit

Pentest Report

Client:
Ground Truth

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Miroslav Štampar, PhD.
- Óscar Martínez, MSc.
- Patrick Ventuzelo, MSc.
- Stefan Nichula, PhD.
- Szymon Grzybowski, MSc.

7ASecurity
Protect Your Site & Apps
From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction

"Disguiser - An Accurate, End-to-End Global Censorship Measurement Framework
The project aims to explore, develop, and deploy a framework that enables end-to-end measurement for accurately and automatically investigating global Internet censorship practices. The key idea is to provide a static payload as ground truth, which can be used to indicate the occurrence of censorship when the static payload has been altered by network devices. Moreover, the deployed end-to-end framework can facilitate extended measurements for investigating more aspects of Internet censorship, for example, pinpointing censor devices' locations and exploring their policies and deployment."

From https://e2ecensor.github.io/

This document outlines the results of a penetration test and *whitebox* security review conducted against the Ground Truth platform. The project was solicited by Ground Truth, funded by the Open Technology Fund (OTF), and executed by 7ASecurity from July until September 2023. The audit team dedicated 57 working days to complete this assignment. Please note that this is the first penetration test for this project. Consequently, identification of new security weaknesses was expected to be easier during this assignment, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration the goal was to review the solution as thoroughly as possible, to ensure researchers using the Ground Truth framework can be provided with the best possible security. This is particularly important, as Ground Truth deals with network traffic potentially tampered by hostile government-sponsored adversaries.

The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, read-only cloud access, SSH server access, documentation and source code. A team of 7 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by July 2023, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Slack channel. The Ground Truth team had a number of difficulties to provide the necessary access and information a number of times, which introduced multiple delays. As an example, access to the Azure infrastructure could not be granted on time and had to be left out of the scope during this iteration as a result. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

This audit split the scope items in the following work packages, which are referenced in the ticket headlines as applicable:

- WP1: Whitebox Tests against Ground Truth Website, Servers and Clients
- WP2: Ground Truth Fuzzing and Fuzzing Test Case Creation
- WP3: Whitebox Tests against Ground Truth Servers, Infrastructure & Configuration via SSH
    - Please note that the server configuration audited was for reference purposes only and is not what users will implement in practice.
    - The Disguiser team put together a *Checklist of Best Practices for Deploying Secure and Reliable Cloud Instances as Backend Servers*[1], based on this engagement.
- WP4: Whitebox Tests against Ground Truth Cloud Infrastructure on AWS & Google Cloud
- WP5: Whitebox Tests against Ground Truth Supply Chain Implementation
- WP6: Ground Truth Lightweight Threat Model documentation
- WP7: Privacy tests against Ground Truth Servers & Clients

The findings of the security audit (WP1-4) can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
7	*13*	*20*

Please note that the analysis of the remaining work packages (WP5-7) is provided separately, in the following sections of this report:

- WP5: Supply Chain Implementation Analysis
- WP6: Ground Truth Lightweight Threat Model
- WP7: Privacy Analysis Findings

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained

[1] https://github.com/e2ecensor/Disguiser_public/blob/main/...20Deployment.md

throughout this test, as well as a summary of the perceived security posture of the Ground Truth framework.

Scope

The following list outlines the items in scope for this project:

- **WP1: Whitebox Tests against Ground Truth Website, Servers and Clients**
  - Source code audit of the Ground Truth server scripts
    - https://github.com/e2ecensor/newDisguiser
    - https://github.com/e2ecensor/Disguiser_public
  - Ground Truth Website hosted on github:
    - URL: https://e2ecensor.github.io/
    - Code: https://github.com/e2ecensor/e2ecensor.github.io
- **WP2: Ground Truth Fuzzing and Fuzzing Test Case Creation**
  - As above
- **WP3: Whitebox Tests against Ground Truth Servers, Infrastructure & Configuration via SSH**
  - SSH access to various servers was provided to 7ASecurity
- **WP4: Whitebox Tests against Ground Truth Cloud Infrastructure on AWS & Google Cloud**
  - Read-only Cloud access was provided to 7ASecurity
  - NOTE: The Azure audit had to be skipped as access could not be granted by the Ground Truth team.
- **WP5: Whitebox Tests against Ground Truth Supply Chain Implementation**
  - As above
- **WP6: Ground Truth Lightweight Threat Model documentation**
  - As above
- **WP7: Privacy tests against Ground Truth Servers & Clients**
  - As above

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *GRT-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

GRT-01-001 WP1/2: Multiple Censorship Spoofing via Error Handling *(Medium)*

While auditing and fuzzing the *newDisguiser* repository, it was found that multiple error handling code snippets are either incorrect or too strict. This leads to false censorship detection. A malicious attacker could leverage this weakness to tamper with Ground Truth censorship statistics, creating false positives during the data gathering process, and hence making Ground Truth collect inaccurate information. These issues can be summarized as follows:

Issue 1: Multiple incorrect handling of BeautifulSoup exceptions

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Analysis/http_server_censorship.py#L72

Affected Code:

```
try:
    vp_title = BeautifulSoup(vp_response, "html.parser").title.string
    local_title = webpage_title_dic[domain]
    if vp_title == local_title and local_title != '':
        data['domain'][domain][url] = "no censorship"
        title = True
    else:
        data['domain'][domain][url] = "detect censorship"
except:
    data['domain'][domain][url] = "detect censorship"
```

Affected Files:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Analysis/http_analysis.py#L69
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Analysis/http_censorship.py#L72
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Analysis/http_suspicious_server.py#L72

https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Analysis/http_server_censorship.py#L72
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Analysis/http_analysis.py#L69

Affected Code:

```
try:
    vp_title = BeautifulSoup(vp_response, "html.parser").title.string
    local_title = webpage_title_dic[domain]
    if vp_title == local_title and local_title != '':
        data['domain'][domain][url] = "no censorship - correct title"
    else:
        data['domain'][domain][url] = "detect censorship - wrong title"
  except:
    data['domain'][domain][url] = "detect censorship - wrong http"
```

The above implementation flaw can be exploited using the following PoC, which triggers an *AssertionError* in *BeautifulSoup*. This will be caught by the *except* clause of the aforementioned code snippets, which leads to a flawed classification of the domain, being incorrectly flagged as censored by Ground Truth:

PoC:

```
from bs4 import BeautifulSoup
html = """
<!DOCTYPE html>
<html>
<head>
    <title>BS4 crash</title>
</head>
<body>
<div>
    <pre>"¥<<![&t&"</pre>
    <h1>My First Heading</h1>
    <p>My first paragraph.</p>
</div>
</body>
</html>
"""
BeautifulSoup(html, 'html.parser')
```

Crash report:

```
Traceback (most recent call last):
  File
"/home/7asecurity/Documents/consulting/groundtruth_audit/replay_crash/bs4_html_parser/bs4_html_parser.py", line 21, in <module>
    BeautifulSoup(html(f.read().decode(encoding='unicode_escape')), 'html.parser')
```

```
  File
"/home/7asecurity/.cache/pypoetry/virtualenvs/groundtruth-audit-W5d_q9LE-py3.10/lib/pyt
hon3.10/site-packages/bs4/__init__.py", line 344, in __init__
    raise ParserRejectedMarkup(
bs4.builder.ParserRejectedMarkup: The markup you provided was rejected by the parser.
Trying a different parser or a different encoding may help.

Original exception(s) from parser:
 AssertionError: expected name token at '<![&t&"</pre>\n      '
```

It is recommended to adequately handle the *BeautifulSoup AssertionError* exception, as well as any other unforeseen exceptions. For parsing exceptions, it is further recommended to reanalyze the webpage using another available HTML parser prior to assigning a censorship classification.

Issue 2: Incorrect censorship detection for CDNs and Anti-bot domains

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/landing_pages.py#L33

Affected Code:

```
def retrieve_landing_page(domain):
    url = 'http://' + domain
    headers = dict()
    headers['User-Agent'] = 'Mozilla/5.0'

    try:
        response = requests.get(url, headers = headers, timeout = 10)
        webpage = response.text
        status_code = response.status_code
    except:
        webpage = 'ERROR'
        status_code = '999'

    landing_page_dic = {'domain': domain, 'status_code': status_code, 'webpage':
webpage}
    return landing_page_dic
```

The use of the *requests* module to fetch data can lead to inaccurate statistics. If the requested server uses *Anti-Bot* or scraper protection such as *Cloudflare* or *Akamai*, the response will be different from the one received by a real user. For example, the HTML title could be different as in this example taken from the *domain_title_dict_2021.txt*[2] file:

[2] https://github.com/e2ecensor/newDisguiser/blob/.../Analysis/domain_title_dict_2021.txt

```
{"003ms.ru": "Attention Required! | Cloudflare"}
```

It is recommended to detect if the requested page is not protected by a *CDN* or *Anti-Bot* prior to censorship classification.

Issue 3: Possible censorship misclassification via uninitialized variables

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/proxyrack_client.py#L102

Affected Code:

```
def get_proxyrack_proxy_info(proxy, finished_countries):

    release_time = 0
    for _ in range(300):

        need_release = False
        proxy_info = proxyrack.get_proxy_info(proxy)

        if proxy_info != None:
            test_sequence = ['dns', 'http', 'sni']
            [...]

    if release_time == 300:
        [...]

    return proxy_info, test_sequence
```

It is recommended to initialize and assign *test_sequence* at the beginning of *get_proxyrack_proxy_info* to prevent an *UnboundLocalError* exception if proxy_info is *None*.

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Analysis/pinpoint_censor_auto.py#L150

Affected Code:

```
    try:
        # tcp handshake
        sock.connect((server, 53))
        port = sock.getsockname()[1]
```

```
        sock.setsockopt(socket.IPPROTO_IP, socket.IP_TTL, struct.pack('I', ttl))
        # send DNS-over-TCP query, receive response
        sock.send(q)
        raw_dns_response = sock.recv(1024)
        is_timeout = False
        addr = get_router_ip(icmp_sock, port)
        # close socket
        # sock.shutdown(socket.SHUT_RDWR)
        # sock.close()

    except socket.timeout:
        raw_dns_response = ''
        is_timeout = True

        addr = get_router_ip(icmp_sock, port)
```

It is recommended to initialize the *port* before calling *get_router_ip* in the handling of *socket.timeout* exception. This will prevent an *UnboundLocalError* exception.

GRT-01-002 WP1/2: Multiple DoS via crafted HTTP Responses *(Medium)*

While fuzzing the *newDisguiser* codebase, it was found that the *process_raw_http_response* function fails to implement adequate exception handling. This led to the discovery of multiple unhandled exceptions that may result in *Denial of Service (DoS)* within multiple scripts such as *pinpoint_censor.py*, *china_client.py*, *proxyrack_client.py*, *vpn_client.py* and *pinpoint_censor_auto.py*. DoS of Ground Truth clients may in turn lead to unsound censorship classification results, as well as general disruption of censorship analysis by researchers. These issues were confirmed as follows:

Issue 1: HTTP response parsing failure when the number of headers is too high

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/pinpoint_censor.py#L164

Affected Code:

```
def process_raw_http_response(raw_http_response, is_timeout):
[...]
        response.begin()
[...]
```

PoC File:
https://7as.es/GroundTruth_bERlScOlr1tj8/PoC/process_raw_http_response_max_head

ers/crash-7cb2cf40c58ef3ccbe91df4b148a325e6544427a.txt

PoC:

```
from targets.newDisguiser.Disguiser.code.pinpoint_censor import
process_raw_http_response

if __name__ == '__main__':
    with
open('./replay_crash/process_raw_http_response_max_headers/crash-7cb2cf40c58ef3ccbe91df
4b148a325e6544427a.txt', 'rb') as f:  # noqa: E501
        process_raw_http_response(f.read(), False)
```

Crash report:

```
Traceback (most recent call last):
  File
"/home/7asecurity/Documents/consulting/groundtruth_audit/replay_crash/process_raw_http_
response_max_headers/process_raw_http_response_max_headers.py", line 5, in <module>
    process_raw_http_response(f.read(), False)
  File
"/home/7asecurity/Documents/consulting/groundtruth_audit/targets/newDisguiser/Disguiser
/code/pinpoint_censor.py", line 192, in process_raw_http_response
    response.begin()
  File "/usr/lib/python3.10/http/client.py", line 337, in begin
    self.headers = self.msg = parse_headers(self.fp)
  File "/usr/lib/python3.10/http/client.py", line 234, in parse_headers
    headers = _read_headers(fp)
  File "/usr/lib/python3.10/http/client.py", line 219, in _read_headers
    raise HTTPException("got more than %d headers" % _MAXHEADERS)
http.client.HTTPException: got more than 100 headers
```

When the number of HTTP headers in the response is more than 100, an *HTTPException* exception is raised and it is recommended to catch it properly. It is also suggested to revisit the *http.client._MAXHEADERS* parameter configuration to a higher value, such as 1000 to prevent potential false positives.

Issue 2: HTTP response parsing failure when the protocol line is malformed

Affected File:

https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/pinpoint_censor.py#L164

Affected Code:

```
def process_raw_http_response(raw_http_response, is_timeout):
      [...]
              response.begin()
```

```
            http_result['text'] = response.read(len(raw_http_response)).decode()
            #http_result['url'] = raw_http_response.url
            http_result['status_code'] = response.status
            http_result['headers'] = dict(response.getheaders())
    [...]
```

PoC File:
https://7as.es/GroundTruth_bERlScOlr1tj8/PoC/process_raw_http_response_unknown_protocol/crash-b5725dd39bffc82d6434c39863ded9e7b11732b6.txt

PoC:

```
from targets.newDisguiser.Disguiser.code.pinpoint_censor import
process_raw_http_response

if __name__ == '__main__':
    with
open('./replay_crash/process_raw_http_response_unknown_protocol/crash-b5725dd39bffc82d6
434c39863ded9e7b11732b6.txt', 'rb') as f:  # noqa: E501
        process_raw_http_response(f.read(), False)
```

Crash report:

```
File
"/home/7asecurity/Documents/consulting/groundtruth_audit/replay_crash/process_raw_http_
response_unknown_protocol/process_raw_http_response_unknown_protocol.py", line 5, in
<module>
    process_raw_http_response(f.read(), False)
  File
"/home/7asecurity/Documents/consulting/groundtruth_audit/targets/newDisguiser/Disguiser
/code/pinpoint_censor.py", line 192, in process_raw_http_response
    response.begin()
  File "/usr/lib/python3.10/http/client.py", line 335, in begin
    raise UnknownProtocol(version)
http.client.UnknownProtocol: HTTP/1.1
```

The protocol line may be malformed and raise an *UnknownProtocol* exception. It is recommended to catch the http client exception properly when calling *response.begin()* to resolve this issue.

Issue 3: HTTP response parsing failure on UnicodeDecodeError

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/pinpoint_censor.py#L164

Affected Code:

```
def process_raw_http_response(raw_http_response, is_timeout):
[...]
        http_result['text'] = response.read(len(raw_http_response)).decode()
[...]
```

PoC File:
https://7as.es/GroundTruth_bERlScOlr1tj8/PoC/process_raw_http_response_unicode_decode/crash-fcd9af74a499d1c451ee733da0e3ab0171fe7842.txt

PoC:
```
from targets.newDisguiser.Disguiser.code.pinpoint_censor import
process_raw_http_response

if __name__ == '__main__':
    with
open('./replay_crash/process_raw_http_response_unicode_decode/crash-fcd9af74a499d1c451e
e733da0e3ab0171fe7842.txt', 'rb') as f:  # noqa: E501
        process_raw_http_response(f.read(), False)
```

Crash report:
```
Traceback (most recent call last):
  File
"/home/7asecurity/Documents/consulting/groundtruth_audit/replay_crash/process_raw_http_
response_unicode_decode/process_raw_http_response_unicode_decode.py", line 5, in
<module>
    process_raw_http_response(f.read(), False)
  File
"/home/7asecurity/Documents/consulting/groundtruth_audit/targets/newDisguiser/Disguiser
/code/pinpoint_censor.py", line 193, in process_raw_http_response
    http_result['text'] = response.read(len(raw_http_response)).decode()
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xba in position 0: invalid start
byte
```

The default encoding in *response.read* used to decode the *raw_http_response* is *utf-8*. When decoding a non-utf-8 response, a *UnicodeDecodeError* will be raised since the default encoding is not compatible with the one used by the server. It is recommended to catch the *UnicodeDecodeError* exception and advise accordingly regarding the censorship classification.

GRT-01-003 WP1/2: Ground Truth RCE via Crafted Domain File (*Critical*)

While auditing the *newDisguiser* codebase, it was found that the *read_domain* function fails to properly sanitize the content of the provided file. This led to the discovery of a *Remote Code Execution (RCE)* vulnerability. A malicious attacker, able to entice a Ground Truth researcher to use a crafted domain text file, could leverage this weakness to run arbitrary code or commands in the system the Ground Truth client script is being run from. This issue can be validated reviewing the following code snippet:

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Analysis/proxyrack.py#L53

Affected Code:

```
def read_domain(file):
        """
        read domain info from plain file
        args:
        file: str. full file path or relevant file path
        returns:
        dict
        raises:
        None
        """
        with open(file, "r") as f:
        data = []
        for r in f.readlines():
                # skip empty lines
                if r == "\n":
                continue
                r = r.replace("false", "False")
                r = r.replace("true", "True")
                try:
                # convert string to dictionary
                data.append(eval(r))
                except Exception as e:
                print(e)
        return data
```

The eval function used to parse the dataset (.txt) in the script can lead to security vulnerabilities. An attacker can craft and inject malicious code into this JSON-looking file in order to achieve remote code execution (RCE) when this function will process the file.

It is recommended to use the json package to parse JSON data, as it eliminates this

attack vector.

GRT-01-004 WP1/2: Ground Truth RCEs via Crafted Config Files *(Critical)*

While auditing the *newDisguiser* codebase, it was found that multiple Ground Truth code paths fail to sanitize the extracted *username, password* and *IP* from configuration files, prior to providing them as arguments to cURL commands. This may lead to Remote Code Execution (RCE) as the cURL command gets executed. A malicious attacker, able to entice a Ground Truth researcher to use a crafted configuration file, could leverage this weakness to run arbitrary code or commands in the system the Ground Truth client script is being run from. This can be confirmed reviewing the following code snippets:

Issue 1: RCEs in *get_curl_cmd* functions

Affected Files:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/proxyrack.py#L16
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Analysis/proxyrack.py#L67

Affected Code:
```
def get_curl_cmd(proxy, url):
    proxy_address, proxy_port, username, password = unpack_proxy_args(proxy)
    return 'curl -m 10 -s -x ' + proxy_address + ':' + str(proxy_port) + ' -U ' +
username + ':' + password + ' ' + url

def get_proxy_stats(proxy, timeout = 5):
    url = 'http://api.proxyrack.net/stats'
    curl_cmd = get_curl_cmd(proxy, url)
    try:
        stats = os.popen(curl_cmd).read()
```

Issue 2: RCE in Ripe Atlas integration

A similar case of command execution was identified in the Ripe Atlas integration in the *ripe_atlas_client.py* that processes a list of IP addresses from a configuration text file without any prior sanitization on the provided input.

Affected File:
https://github.com/e2ecensor/Disguiser_public/blob/6625710d013aeb78ac7588bbf0739e9ea4e9843b/code/ripe_atlas_client.py#L280

Affected Code:

```
input_file = '../results/ripe_atlas/ripe_atlas_iran_results.txt'
with open(input_file, 'r') as f:
    entries = f.read().strip().split('\n')
    for entry in entries:
        temp_dic = json.loads(entry)
        ip = temp_dic['probe']
        if ip not in ip_info_dic.keys():
            while True:
                try:
                    response = os.popen('curl -m 10 -s http://ip-api.com/json/' +
ip).read()
```

It is recommended to adequately escape user-controlled parameters to prevent command injection. For enhanced protection, the current cURL and os.popen approach ought to be replaced with a safer approach, such as using the *requests* library[3] instead. The proposed solution will eliminate this attack vector.

GRT-01-016 WP1: Censorship Detection Bypass via Hash Collision *(High)*

During the audit of the *newDisguiser* codebase, it was found that attackers may evade censorship detection by exploiting a hash collision. Malicious adversaries could accomplish this by utilizing an identical combination of HTML tags (DOM Tree), present in both a maliciously crafted HTTP page and an existing website within the trusted portfolio. This issue was confirmed as follows:

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/http_manual_validation.py#L30

Affected Code:

```
def get_webpages():
    [...]
    for subdir in subdirs:
        if subdir.split('/')[-1].startswith('20'):
            [...]
            with open(subdir + '/' + 'http_manual_case.txt') as f:
                [...]
                for vp in vps:
                    [...]
                    for domain in vps[vp]['domain']:
                        [...]
                        soup = BeautifulSoup(webpage, 'html.parser')
```

[3] https://pypi.org/project/requests/

```
                    document = soup.find_all()
                    tags = [x.name for x in document]
                    tags_string = ' '.join(tags)
                    webpage_hash = hashlib.md5(tags_string.encode()).hexdigest()
    [...]
```

As can be seen in the code snippet above, the *webpage_dic* dictionary located in the *http_manual_validation.py* file will be populated with MD5 hashes of the HTML tags of the pages. Consequently, it is possible to bypass detection by crafting web pages that will result in the same MD5 *webpage_hash* value, by simply using an identical HTML structure as a trusted web page:

PoC: MD5 html.parser hash collision

```
import hashlib

from bs4 import BeautifulSoup

HTML_PAGE_1 = """
<!DOCTYPE html>
<html>
        <head>
        <title>Page Title</title>
        </head>
        <body>
        <div>
                <h1>My First Heading</h1>
                <p>My first paragraph.</p>
        </div>
        </body>
</html>
"""

HTML_PAGE_2 = """
<!DOCTYPE html>
<html>
        <head>
        <title>Page Title</title>
        </head>
        <body>
        <div>
                <h1>My Second Heading</h1>
                <p>My second paragraph.</p>
        </div>
        </body>
</html>
"""
```

```
def get_webpage_hash(content: str) -> str:
        soup = BeautifulSoup(content, 'html.parser')
        return        hashlib.md5('        '.join([x.name        for        x        in
soup.find_all()]).encode()).hexdigest()

if __name__ == '__main__':
        print(f"Hash of HTML_PAGE_1: {get_webpage_hash(HTML_PAGE_1)}")
        print(f"Hash of HTML_PAGE_2: {get_webpage_hash(HTML_PAGE_2)}")
```

Command:

```
python3 poc.py
```

Output:

```
Hash of HTML_PAGE_1: dcaf505af9b4d4a90a709e751f4eecae
Hash of HTML_PAGE_2: dcaf505af9b4d4a90a709e751f4eecae
```

Resolving this issue is particularly challenging. On one hand, if only HTML tags are hashed, attackers can simply clone the page structure as described in this issue, plus any censored text contents will be completely undetected. On the other hand, if Ground Truth switches to hashing entire web pages any small text change or timestamp in the HTML will result in a different hash. A more advanced approach is therefore required to work around the aforementioned problems. This should include usage of *Fuzzy hashing*[4] and *Rolling hash*[5] algorithms, with particular care to ensure no text has been censored in the pertinent HTML page. Moreover, if the complexity of the censored text comparison allows it, a rolling hash algorithm such as *Rabin-Karp*[6] can be applied in order to identify the existence of certain subscripts in the entire webpage. This algorithm can be leveraged to determine whether key components of the webpage are often prone to censorship as the user controls the content of the experimental data.

[4] https://en.wikipedia.org/wiki/Fuzzy_hashing
[5] https://en.wikipedia.org/wiki/Rolling_hash
[6] https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm

GRT-01-017 WP1: Censorship Misclassification via Proxyrack Logic Flaw *(High)*

While auditing the *newDisguiser* codebase, it was found that proxyrack is always released. Specifically, the *need_release* variable will always be set to **True**, which adds unnecessary complexity to the code and increases false positives when checking for censorship using different sticky proxy addresses when rotating different countries. This can be confirmed by analyzing the following code snippet:

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/proxyrack_client.py#L76

Affected Code:

```
def get_proxyrack_proxy_info(proxy, finished_countries):

        release_time = 0
        for _ in range(300):
                need_release = False
                proxy_info = proxyrack.get_proxy_info(proxy)

                if proxy_info != None:
                        test_sequence = ['dns', 'http', 'sni']
                        test_sequence = list(filter(lambda x:
finished_countries[x].get(proxy_info['country'], 0) < max_per_country, test_sequence))
                        if len(test_sequence) != 0:
                                break
                        else:
                                need_release = True
                else:
                        need_release = True

[...]
```

It is recommended to simplify the function by removing the *need_release* variable and always calling *proxyrack.release_exit_node(proxy).*

GRT-01-019 WP1: Ground Truth RCEs & Spoofing via clear-text HTTP *(**Critical**)*

It was found that the RCE described in GRT-01-004, can not only be triggered via crafted configuration files, but also through modification of clear-text HTTP communications. This issue is particularly concerning given the nature of Ground Truth, which deals with network traffic potentially tampered by government-sponsored adversaries. In a worst-case scenario, a malicious Man-In-The-Middle (MitM) attacker, with the ability to intercept and modify clear-text network communications (i.e. via BGP hijacking[7], DNS rebinding[8], ISP MitM, public Wi-Fi without guest isolation, etc.), could leverage this weakness to run arbitrary commands on the operating system of the researcher running Ground Truth scripts such as *proxyrack.py*. Please note a malicious attacker may additionally exploit this weakness to spoof censorship results. This issue can be trivially confirmed by looking at the following code snippets:

Affected Files: Proxy Configuration Scripts

https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d22dade26/Disguiser/code/proxyrack.py#L50
https://github.com/e2ecensor/newDisguiser/blob/7b1ba206d22c17a33c7c042159f47f4487d8155f/Analysis/proxyrack.py#L69
https://github.com/e2ecensor/Disguiser_public/blob/8ec0f0ae76ecb894e68e2b810fa98ba007b2e8e7/code/proxyrack.py#L19

Affected Code: Proxy set up

```
def get_proxy_info(proxy, timeout = 5):
        url = 'http://ip-api.com/json'
        curl_cmd = get_curl_cmd(proxy, url)
[...]
def get_curl_cmd(proxy, url):
    proxy_address, proxy_port, username, password = unpack_proxy_args(proxy)
    return 'curl -m 10 -s -x ' + proxy_address + ':' + str(proxy_port) + ' -U ' +
username + ':' + password + ' ' + url

def get_proxy_stats(proxy, timeout = 5):
    url = 'http://api.proxyrack.net/stats'
    curl_cmd = get_curl_cmd(proxy, url)
    try:
[...]
def release_exit_node(proxy, timeout = 5):
    url = 'http://api.proxyrack.net/release'
    curl_cmd = get_curl_cmd(proxy, url)
```

[7] https://en.wikipedia.org/wiki/BGP_hijacking
[8] https://en.wikipedia.org/wiki/DNS_rebinding

```
    flag = False
    try:
        response = os.popen(curl_cmd).read()
```

Affected Files: VPN Configuration Setup
https://github.com/e2ecensor/newDisguiser/blob/7b1ba206d22c17a33c7c042159f47f4487d8155f/Disguiser/code/vpn_client.py#L88
https://github.com/e2ecensor/newDisguiser/blob/7b1ba206d22c17a33c7c042159f47f4487d8155f/Disguiser/code/china_client.py#L64
https://github.com/e2ecensor/Disguiser_public/blob/8ec0f0ae76ecb894e68e2b810fa98ba007b2e8e7/code/vpn_client.py#L88
https://github.com/e2ecensor/Disguiser_public/blob/8ec0f0ae76ecb894e68e2b810fa98ba007b2e8e7/analysis/china_client.py#L64

Affected Code:

```
def get_vpn_info():
    url = 'http://ip-api.com/json'
    curl_cmd = 'curl -m 10 -s ' + url

    try:
        response = os.popen(curl_cmd).read()
        response = json.loads(response)
```

Affected Files: Ripe Atlas Configuration
https://github.com/e2ecensor/newDisguiser/blob/7b1ba206d22c17a33c7c042159f47f4487d8155f/Disguiser/code/ripe_atlas_client.py#L280
https://github.com/e2ecensor/Disguiser_public/blob/8ec0f0ae76ecb894e68e2b810fa98ba007b2e8e7/code/ripe_atlas_client.py#L280

Affected Code:

```
if ip not in ip_info_dic.keys():
    while True:
        try:
            response = os.popen('curl -m 10 -s http://ip-api.com/json/' + ip).read()
            probe_info = json.loads(response)
            ip_info_dic[ip] = probe_info
            time.sleep(2)
            break
```

It is recommended to extrapolate the mitigation guidance offered under GRT-01-004 to resolve this issue. Once that is done, protocols using clear-text traffic should be avoided as much as possible to eliminate this attack vector. For example, while the TLS version of https://ip-api.com/json is only available to paid subscribers, Ground Truth could

prompt researchers to enter their subscription key when they run the script and/or at least warn them that the traffic may be tampered with over clear-text HTTP. Furthermore, in situations where clear-text protocols are required for testing data, exposure to attacks should be limited by implementing sandboxing on a pivoting service designed to route inbound and outbound traffic. After that, for communications that use TLS, pinning may be considered to further protect the integrity of network communications against high-profile adversaries able to craft valid TLS certificates trusted by the operating system. For additional guidance about Pinning, please see the *OWASP Pinning Cheat Sheet*[9].

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

GRT-01-005 WP4: AWS Leaks via Unencrypted EBS Volumes & Snapshots *(Low)*

A number of Ground Truth volumes across all used regions were found to be stored without prior encryption at rest. In case sensitive data is stored on these unencrypted volumes, this may not only leak data, but also violate compliance with multiple frameworks. It should be noted that, when the encryption option is disabled, potential flaws in the AWS implementation might allow unauthorized attackers to access the volume. This might occur through an AWS access control flaw, as well as physical attacks where hard/SSD drives are replaced in the data center. Hence, encryption provides an additional security layer for such scenarios and minimizes potential unintentional data disclosure.

Affected Resources:
AWS Account 781265170134

This issue can be confirmed navigating to the *EC2 Volumes* or *Snapshots* areas on the *AWS Management Console* in the *us-east-1* region:

URL:

[9] https://cheatsheetseries.owasp.org/cheatsheets/Pinning_Cheat_Sheet.html

https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Volumes:v=3;encrypted=false

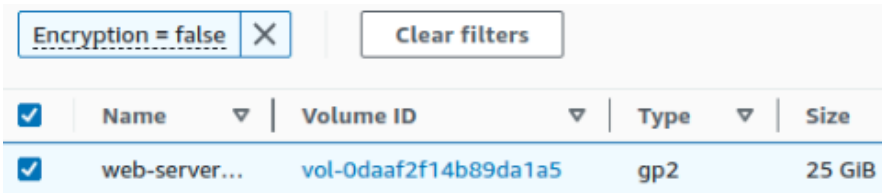


Fig.: Unencrypted Volume

It is recommended to enable encryption, ideally by default, for all newly created volumes[10].

GRT-01-006 WP4: AWS Weaknesses in Vuln Management Processes *(Medium)*

During the configuration audit of the AWS production account, it was discovered that multiple AWS security-relevant services are not configured correctly. Failure to leverage these services can leave the infrastructure open to attacks due to insufficient hardening.

Affected Resources:
AWS Account 781265170134

Please note that, as most of the AWS services are region-based, it is important to determine which regions are used first, to focus the analysis on the regions that are actually in use. Regions with defined resources in the analyzed environment are: *us-west-1, us-east-1, sa-east-1, me-south-1, eu-west-3, eu-west-2, af-south-1*

Issue 1: *Security Hub* is not enabled

Security Hub[11] is a region-based service that provides a comprehensive view of security issues from regions where it is enabled. The following command describes the status of *Security Hub* for the regions in use:

Command:

```
aws securityhub describe-hub
```

[10] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html
[11] https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-get-started.html

Output:

```
An error occurred (InvalidAccessException) when calling the DescribeHub operation:
Account 781265170134 is not subscribed to AWS Security Hub
```

Issue 2: Guard Duty is not enabled

The following command describes the status of *Guard Duty*[12] for various regions, which confirms *Guard Duty* is not enabled.

Command:

```
aws guardduty list-detectors
```

Output:

```
{ "DetectorIds": []}
[...]
```

Issue 3: AWS Config is not enabled

AWS Config[13] is a service that maintains the configuration history for AWS resources and evaluates best practices. The following command can be used to confirm *AWS Config* is not enabled on the used region:

Command:

```
aws configservice get-status
```

Output:

```
Configuration Recorders:
Delivery Channels:
```

It is recommended to implement as many AWS Security related services as possible. This should include tools like *Security Hub*[14], *Config*[15], *Guard Duty*[16], *Macie*[17] and *Inspector*[18]. After this, the infrastructure team should ensure that all relevant services, and equivalent products, are enabled for the whole environment in all used regions.

[12] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_settingup.html
[13] https://aws.amazon.com/blogs/mt/aws-config-best-practices/
[14] https://aws.amazon.com/security-hub/
[15] https://docs.aws.amazon.com/config/latest/developerguide/security-best-practices.html
[16] https://docs.aws.amazon.com/guardduty/latest/ug/what-is-guardduty.html
[17] https://aws.amazon.com/macie/
[18] https://docs.aws.amazon.com/inspector/v1/userguide/inspector_introduction.html

Furthermore, any reported issues should be regularly reviewed and remediated. This should ideally be accomplished by leveraging an *infrastructure-as-code* approach such as *Terraform*[19], which would significantly simplify applying the same settings across all AWS accounts. Please note that cloud-native security tools are not perfect, however they provide a solid baseline for each environment. Special consideration should be given to *Security Hub* and *Config*, as they allow to streamline and discover common misconfigurations.

GRT-01-007 WP4: Possible AWS Takeover via IAM Root Account Use *(High)*

It was found that the analyzed environment uses only the main AWS root account for actions that could be performed with more restricted accounts. AWS root accounts are the main and most privileged accounts in the AWS environment. Using a root account, either via the API or interactively via the AWS Web Console, unnecessarily increases the likelihood of unauthorized access. In certain cases, this may also weaken the security policy, as commonly MFA is enabled only for Web Console access and is disabled for the API.

Affected Resources:
AWS Account 781265170134

The following example illustrates how to identify root account activity within last 90 days:

Example: Confirm recent root user activity

1. Navigate to the *global EC2* view, on the *AWS Management Console*:
   URL:
   https://us-east-1.console.aws.amazon.com/cloudtrail/home?region=us-east-1#/events?Username=root
2. Adjust filters to make sure the name attribute is set to *root* and select an adequate timespan.
3. Review the logs from different regions or adjust filters in any external logging solution integrated with the environment to review all activities across all regions.

Result:
Multiple actions performed by the root account were logged.

[19] https://www.terraform.io/use-cases/infrastructure-as-code

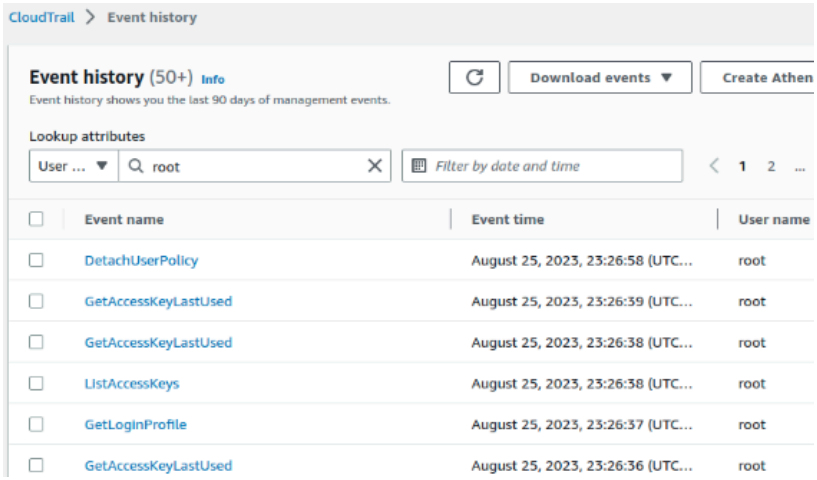


Fig.: Recent root account actions in us-east-1

It is recommended to protect AWS root accounts. This should be accomplished utilizing a strong password and MFA, ideally a hardware-based MFA mechanism. These accounts should only be used occasionally, instead personal accounts for daily operations[20] should be created and configured to only have the absolute minimum permissions necessary to perform their function.

GRT-01-008 WP4: Insufficient AWS Logging & Monitoring *(High)*

It was found that AWS *CloudTrail*[21] is not enabled for all regions. This tool records all activities in an AWS account as events. Without adequate logging, it may be impossible to monitor malicious activities, or use integrated tools that analyze *CloudTrail* for anomalies, all of which may be critical in the event of a security breach.

Affected Resources:
AWS Account 781265170134

Regions with defined resources in the analyzed environment are: *us-west-1, us-east-1, sa-east-1, me-south-1, eu-west-3, eu-west-2, af-south-1* and all were found to be affected.

Issue 1: *CloudTrail* is not enabled

The following command reveals there are no trails defined:

[20] https://aws.amazon.com/iam/identity-center/
[21] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html

Command:

```
aws cloudtrail list-trails
```

Output:

```
{ "Trails": [] }
```

Issue 2: No *VPC flow logs* defined

No VPC flow logs were found to be defined. At a minimum, these should be listed for the VPCs with the main workloads (ECS). This can be confirmed by reviewing the VPC flow logs like so:

1. Open the *AWS Management Console*
2. Navigate to the *VPC Settings* and select a VPC to check.
   PoC URL:
   https://us-east-1.console.aws.amazon.com/vpc/home?region=us-east-1#VpcDetails:VpcId=vpc-0f19e3ec3e547f2a8
3. Review the *Flow Logs* tab.

The following command confirms there are no flow logs defined in the regions for the AWS accounts provided during this assignment:

Command:

```
aws ec2 describe-flow-logs
```

Output:

```
{ "FlowLogs": [] }
```

It is recommended to enable *CloudTrail* for all regions, and ensure logs are automatically archived in encrypted S3 buckets that belong to a separate AWS account. By default *CloudTrail* stores only the last 90 days of activity in AWS, thus archiving is crucial for potential forensic investigations in case of a breach. Additionally, logs from virtual machines should be considered to be integrated with a centralized logging system for better coverage.

In general, all logging and monitoring settings should be adjusted depending on the threat model, compliance requirements and volume of generated data. Excessively verbose logs may increase the overall infrastructure cost significantly, however, lack of appropriate logging and monitoring decreases the chances of successful threat detection and analysis in case of a breach. It is advised to review and improve the logging and

monitoring configuration in the context of a potential incident response case rather than just regular daily operations of the infrastructure[22].

GRT-01-009 WP4: Lack of AWS/GCP Infrastructure Automation *(Info)*

The Ground Truth environment, despite being a multi-cloud solution, fails to leverage infrastructure as code to create and manage the supported cloud configurations. All analyzed cloud environments were found to be created manually. Hence, they are prone to human errors and inconsistencies, which may expose the environment to unnecessary threats. Environments without adequate automation cannot be easily deployed in a repeatable fashion and are difficult to manage over time.

Affected Resources:
AWS Account 781265170134
GCP Project ID 117546701090 (operating-bolt-366020)

It is recommended to review, research and employ all or some of the following solutions:

- Terraform, or similar solutions, to implement an infrastructure as code approach in a multi-cloud environment.
- Robust configuration to automate deployments via Github Actions.
- Github Secrets, HashiCorp Vault, or similar, for secret management, which should be exercised during the deployment process.
- Ansible or similar solutions to automate the provisioning of virtual machines.

Additionally, it is advised to consistently bind all cloud accounts to a single management email account for billing and management purposes instead of using personal emails.

[22] https://docs.aws.amazon.com/whitepapers/.../aws-security-incident-response...html

GRT-01-010 WP3: Usage of Unsupported Ubuntu Version *(Low)*

During whitebox testing against Ground Truth servers over SSH, the backed control servers with IP address 35.180.190.69 and 20.115.40.63 were found to have *Ubuntu 18.04.6 LTS* installed. That *Ubuntu* version is end-of-life and no longer receives updates[23]. Therefore newly discovered vulnerabilities or security issues cannot be fixed. While no public exploit was found for that version at the time of the assessment, this is still a bad practice that could result in unwanted security vulnerabilities and highlights room for improvement in the current software patching processes. This issue can be trivially confirmed with the following commands:

Command:
```
lsb_release -a
```

Output:
```
Description:    Ubuntu 18.04.6 LTS
```

Command:
```
pro status
```

Output:
```
This machine is not attached to an Ubuntu Pro subscription.
```

It is recommended to upgrade the backed control server to a supported *Ubuntu* version, such as *20.04 LTS* or *22.04 LTS*.

GRT-01-011 WP4: Unrestricted Inbound Traffic on GCP *(Medium)*

It was discovered that the Google Cloud Platform (GCP) firewall rules fail to restrict access to virtual machines. This weakness appears to be due to VPC usage, which creates insecure firewall rules by default. This implies that services launched by administrators, which listen on a network interface, will be immediately exposed to attacks from the Internet. For example, malicious adversaries that constantly scan the Internet for easy targets might be able to exploit misconfigurations in the exposed services. This issue was confirmed as follows:

Affected Resources:
GCP Project ID 117546701090 (operating-bolt-366020)

[23] https://ubuntu.com/about/release-cycle

Issue: All ports are exposed to the Internet

The following command reveals the firewall rule that allows all connections on all ports from the Internet:

Command:

```
gcloud compute firewall-rules list
```

Output:

```
default-allow-http
 INGRESS 1000 tcp:80
default-allow-https
 INGRESS 1000 tcp:443
default-allow-icmp
 INGRESS 65534 icmp
default-allow-internal
 INGRESS 65534 tcp:0-65535,udp:0-65535,icmp
default-allow-rdp
 INGRESS 65534 tcp:3389
default-allow-ssh
 INGRESS 65534 tcp:22
```

The following command can be used to scan a sample IP address belonging to a virtual machine from the internet.

Command:

```
nmap -Pn --top-ports 1000 --open 34.155.45.233
```

Output:

```
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 1.89 seconds
```

It is recommended to remove default VPC as multiple insecure firewall rules are defined automatically[24] when a default VPC is in use. It is further suggested to restrict traffic to ports that have to be exposed to the Internet. In case of management access to the virtual machines, either SSH should be open to limited IP addresses, or OS Login with

[24] https://cloud.google.com/firewall/docs/firewalls#more_rules_default_vpc

multi-factor authentication should be used. For additional mitigation guidance, please see the *CIS Google Cloud Computing Platform Benchmark*[25].

GRT-01-012 WP4: Insufficient GCP Logging and Monitoring *(Low)*

It was found that the Ground Truth GCP environment lacks adequate logging and monitoring, which is crucial for compromise detection. Currently, only basic logs from the environment are collected, using a default retention policy, and no agents collecting logs inside virtual machines are installed. Specifically, none of the virtual machines in the environment have Ops Agent installed. This was confirmed as follows:

Affected Resources:
GCP Project ID 117546701090 (operating-bolt-366020)

Example: Confirming the lack of an installed monitoring agent

1. Navigate to the *Monitoring* view, on the *Google Cloud Platform Web Console*:
   URL:
   https://console.cloud.google.com/monitoring/...
2. Review the *Agent* column.
3. Alternatively navigate to *Monitoring* > *Overview* and verify the status of *VMs with ops agent*.

Result:

	Name	Agent	Active alerts	System events
☐	australia-server2	Not detected	0	0
☐	control-server	Not detected	0	0
☐	paris-server	Not detected	0	0
☐	sao-paulo-server	Not detected	0	0

Fig.: Agents are not installed on any virtual machine

It is advised to consider employing a centralized logging solution, which may be implemented either using cloud-native services[26] or via third-party software like *Splunk*[27],

[25] https://www.cisecurity.org/benchmark/google_cloud_computing_platform
[26] https://cloud.google.com/stackdriver/docs/solutions/agents/ops-agent/third-party/apache
[27] https://dev.splunk.com/observability/docs/integrations/gcp_integration_overview/

ElasticStack[28], or similar. As the environment consists mainly of virtual machines, it is recommended to forward all collected logs (from *auditd, apache*, *etc.*) to a centralized logging solution. Additionally, depending on business requirements, it might be necessary to adjust the default retention policies[29].

GRT-01-013 WP4: Potential GCP PrivEsc via Privileged Service Account *(Low)*

It was uncovered that the Ground Truth GCP environment employs virtual machines that use a default Compute Engine service account, which has the Editor role on the project[30]. Please note that the exploitability of this issue from the internet is limited, this is due to the restrictions in the Cloud API access scope for the affected role.

PoC Steps:

These example steps confirm that the default editor role is attached to a virtual machine:

Step 1: Navigate to a sample virtual machine

From the *Compute Engine* > *VM instances* view, on the *Google Cloud Platform Web Console*:

URL:
https://console.cloud.google.com/compute/instancesDetail/zones/australia-southeast1-b/instances/australia-server2?project=operating-bolt-366020

Step 2: Review the *API* column on the *identity management* section

Result:

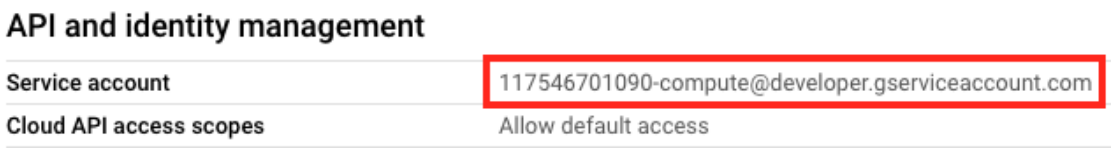


Fig.: Default Compute Engine Service Account

Step 3: Confirm the default service accounts

From the *IAM and admin* section:

[28] https://docs.elastic.co/integrations/gcp
[29] https://cloud.google.com/logging/quotas#logs_retention_periods
[30] https://cloud.google.com/compute/docs/access/service-accounts#default_service_account

URL:
https://console.cloud.google.com/iam-admin/iam?referrer=search&project=operating-bolt-366020

Result:

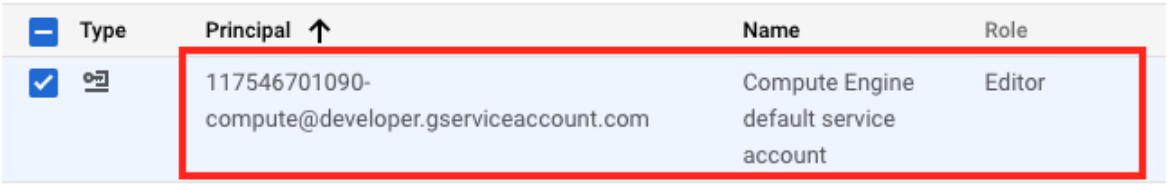


Fig.: Editor role attached to a Default Compute Engine Service Account

It is recommended to remove the default service account and create a custom restricted service account to follow the least privilege principle.

GRT-01-014 WP3: Possible root Access via Passwordless *sudo* *(Low)*

During whitebox testing against Ground Truth servers over SSH, the backed control server with IP address 20.115.40.63 was found to have a passwordless *sudo* implementation for the *testing1* user. Leaving passwordless *sudo* on any server presents a security risk[31] and should be avoided. The following quote from the *StackExchange* thread "*How secure is NOPASSWD in passwordless sudo mode?*"[32] summarizes this issue:

"NOPASSWD doesn't have a major impact on security.[...] Nonetheless, requiring the password does raise the bar for the attacker. In many cases, protection against unsophisticated attackers is useful, particularly in unattended-workstation scenarios where the attack is often one of opportunity and the attacker may not know how to find and configure discreet malware at short notice."

This issue can be trivially confirmed with the following command:

Command:
```
sudo -l
```

Output:
```
User testing1 may run the following commands on Viginia-server:
```

[31] https://attack.mitre.org/techniques/T1548/003/
[32] https://security.stackexchange.com/a/45728

```
(ALL) NOPASSWD: ALL
```

It is recommended to require a password when running the sudo command to resolve this issue.

GRT-01-015 WP3: Usage of Vulnerable Outdated Software *(Low)*

During whitebox testing against Ground Truth servers over SSH, the backed control servers with IP address 35.180.190.69 and 20.115.40.63 were found to use a number of outdated software components with known vulnerabilities. While no public exploits were found at the time of evaluation, this is still a bad practice that could result in unwanted security vulnerabilities and highlights room for improvement in the current software patching processes. The following table summarizes the vulnerabilities identified in the installed software:

Software	Vulnerabilities
busybox-initramfs_1:1.27.2-2ubuntu3.4 *busybox-static_1:1.27.2-2ubuntu3.4*	BusyBox incorrectly handled certain malformed gzip archives and did not properly validate user input when performing certain arithmetic operations.[33]
libpython2.7_2.7.17-1~18.04ubuntu1.11 *libpython2.7-minimal_2.7.17-1~18.04ubuntu1.11* *libpython2.7-stdlib_2.7.17-1~18.04ubuntu1.11* *python2.7_2.7.17-1~18.04ubuntu1.11* *python2.7-minimal_2.7.17-1~18.04ubuntu1.11*	Python could be made to crash or leak sensitive information if it received specially crafted input.[34] Python could be made to bypass blocklisting methods if a specially crafted URL was provided.[35]
python3-requests_2.18.4-2ubuntu0.1	Requests could be made to expose sensitive information over the network.[36]
openssh-client_1:7.6p1-4ubuntu0.7 *openssh-server_1:7.6p1-4ubuntu0.7* *openssh-sftp-server_1:7.6p1-4ubuntu0.7*	OpenSSH could be made to run programs as your login when using ssh-agent forwarding.[37]

[33] https://ubuntu.com/security/notices/USN-6335-1
[34] https://ubuntu.com/security/notices/USN-6354-1
[35] https://ubuntu.com/security/notices/USN-6139-1
[36] https://ubuntu.com/security/notices/USN-6155-2
[37] https://ubuntu.com/security/notices/USN-6242-2

open-iscsi_2.0.874-5ubuntu2.11	Open-iSCSI incorrectly handled certain checksums for IP packets, certain parsing TCP MSS options, and certain TCP data.[38]
vim_2:8.0.1453-1ubuntu1.13 *vim-common_2:8.0.1453-1ubuntu1.13* *vim-runtime_2:8.0.1453-1ubuntu1.13* *vim-tiny_2:8.0.1453-1ubuntu1.13* *xxd_2:8.0.1453-1ubuntu1.13*	Vim was using uninitialized memory when fuzzy matching, which could lead to invalid memory access. Vim was not properly performing bounds checks when processing register contents, which could lead to a NULL pointer dereference. Vim was not properly limiting the length of substitution expression strings, which could lead to excessive memory consumption. Etc.[39][40][41]
libcap2_1:2.25-1.2 *libcap2-bin_1:2.25-1.2* *libpam-cap_1:2.25-1.2*	libcap could be made to crash or possibly execute arbitrary code if it received a specially crafted input.[42]
bind9-host_1:9.11.3+dfsg-1ubuntu1.18 *dnsutils_1:9.11.3+dfsg-1ubuntu1.18* *libbind9-160_1:9.11.3+dfsg-1ubuntu1.18*	Bind could be made to crash if it received specially crafted network traffic.[43]
libldap-2.4-2_2.4.45+dfsg-1ubuntu1.11 *libldap-common_2.4.45+dfsg-1ubuntu1.11*	OpenLDAP could be made to crash if it received specially crafted input.[44]
libnghttp2-14_1.30.0-1ubuntu1	nghttp2 could be made to crash if it opened a specially crafted file.[45]
libx11-6_2:1.6.4-3ubuntu0.4 *libx11-data_2:1.6.4-3ubuntu0.4* *libx11-xcb1_2:1.6.4-3ubuntu0.4*	libx11 could be made to crash if it received specially crafted network traffic.[46]

[38] https://ubuntu.com/security/notices/USN-6259-1
[39] https://ubuntu.com/security/notices/USN-6154-1
[40] https://ubuntu.com/security/notices/USN-6270-1
[41] https://ubuntu.com/security/notices/USN-6302-1
[42] https://ubuntu.com/security/notices/USN-6166-2
[43] https://ubuntu.com/security/notices/USN-6183-2
[44] https://ubuntu.com/security/notices/USN-6197-1
[45] https://ubuntu.com/security/notices/USN-6142-1
[46] https://ubuntu.com/security/notices/USN-6168-2

libcups2_2.2.7-1ubuntu2.10	CUPS could be made to crash or expose sensitive information over the network.[47]
screen_4.6.2-1ubuntu1.1	GNU Screen could be made to crash applications if it received specially crafted input.[48]
libdw1_0.170-0.4ubuntu0.1 *libelf1_0.170-0.4ubuntu0.1*	elfutils incorrectly handled certain malformed files and incorrectly handled bounds checks in certain functions when processing malformed files.[49]
libavahi-client3_0.7-3.1ubuntu1.3 *libavahi-common-data_0.7-3.1ubuntu1.3* *libavahi-common3_0.7-3.1ubuntu1.3*	Avahi could be made to crash if it received specially crafted DBus traffic.[50]
open-vm-tools_2:11.0.5-4ubuntu0.18.04.2	open-vm-tools could be made to bypass authentication.[51]
openssh-client_1:7.6p1-4ubuntu0.7 *openssh-server_1:7.6p1-4ubuntu0.7* *openssh-sftp-server_1:7.6p1-4ubuntu0.7*	OpenSSH has an observable discrepancy leading to an information leak in the algorithm negotiation.[52]

It is recommended to upgrade all vulnerable software to the latest version. Furthermore, a software patching program ought to be implemented to ensure vulnerabilities in installed software are patched in a timely fashion. A common way to do this is to have a nightly job that looks for vulnerabilities in installed software and alerts a system administrator if any new issues affect any installed software. Automated tools such as vulnerability scanners could be helpful to facilitate this process.

[47] https://ubuntu.com/security/notices/USN-6184-2
[48] https://ubuntu.com/security/notices/USN-6198-1
[49] https://ubuntu.com/security/notices/USN-6322-1
[50] https://ubuntu.com/security/notices/USN-6129-2
[51] https://ubuntu.com/security/notices/USN-6257-1
[52] https://ubuntu.com/security/notices/USN-6279-1

GRT-01-018 WP1: Possible Quota Exhaustion via Exposed Secrets *(Low)*

While auditing the *newDisguiser* codebase, it was found that some APIs keys are exposed publicly. Attackers might reuse, abuse those APIs or might try to blacklist them in order to disrupt other users. Please note the impact of this issue is lowered by the fact that the *ATLAS_API_KEY* is no longer valid. Additionally, in a worst-case scenario the *ipinfo.io* token leak could only be used to exceed the API quota of 50k requests per month.

Example 1: Atlas API Key

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/ripe_atlas_client.py#L19

Affected Code:
```
ATLAS_API_KEY = 'f53[...]'
```

Example 2: ipinfo Token Leak

Affected File:
https://github.com/e2ecensor/newDisguiser/blob/d75edcf6bdd19815954d3c33c1fb70f5e1be4a01/Disguiser/code/filtered_request.py#L37

Affected Code:
```
url = 'http://ipinfo.io/' + ip + '?token=8342[...]'
```

It is recommended to remove all hard-coded credentials, tokens and private keys from the affected repositories. Once that is done, the git history ought to be scrubbed from these secrets. This could be accomplished utilizing tools like *BFG Repo-Cleaner*[53]. It is advised to invalidate all identified credentials and generate new ones. Automated tools such as *GitGuardian*[54], *TruffleHog*[55] and *Git Secrets commit hooks*[56] should be then considered for inclusion in the development process. This will drastically reduce the potential for similar issues in the future, due to repositories being scanned for secrets as developers commit code, and regularly.

[53] https://rtyley.github.io/bfg-repo-cleaner/
[54] https://www.gitguardian.com/
[55] https://github.com/trufflesecurity/trufflehog
[56] https://github.com/awslabs/git-secrets

Regarding the removal of credentials from the source code, please note that while environment variables would be better than hard-coding secrets in the source code (i.e. use a .env file added to a .gitignore file to store them and avoid publishing them publicly), these still have downsides and a dedicated secret management tool should be preferred[57]. Instead, applications should retrieve credentials from *AWS Secrets Manager*[58] or an equivalent secure vault that provides the application with credentials as needed at runtime but encrypts them at rest. This ensures that the applications can keep using the credentials while not being available to potential adversaries with access to leaked source code, a developer machine, or any other leak. Furthermore, credentials, secrets, and API keys should be randomly generated to mitigate the potential for brute force or password-guessing attacks. For additional mitigation guidance, please see the *OWASP Cryptographic Storage Cheat Sheet*[59] and the *CWE-798: Use of Hard-coded Credentials* page[60].

More broadly, it is important to have appropriate processes in place to:

- Regularly rotate credentials
- Revoke and replace credentials in the event of a compromise

GRT-01-020 WP1: Possible DoS via Predictable Proxy IPs *(Medium)*

It was found that Ground Truth routes traffic through pre-defined vantage points, such as the proxyrack setup, which can be prone to data corruption if the proxy provider is compromised. More broadly, the list of static IP addresses may be leveraged as a vantage point by scripts, or could become targets of attacks in order to cause data misclassification or damage the service availability. This issue can be confirmed reviewing the following code snippet example:

Affected File: Proyrack configuration script
https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d22dade26/Disguiser/code/setup.py#L4

Affected Code: Set up

```
def setup(platform):

    if platform == 'proxyrack':
        # proxyrack proxy setup
        proxyrack_proxy = dict()
```

[57] https://security.stackexchange.com/questions/197784/is-it-unsafe-to-use-env…
[58] https://aws.amazon.com/.../aws-secrets-manager-store-distribute-and-rotate-credentials.../
[59] https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html
[60] https://cwe.mitre.org/data/definitions/798.html

```
proxyrack_proxy['username'] = 'linjin'
proxyrack_proxy['password'] = 'd16dc4-8d5895-7a81c6-df52b2-ae9182'
proxyrack_proxy['proxy_address'] = 'megaproxy.rotating.proxyrack.net'
lower_port = 10000
upper_port = 10249
proxyrack_proxy['proxy_port'] = random.randint(lower_port, upper_port)
proxy = proxyrack_proxy
concurrency = 20
result_path = '../results/proxyrack/'
result_suffix = '_proxyrack_censorship_json.txt'
cert_filename = '../results/proxyrack/proxyrack_certs.json'
finished_countries_file = 'proxyrack_finished_countries.json'
log_file_path = '../results/proxyrack/'
timeout = 15
dns_server = '184.73.92.183'
http_server = '100.26.203.116'
sni_server = '54.166.38.207'
sni_server = '54.235.225.189'
```

It is recommended to switch from the current framework proxy setup, which relies on hardcoded services that may easily become targeted, to a dynamic configuration where user-specific configuration data could be added to avoid a single point of failure.

WP5: Supply Chain Implementation Analysis

Introduction and General Analysis

The *8th Annual State of the Software Supply Chain Report*, released in October 2022[61], revealed a 742% average yearly increase in software supply chain attacks since 2019. Some notable compromise examples include *Okta*[62], *Github*[63], *Magento*[64], *SolarWinds*[65] and *Codecov*[66], among many others. In order to mitigate this concerning trend, Google released an End-to-End Framework for *Supply Chain Integrity* in June 2021[67], named *Supply-Chain Levels for Software Artifacts* (*SLSA*)[68].

This area of the report elaborates on the current state of the supply chain integrity implementation of the Ground Truth project, as audited against the SLSA framework. SLSA assesses the security of software supply chains and aims to provide a consistent way to evaluate the security of software products and their dependencies.

The following sections elaborate on the results against version 1.0 of the SLSA standard.

In general, the first notable finding was that the Ground Truth team had no formal documentation for processes or procedures specific to supply chain security.

At the time of this assignment, components in scope can be grouped into:

- Web server
  - https://e2ecensor.github.io/
  - https://github.com/e2ecensor/e2ecensor.github.io
- Python scripts
  - https://github.com/e2ecensor/Disguiser_public
  - https://github.com/e2ecensor/newDisguiser
- Backend Control Server(s)

[61] https://www.sonatype.com/press-releases/2022-software-supply-chain-report
[62] https://www.okta.com/blog/2022/03/updated-okta-statement-on-lapsus/
[63] https://github.blog/2022-04-15-security-alert-stolen-oauth-user-tokens/
[64] https://sansec.io/research/rekoobe-fishpig-magento
[65] https://www.techtarget.com/searchsecurity/ehandbook/SolarWinds-supply-chain-attack...
[66] https://blog.gitguardian.com/codecov-supply-chain-breach/
[67] https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html
[68] https://slsa.dev/spec/

The web server and Python scripts are hosted in Github. The web server is powered by the *al-folio*[69] theme with *Jekyll* and is created automatically from source code using *Github Actions*[70], generating unsigned metadata[71] about how the build is created and the deployment is performed. The python scripts are simple files and no build process was found based on them. These scripts have been included in the analysis because researchers need information to decide whether or not to trust the scripts.

Finally, the backend control servers only have ports 80, 443, and 53 open for simultaneously accepting HTTP, HTTPS, and DNS requests hosted in AWS. These services have been installed by default and have almost no additional configuration. For this reason, backend control servers have not been included in the Supply Chain Implementation Analysis.

In order to produce artifacts with a specific SLSA level, the responsibility is split between the *Producer* and the *Build* platform. Broadly speaking, the *Build* platform must strengthen the security controls in order to achieve a specific level, while the *Producer* must choose and adopt a *Build* platform capable of achieving a desired SLSA level, implementing security controls as specified by the chosen platform.

SLSA v1.0 Analysis and Recommendations

SLSA v1.0 defines a set of four levels that describe the maturity of the software supply chain security practices implemented by a software project as follows:

- **Build L0: No guarantees**, represents the lack of SLSA[72].
- **Build L1: Provenance exists**. The package has provenance showing how it was built. This can be used to prevent mistakes but is trivial to bypass or forge[73].
- **Build L2: Hosted build platform**. Builds run on a hosted platform that generates and signs the provenance[74].
- **Build L3: Hardened builds**. Builds run on a hardened build platform that offers strong tamper protection[75].

The following sections summarize the results of the software supply chain security implementation audit, based on the SLSA v1.0 framework. Green check marks indicate that evidence of the SLSA requirement was found.

[69] https://github.com/alshedivat/al-folio
[70] https://github.com/features/actions
[71] https://github.com/e2ecensor/e2ecensor.github.io/actions
[72] https://slsa.dev/spec/v1.0/levels#build-l0
[73] https://slsa.dev/spec/v1.0/levels#build-l1
[74] https://slsa.dev/spec/v1.0/levels#build-l2
[75] https://slsa.dev/spec/v1.0/levels#build-l3

Producer

A package producer is the organization that owns and releases the software. It might be an open-source project, a company, a team within a company, or even an individual. The producer must select a build platform capable of reaching the desired SLSA Build Level.

Producer - Web server

Ground Truth selected Github as the build platform. Github is capable of producing Build Level 3 provenance. The build process is consistent, as all steps are scripted using *Github Actions*. Given that each time the *Build* is run, the *Build* platform generates logs that would be considered as valid unstructured *Provenance*, sufficient to comply with Level 1 of SLSA v1.0.

Requirement	L1	L2	L3
Choose an appropriate build platform	✅	✅	✅
Follow a consistent build process	✅	⛔	⛔
Distribute provenance	✅	⛔	⛔

Producer - Python scripts

Ground Truth selected Github to host python scripts. Github is capable of producing Build Level 3 provenance. The python scripts are simply stored files and no build process was found based on them.

Requirement	L1	L2	L3
Choose an appropriate build platform	✅	✅	✅
Follow a consistent build process	⛔	⛔	⛔
Distribute provenance	⛔	⛔	⛔

Build platform

A package build platform is the infrastructure used to transform the software from source to package. This includes the transitive closure of all hardware, software, persons, and

organizations that may influence the build. A build platform is often a hosted, multi-tenant build service, but it could be a system of multiple independent rebuilders, a special-purpose build platform used by a single software project, or even the workstation of an individual.

Build platform - Web server

The build process is scripted using *Github Actions*, meeting the *Hosted* requirement. Given that each time the *Build* is run, the *Build* platform generates unsigned logs that would be considered as valid unstructured *Provenance*, sufficient to comply with Level 1 of SLSA v1.0.

Requirement	Degree	L1	L2	L3
Provenance generation	Exists	✅	✅	✅
	Authentic		⛔	⛔
	Unforgeable			⛔
Isolation strength	Hosted		✅	✅
	Isolated			⛔

Build platform - Python scripts

Ground Truth selected Github to host python scripts. The python scripts are simple files and no build process was found based on them.

Requirement	Degree	L1	L2	L3
Provenance generation	Exists	⛔	⛔	⛔
	Authentic		⛔	⛔
	Unforgeable			⛔
Isolation strength	Hosted		⛔	⛔
	Isolated			⛔

In conclusion, only the Ground Truth Web server component is SLSA *Build* L1 (v1.0) compliant. Since the Web server and Python scripts components are stored in Github,

due to the available GitHub tools it is possible to improve the *Build* level as follows:

- From the python scripts, build a python package and upload[76] it to the Python Package Index (PyPI)[77].
- *GitHub Actions*[7879] should be leveraged to build and release the new python package. This would satisfy the requirement for choosing an appropriate build platform, as well as resolve the provenance-generation issue, given that each time the build is run, the build log would be considered as a valid unstructured provenance, sufficient to comply with SLSA *Build* L1 (v1.0).
- After the above, automated tools like *slsa-github-generator*[80] and *slsa-verifier*[81], could be integrated into the build process for the Web server and python scripts components to further harden the supply chain implementation.

[76] https://packaging.python.org/en/latest/tutorials/packaging-projects/
[77] https://pypi.org/
[78] https://docs.github.com/en/actions
[79] https://pythonprogramming.org/automatically-building-python-package-using-github-actions/
[80] https://github.com/slsa-framework/slsa-github-generator
[81] https://github.com/slsa-framework/slsa-verifier

WP6: Ground Truth Lightweight Threat Model

Introduction

The Disguiser and newDisguiser tools aim to provide an end-to-end framework for measuring Internet censorship practices with Ground Truth. The project involves the deployment of various components, datasets, and vantage points to accurately investigate global Internet censorship practices.

Threat model analysis assists organizations to proactively identify potential security threats and vulnerabilities, enabling them to develop effective strategies to mitigate these risks before they are exploited by attackers. Furthermore, this often helps to improve the overall security and resilience of a system or application.

The aim of this section is to facilitate the identification of potential security threats and vulnerabilities that may be exploited by adversaries, along with possible outcomes and appropriate mitigations.

Relevant assets and threat actors

The following assets are considered important for the Ground Truth project:

- Ground Truth source code
- Underlying Ground Truth dependencies
- Ground Truth researcher device (client)
- Ground Truth backend control server infrastructure
- Ground Truth project results
- Ground Truth project documentation

The following threat actors are considered relevant to the Ground Truth project:

- External malicious attackers (TA1)
- Internal collaborator (TA2)
- Third-party libraries (TA3)

Attack surface

In threat modeling, an attack surface refers to any possible point of entry that an attacker might use to exploit a system or application. This includes all the paths and interfaces that an attacker may use to access, manipulate or extract sensitive data from a system. By understanding the attack surface, organizations are typically able to identify potential attack vectors and implement appropriate countermeasures to mitigate risks. The following diagram provides an overview of potential attacks against the framework as envisioned by 7ASecurity:

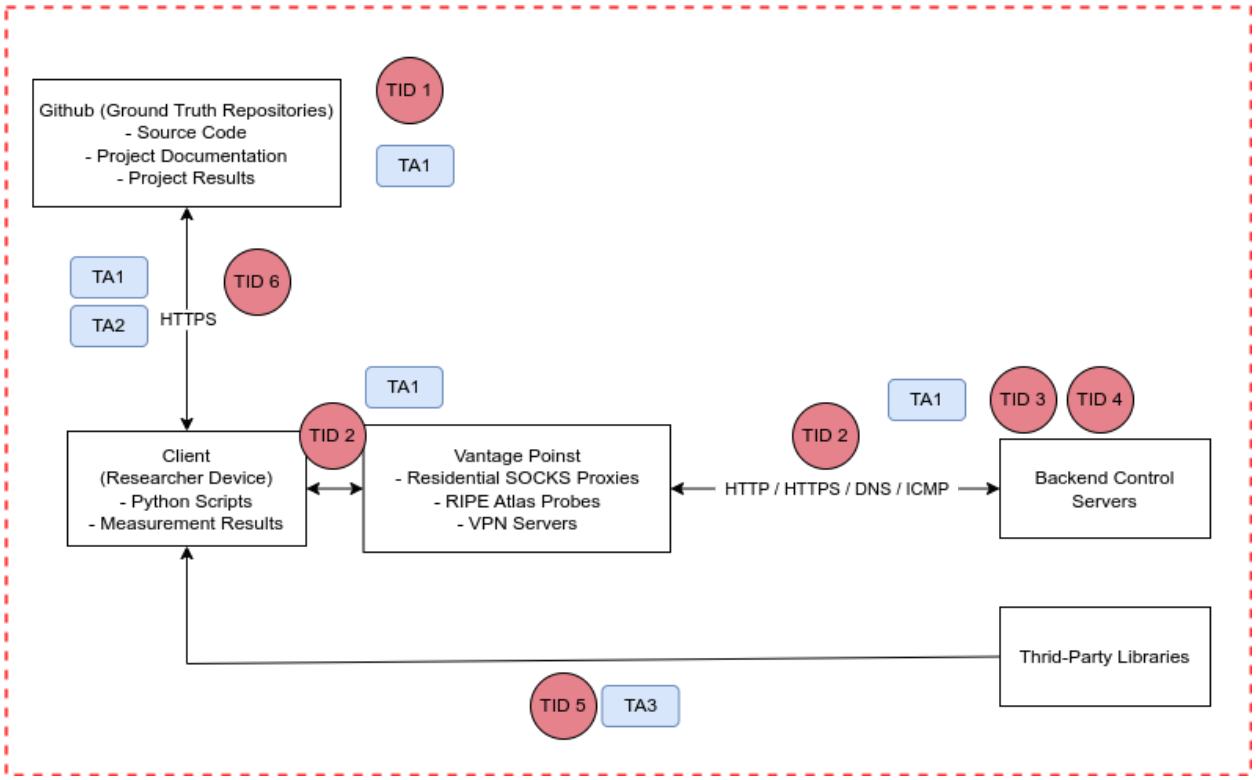


Fig.: Possible attacks

The identified threats against the Ground Truth components are as follows:

Threat 1 (TID1): Leakage of hard-coded credentials

Overview: Sensitive data extracted from the publicly available source code may allow attackers to gain unauthorized access to internal systems or services used by the environment.

Possible Outcome: Service disruption and financial loss in case of access to paid 3rd party services. Potential privilege escalation, in case of access to internal infrastructure resources (e.g. cloud environments, virtual machines).

Attack Scenario: The attacker extracts publicly available credentials hard-coded in the source code to access 3rd party services like IPinfo[82], ProxyRacks[83], RIPE Atlas[84], etc.

Recommendation: Utilization of secret scanning solutions to prevent storing sensitive data in the source code. Specialized solutions to store and manage secrets. Automated and parameterized build and deployment strategies. Separate development, testing and production environments compliant with appropriate security standards.

Threat 2 (TID2): Man-in-the-middle attacks against the client scripts

Overview: Data transferred over insecure, especially plaintext protocols may be easily modified by a malicious third party.

Possible Outcome: In-transit data modification or data sniffing may lead to service disruptions, censorship evasion, incorrect data collection, sensitive data disclosure and attacks against the scripts/libraries used by the researchers potentially leading to a compromise of the underlying machines.

Attack Scenario:
In-transit data modification to 3rd party services like IPinfo. Sniffing data to ProxyRack HTTP proxy. DNS rebinding attacks. Crafting HTML responses to target internal libraries.

Recommendation: Adequate input validation and exception handling should be implemented to prevent attackers from injecting malicious code. Parsing libraries ought to be regularly patched and any parser-related code needs to be reviewed to ensure secure coding practices are enforced in order to minimize the risk of introducing user-input parsing vulnerabilities.

Threat 3 (TID3): Unauthorized access to servers via management interfaces

Overview: Exposed management interfaces, such as SSH, may be used by attackers to gain a foothold within the environment.

[82] https://ipinfo.io/developers#authentication
[83] https://help.proxyrack.com/en/articles/5821332-authentication-and-ip-whitelisting
[84] https://atlas.ripe.net/docs/apis/rest-api-manual/api_keys.html

Possible Outcome: The attacker may gain access to the underlying operating system.

Attack Scenario: An exposed SSH interface with a weak password may be easily brute-forced by the attacker to gain access to the remote machine.

Recommendation:
Configure firewalls to prevent management interface access from unknown IP addresses. Harden the underlying operating systems and use strong authentication methods. Configure automatic updates to promptly apply security patches. use only strong authentication methods. Create personal accounts for all users to ensure proper accountability and follow the least privilege principle. Configure logging, monitoring and adequate alerts for early detection and contention of potential attacks.

Threat 4 (TID4): Exploitation of a known vulnerability in a service

Overview: Outdated software may contain known vulnerabilities that may be easily exploited using publicly known methods.

Possible Outcome: The attacker may gain access to the underlying operating system and tamper the Ground Truth data leading to service disruption or incorrect data collection.

Attack Scenario: Enumerate the version of the services running on the virtual machine. Exploitation using a publicly known exploit.

Recommendation: Operating system and service hardening including a configuration of automatic updates. Server banner removal to prevent easy version enumeration. Robust logging and monitoring as well as WAF configuration to detect and block exploitation attempts. Strict firewall rules to expose only necessary resources.

Threat 5 (TID5): Exploitation of a known vulnerability in a library used by the software

Overview: Outdated libraries may be used by more sophisticated attackers to target the researchers running the software.

Possible Outcome: Attacking the parsers used by the software may vary from low-level issues disrupting the service (denial-of-service) to critical issues (remote code execution) targeting researcher machines.

Attack Scenario: The attacker may monitor the dependencies used by the software to identify vulnerable libraries. In the case of parsers, the attacker may potentially craft a malicious HTML page to trigger the vulnerability and potentially gain control over the underlying machine.

Recommendation: Dependency vulnerability scanning should be regularly leveraged to scan the source code. The development team ought to be alerted in the event of the identification of a vulnerable dependency. Consideration should be given to pinning the dependencies to known stable versions, as this prevents supply chain attacks, when the provider of the library is compromised and pushes malicious code to the public repository.

Threat 6 (TID6): Compromised GitHub Collaborator

Overview: An attacker with privileged access may tamper with the source code and pivot to various environments.

Possible Outcome: All resources executing tampered source code may get compromised, leading to sensitive data leakage, service disruption and reputation damage.

Attack Scenario: The attacker gains access to a GitHub account (e.g. via a phishing campaign) of a user who has collaborator privileges and pushes malicious code to the main branch.

Recommendation: A robust DevSecOps process ought to be employed to protect all possible phases of the software development life cycle. Consideration should be given to automated, but staged builds which require multiple collaborators to review and accept changes to the source code before it is deployed. Security settings (e.g. MFA) should be enforced for services used by staff. Security awareness and sensitive data handling training may then be provided to staff members. Emergency accounts must then be in place, in case a highly privileged account is compromised. security procedures may then be tested using a scenario-based approach to validate in practice security processes.

WP7: Privacy Analysis Findings

This section covers the privacy-related analysis results that attempt to answer 12 questions for *WP7: Privacy tests against Ground Truth Servers & Clients*. For this portion of the engagement, the *7ASecurity* team utilizes the following classification to specify the level of certainty regarding the documented findings. Given that this research occurred on the basis of reverse engineering, and source code analysis, it is necessary to classify the findings to address the level of confidence that can be assumed for each discovery:

- Proven: Source code and runtime activity clearly confirm the finding as fact
- Evident: Source code strongly suggests a privacy concern, but this could not be proven at runtime
- Assumed: Indications of a potential privacy concern was found but a broader context remains unknown.
- Unclear: Initial suspicion was not confirmed. No privacy concern can be assumed.

GRT-01-Q01: Files & Information gathered by Ground Truth *(Unclear)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q1: What files/information are gathered by the Ground Truth scripts and servers?

During the audit, it was confirmed that Ground Truth does not collect sensitive information, as the framework mainly deals with censorship statistics. However, 7ASecurity identified a number of cases where the scripts are prone to Denial of Service attacks through crafted HTTP responses. The vulnerabilities are especially important as they may compromise data gathering processes and alter the collected statistics directly from the censor-level sending the response back to the client (GRT-01-002).

By design, the Ground Truth framework processes information provided by the control server deployed and acts as a static payload used for experimental data. The same information is being processed for censorship detection purposes and the results are interpreted, compared and stored in local files. Among the primary information stored by the framework, the team noted various text and JSON formatted files containing configuration data or a list of domain names gathered from public data. The results are saved using either CSV or JSON file extensions. The following examples illustrate the non-sensitive nature of the information gathered:

Example File: Configuration File Loading
https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d

22dade26/Analysis/proxyrack.py#L208

Example Code: Proxy Script

```
if __name__ == "__main__":
    start = time.time()
    country_count = {}
    while True:
        end = time.time()
        if end - start > 86400 * 7:
            break
        if len(country_count) != 0:
            flag = True
            for item_count in country_count.items():
                if item_count[1]<=20:
                    flag = False
                    break
            if flag:
                break
        # read domain
        data = read_domain("./http_overall.txt")

        # read proxy info
        with open("./proxy.json") as f:
            proxy = json.loads(f.read())
```

Example File Contents: *http_overall.txt*

{"country": "Japan", "total_domains": 1914, "domain": ["www.serials.ws", "go-text.me", "astalavista.box.sk", "cultdeadcow.com", "insecure.org", "metasploit.com", "rarbg.to", "thepiratebay.org", "www.3sk.co", "www.52pojie.cn", "www.abortionno.org", "www.blackhat.be", "www.darknet.org.uk", "www.exploit-db.com", "www.hackcanada.com", "www.hackforums.net", "www.hackhull.com", "www.hacktivismo.com", "www.kinogo.by", "www.oovoo.com", "www.royalvegas.com", "www.seasonvar.ru", "www.yts.mx", "www.fadsabs.com", "www.fadsecs.com", "www.fadsims.com", "www.fadsips.com", "www.fadskis.com", "www.fadslims.com", "www.fadsoks.com", "www.fadspms.com", "www.sadsecz.com", "secure.proxpn.com", "www.ladbible.com", "www.phenoelit.org"], "category": ["Illegal or Unethical", "Instant Messaging", "Illegal or Unethical", "Hacking", "Information Technology", "Information Technology", "Illegal or Unethical", "Illegal or Unethical", "Information Technology", "Newsgroups and Message Boards", "Abortion", "Hacking", "Information Technology", "Hacking", "Hacking", "Hacking", "Hacking", "Hacking", "Illegal or Unethical", "Internet Telephony", "Gambling", "Illegal or Unethical", "Illegal or Unethical", "Advertising", "Spam URLs", "Spam URLs", "Spam URLs", "Malicious Websites", "Advertising", "Spam URLs", "Malicious Websites", "Spam URLs", "Proxy Avoidance", "Personal Websites and Blogs", "Hacking"], "text": [13964, 28998, 3451, 3448, 3445, 3447, 3649, 3665, 3458, 3462, 3621, 3448, 3451, 3451, 3451, 3451, 3449, 3452, 3461, 3611, 3621, 3464, 3447, 3458, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 21510, 21510, 21510, 21510, 21510, 21512, 21510, 21510, 21510], "is_timeout": [false, false,

false, false, true, true, true, true, true, true, true, true, true, false, false,
false, false, false, false, false, false, false, false, false, false, false],
"total_requests": 344520, "count": 35, "percentage": 0.018286311389759665}[...]

Example File: Domain File loading

https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d22dade26/Analysis/allDomain/processAllDomain.py#L4

Example Code: Domain Loading

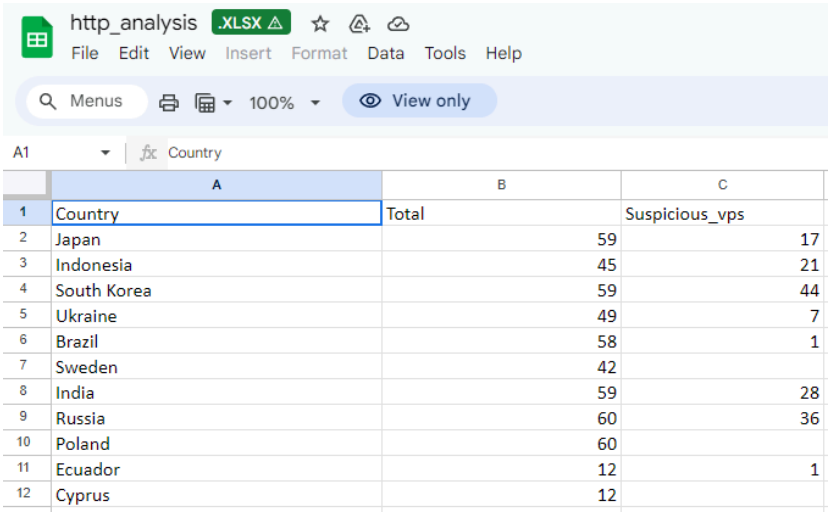
```
import sys, os
import json

def main():
	with open("allDomain.txt", "r") as f:
		data = f.read()

	domainJson = json.loads(data)
	domainList = domainJson.keys()

	for domain in domainList:
		print(domain)

	return 0

if __name__ == '__main__':
	main()
```

Example File Contents: *allDomain.txt*


```
{
    "www.serials.ws": [
        "Japan",
        "Mexico",
        "Italy",
        "Thailand",
        "South Korea",
        "France",
        "Argentina",
        "Chile",
        "Australia",
        "Kazakhstan",
        "United Arab Emirates",
        "Ukraine",
        "Germany",
        "United Kingdom",
        "New Zealand",
        "Taiwan",
        "Czechia",
```

```
        "Ecuador",
        "Panama",
        "Palestine",
        "Israel",
        "Nicaragua",
        "Moldova",
        "Kuwait",
        "El Salvador",
        "Palestinian Territory",
        "Yemen"
    ],[...]
```

The HTTP analysis results are saved in excel files and correlated to determine the number of suspicious censored traffic:



http_analysis .XLSX

File Edit View Insert Format Data Tools Help

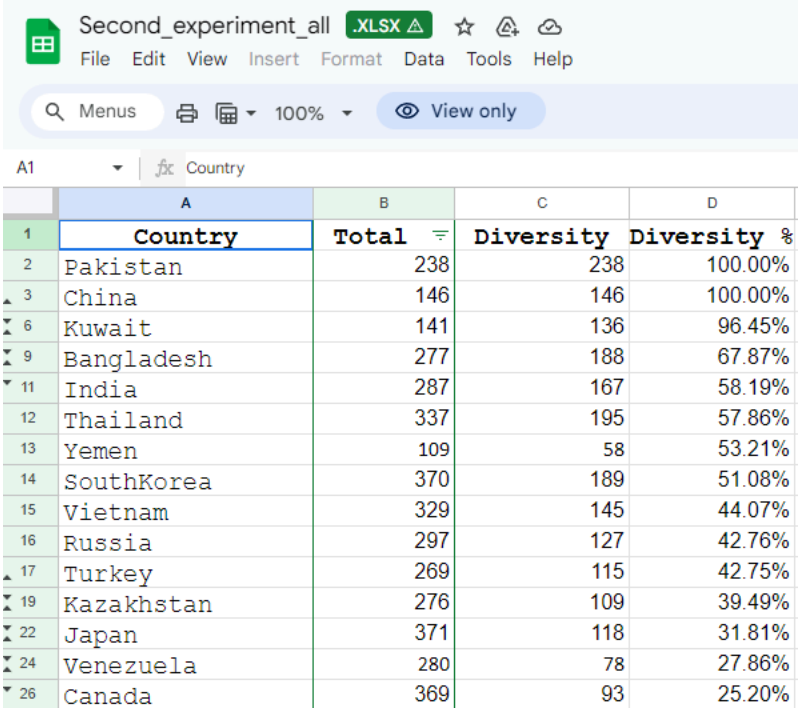
Menus 100% View only

A1 Country

	A	B	C
1	Country	Total	Suspicious_vps
2	Japan	59	17
3	Indonesia	45	21
4	South Korea	59	44
5	Ukraine	49	7
6	Brazil	58	1
7	Sweden	42	
8	India	59	28
9	Russia	60	36
10	Poland	60	
11	Ecuador	12	1
12	Cyprus	12	

Fig.: Results from HTTP analysis

Furthermore, interpreted results are saved in Excel files and contain numbers resulting from the experimental data sent back and forth between the control server, the client and the censor list selected with the applied logic evaluation.



Second_experiment_all .XLSX

File Edit View Insert Format Data Tools Help

Menus 100% View only

A1 Country

	A	B	C	D
1	Country	Total	Diversity	Diversity %
2	Pakistan	238	238	100.00%
3	China	146	146	100.00%
6	Kuwait	141	136	96.45%
9	Bangladesh	277	188	67.87%
11	India	287	167	58.19%
12	Thailand	337	195	57.86%
13	Yemen	109	58	53.21%
14	SouthKorea	370	189	51.08%
15	Vietnam	329	145	44.07%
16	Russia	297	127	42.76%
17	Turkey	269	115	42.75%
19	Kazakhstan	276	109	39.49%
22	Japan	371	118	31.81%
24	Venezuela	280	78	27.86%
26	Canada	369	93	25.20%

Fig.: Experimental results file

Depending on the protocol chosen by the user to execute censorship tests, the scripts perform different traffic analysis operations and save the results in local files.

Example File: Proxyrack Process Script
https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d22dade26/Disguiser/code/proxyrack_process.py#L232

Example Code: How results are saved

```
with open('../materials/domain_webpage/valid_domains.txt') as f:
    valid_domains = f.read().strip().split()
valid_domain_dic = dict()
for domain in valid_domains:
    valid_domain_dic[domain] = True


protocol = sys.argv[1]


if protocol == 'dns':
    result_dic = process_dns()
if protocol == 'http':
    result_dic = process_http()
```

```
if protocol == 'sni':
    result_dic = process_sni()

with open(result_path + start_date + '/' + start_date + '_' + protocol + '_final.txt',
'w') as f:
    for country in result_dic.keys():
        json_string = json.dumps(result_dic[country])
        f.write(json_string + '\n\n')
```

As illustrated in the examples above, since the nature of the data gathered is not sensitive, there is no action required from the Ground Truth team to improve the privacy posture in regards to this question.

GRT-01-Q02: Insecure Ground Truth Traffic Leads to RCE & Spoofing *(Proven)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q2: Where and how are the files/information gathered transmitted?
What information can the ISP see, if a user is using the scripts in a high risk scenario?

Ground Truth was found to use clear-text HTTP communications for a number of purposes, which enables the exploitation of Remote Command Execution (RCE) vulnerabilities (GRT-01-004, GRT-01-019), as well as spoofing and misclassification of censorship results, via modification of clear-text HTTP traffic (GRT-01-019).

Additionally, in regard to the censorship results and analysis process, the team noted a number of vulnerabilities that could create potential false positives, misclassifications or spoofed results (GRT-01-001, GRT-01-017).

Regarding where the information is transmitted, Ground Truth sends data generated from public sources when making the setup for the framework. The control server contains static payloads that are used for censorship tests and the traffic is sent to the list of censors.

Example File: Framework Setup Script
https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d22dade26/Disguiser/code/setup.py#L4

Example Code: Set up

```
def setup(platform):

    if platform == 'proxyrack':
```

```
        # proxyrack proxy setup
        proxyrack_proxy = dict()
        proxyrack_proxy['username'] = 'linjin'
        proxyrack_proxy['password'] = 'd16dc4-8d5895-7a81c6-df52b2-ae9182'
        proxyrack_proxy['proxy_address'] = 'megaproxy.rotating.proxyrack.net'
        lower_port = 10000
        upper_port = 10249
        proxyrack_proxy['proxy_port'] = random.randint(lower_port, upper_port)
        proxy = proxyrack_proxy
        concurrency = 20
        result_path = '../results/proxyrack/'
        result_suffix = '_proxyrack_censorship_json.txt'
        cert_filename = '../results/proxyrack/proxyrack_certs.json'
        finished_countries_file = 'proxyrack_finished_countries.json'
        log_file_path = '../results/proxyrack/'
        timeout = 15
        dns_server = '184.73.92.183'
        http_server = '100.26.203.116'
        sni_server = '54.166.38.207'
        sni_server = '54.235.225.189'

if platform == 'vpn':
        proxy = dict()
        concurrency = 50
        result_path = '../results/vpn/'
        result_suffix = '_vpn_censorship_json.txt'
        cert_filename = '../results/vpn/vpn_certs.json'
        finished_countries_file = ''
        log_file_path = '../results/vpn/'
        timeout = 5
        dns_server = '3.91.105.244'
        http_server = '52.91.166.212'
        sni_server = '18.207.203.33'
        #sni_server = '3.80.202.200'
```

Example File: HTTP Analysis Script

https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d22dade26/Analysis/http_analysis.py#L8

Example Code: Censhorship Analysis

```
url_list = ["", "", "", "", "", "", ""]
correct_http_page = 'http\n'
[...]
for domain in data['domain']:
            for url in url_list:
                vp_response = data['domain'][domain][url]['text']
```

```
                if vp_response == correct_http_page:
                    data['domain'][domain][url] = "no censorship - correct http"

                else:
                    try:

                        vp_title = BeautifulSoup(vp_response,
"html.parser").title.string
                        local_title = webpage_title_dic[domain]
                        if vp_title == local_title and local_title != '':
                            data['domain'][domain][url] = "no censorship - correct
title"
                        else:

                            data['domain'][domain][url] = "detect censorship - wrong
title"

                    except:
                        data['domain'][domain][url] = "detect censorship - wrong http"
```

It is recommended to extrapolate the mitigation guidance offered under GRT-01-019 to resolve this issue.

GRT-01-Q03: Ground Truth does not store or deal with PII *(Unclear)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q3: Is sensitive PII insecurely stored or easily retrievable from the scripts or servers?

During the testing and code review processes, the audit team did not identify any processing of sensitive PII data. The results of the scripts are stored locally and they only contain statistical information about the experiment. Furthermore, any emails contained in the scripts belong to the Ground Truth developers.

Example File: HTTP Analysis Request Script
https://github.com/e2ecensor/Disguiser_public/blob/8ec0f0ae76ecb894e68e2b810fa98ba007b2e8e7/analysis/http_heuristics.py#L9

Example Code:

```
correct_http_page = 'http\n'
correct_http_contact = 'linjin@udel.edu'

result_path = '../results/proxyrack/'
start_date = sys.argv[1]
suffix = 'http_proxyrack_censorship_json.txt'
```

```
with open('../results/proxyrack/excluded/excluded_http_probe.txt') as f:
    http_excluded_ip = f.read().strip().split()
with open('../results/proxyrack/excluded/excluded_http_probe_manual.txt') as f:
    http_excluded_ip_manual = f.read().strip().split()

with open('../materials/domain_webpage/valid_domains.txt') as f:
    valid_domains = f.read().strip().split()
valid_domain_dic = dict()
for domain in valid_domains:
    valid_domain_dic[domain] = True
[...]
```

Command:

```
newuser@ip-172-31-39-21:/var/www/35.180.190.69$ ls -lah
```

Output:

```
total 12K
drwxr-xr-x 2 root root 4.0K Aug 21 18:45 .
drwxr-xr-x 5 root root 4.0K Aug 21 16:37 ..
-rw-r--r-- 1 root root    5 Aug 21 18:45 index.html
```

Command:

```
newuser@ip-172-31-39-21:/var/www/35.180.190.69$ cat index.html
```

Output: Content of the control server static page

```
http
```

As illustrated in the example above, since the nature of the data gathered is not sensitive, there is no action required from the Ground Truth team to improve the privacy posture in regards to this question.

GRT-01-Q04: Ground Truth does not protect Data at Rest or in Transit *(Proven)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q4: Do the scripts and servers protect the data appropriately at rest and in transit?

First of all, it should be noted that the kind of data Ground Truth deals with is merely statistics, not Personally Identifiable Information (PII) and hence, the data is not sensitive by nature. That being said, during the code review phase, 7ASecurity noted that data is not protected in transit or at rest. The majority of the results and configuration files are kept locally, on the script root directory location.

Regarding data protection in transit, a general weakness exists via the predictability of the IPs that Ground Truth will use, as described in GRT-01-020. However, a more serious concern is the usage of clear-text network protocols leading to RCE vulnerabilities (GRT-01-019).

Regarding data protection at rest, the following code snippet illustrates how Ground Truth uses clear-text files:

Example File: Proxyrack Process Script
https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d22dade26/Disguiser/code/proxyrack_process.py#L232

Example Code: How results are saved

```
with open('../materials/domain_webpage/valid_domains.txt') as f:
    valid_domains = f.read().strip().split()
valid_domain_dic = dict()
for domain in valid_domains:
    valid_domain_dic[domain] = True
protocol = sys.argv[1]
if protocol == 'dns':
    result_dic = process_dns()
if protocol == 'http':
    result_dic = process_http()
if protocol == 'sni':
    result_dic = process_sni()
with open(result_path + start_date + '/' + start_date + '_' + protocol + '_final.txt',
'w') as f:
    for country in result_dic.keys():
        json_string = json.dumps(result_dic[country])
        f.write(json_string + '\n\n')
```

It is recommended to store output data in a secure location that can provide access control, data encryption and anti-tampering options.

It is also recommended to extrapolate the mitigation guidance offered under GRT-01-019 to resolve this issue.

GRT-01-Q05: Ground Truth does not gather Excessive Data *(Unclear)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q5: Is there any data gathered on the scripts & servers beyond what is necessary for the service?

During the code review and dynamic analysis of the Ground Truth framework, the audit team noted that no excessive data is either generated or gathered. The Ground Truth static payloads are exercised exclusively for experimental data and the public domain list and configuration files are used primarily for statistical analysis and setup. Furthermore, none of this information is sensitive.

The following example illustrates how the gathered data consists of the static payload data being received from the censor through the client-control server communication.

Example File: Probing censor
https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d22dade26/Disguiser/code/proxy_request.py#L261

Example Code:

```
try:
    sni_result['cert'] = ssl.DER_cert_to_PEM_cert(wrapped_socket.getpeercert(True))
    x509 = OpenSSL.crypto.load_certificate(OpenSSL.crypto.FILETYPE_PEM,
sni_result['cert'])
    sni_result['cert_serial'] = str(x509.get_serial_number())
    if sni_result['cert_serial'] != '[..]' and sni_result['cert_serial'] != '[..]' and
sni_result['cert_serial'] != '0':
        request = "GET / HTTP/1.1\r\nHost: %s\r\nUser-Agent: Mozilla/5.0\r\n\r\n" %
domain
        request = request.encode()
        wrapped_socket.send(request)

        try:
            raw_http_response = recvall(wrapped_socket)
            is_http_timeout = False
            except socket.timeout:
```

```
                raw_http_response = b''
                is_http_timeout = True
                except:
                    raw_http_response = b''
                    is_http_timeout = False
                http_result = process_raw_http_response_from_sni(raw_http_response,
is_http_timeout)
                sni_result['http_result'] = http_result
```

No action is required by Ground Truth to improve the privacy posture in this regard.

GRT-01-Q06: Ground Truth does not Track Users *(Unclear)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q6: Do the scripts implement any sort of user tracking function via location or other means?

During the source code review, 7ASecurity did not identify any signs of user tracking functionality, as by design, the framework does not implement such features. Instead, the scripts perform data tracking and integrity checks. The following diagram illustrates how Ground Truth just tracks censorship, not users:

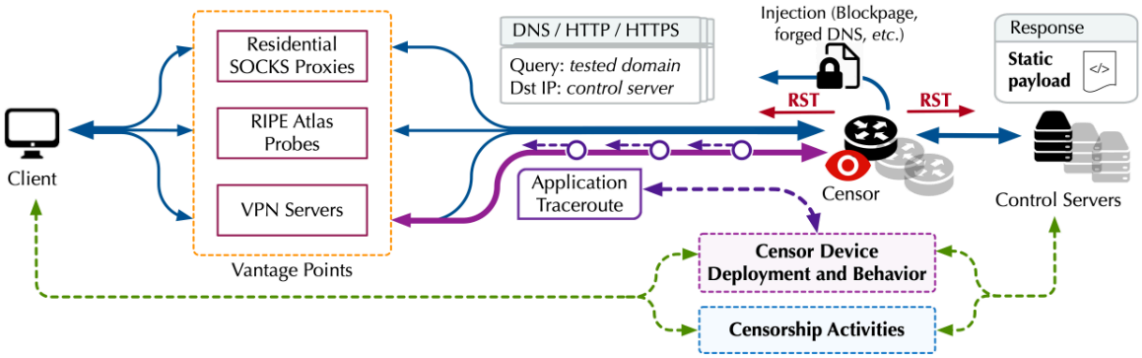


Fig.: Framework design according to https://e2ecensor.github.io/design/

No action is required by Ground Truth to improve the privacy posture in this regard.

GRT-01-Q07: Ground Truth does not weaken Crypto *(Unclear)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q7: Do the scripts intentionally weaken cryptographic procedures to ensure third-party decryption?

7ASecurity was unable to find any instances of intentional weakening of cryptography during this assignment. Broadly speaking, as per the design of the framework, the cryptographic procedures are standard for the outbound and inbound traffic sent through the control server and the client. Furthermore, the scripts are using checks for analyzing censorship certificate tampering, as illustrated in the following code snippet:

Example File: Proxyrack analysis
https://github.com/e2ecensor/newDisguiser/blob/1aa8e246f3dc69470bd17be073652e1d22dade26/Disguiser/code/proxyrack_process.py#L6

Example Code:

```
correct_http_page = 'http\n'
correct_http_contact = 'linjin@udel.edu'
correct_cert_serial = ['85723161702102284164881707705813409552803205256',
'20161409920381783884204342667071563908125516496 4']
[...]
if test_result['cert_serial'] not in correct_cert_serial:
                        if domain not in result_dic[country]['domain']:
                            result_dic[country]['domain'][domain] = dict()
                            result_dic[country]['domain'][domain]['category'] =
domain_category_dic[domain]
                            result_dic[country]['count'] += 1
                            # result_dic[country]['domain'][domain]['status_code'] =
test_result['status_code']
                            # result_dic[country]['domain'][domain]['url'] =
test_result['url']
                            result_dic[country]['domain'][domain]['cert_serial'] =
[test_result['cert_serial']]
                            result_dic[country]['domain'][domain]['ip'] = [ip]
```

No action is required by Ground Truth to improve the privacy posture in this regard.

GRT-01-Q08: Ground Truth saves Data in Insecure Locations *(Assumed)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q8: Is data dumped in insecure locations from where it could be retrieved later by an attacker or malicious insiders?

First of all, it must be emphasized again that the nature of the data that Ground Truth deals with is of a non-sensitive nature, hence, even though the data is saved insecurely, there are no sensitive secrets to protect either.

In the event that Ground Truth starts collecting more sensitive data in the future, the following points can be considered to further enhance the privacy posture:

The AWS cloud configuration data volumes across all used regions were found to be stored without prior encryption at rest. Although no sensitive data was identified in the stored volumes, potential flaws in the AWS implementation might allow unauthorized attackers to access the volume (GRT-01-005).

It is advised to extrapolate the mitigation guidance offered under GRT-01-005 to resolve this issue.

GRT-01-Q09: Ground Truth contains RCE Vulnerabilities *(Proven)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q9: Do the scripts or servers contain vulnerabilities or shell commands that could lead to RCE in any way?

During the code review and fuzzing of the Ground Truth scripts and web deployments, the team noted multiple Remote Code Execution vulnerabilities which may be exploited though attacker-supplied domain files (GRT-01-003), crafted configuration files (GRT-01-004), as well as attacker-tampered clear-text traffic (GRT-01-019).

While 7ASecurity believes these vulnerabilities have not been introduced intentionally, they should be resolved to improve the overall security posture of the platform, and avoid putting researchers using this tool at risk from government-sponsored adversaries.

GRT-01-Q10: Ground Truth does not appear to contain Backdoors *(Assumed)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q10: Do the scripts or servers have any kind of backdoor?

7ASecurity did not identify any evidence of intentional process or command execution calls commonly used by backdoors or malware in the Ground Truth framework scripts during this audit. However, the seemingly unintended RCE vulnerabilities spotted throughout this audit (GRT-01-003, GRT-01-004, GRT-01-019) should be resolved to improve the security posture and fully resolve this issue.

GRT-01-Q11: Ground Truth does not try to gain Root Privileges *(Unclear)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q11: Do the scripts attempt to gain root access through public vulnerabilities or in other ways?

At the time of writing, no evidence could be identified to suggest that any of the framework components are trying to leverage or exploit platform-specific vulnerabilities to gain elevated privileges. Therefore, no action is required by Ground Truth to improve the privacy posture in this regard.

GRT-01-Q12: Ground Truth uses no Obfuscation *(Unclear)*

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q12: Do the scripts use obfuscation techniques to hide code and if yes for which files and directories?

7ASecurity found no obfuscation evidence across the codebase. Furthermore, the Ground Truth framework is operating at a high transparency level already, as the code is publicly available online, without any closed-source components. Hence, no action is required by Ground Truth to improve the privacy posture in this regard.

Conclusion

Despite the number and severity of findings encountered in this exercise, the Ground Truth solution defended itself well against a broad range of attack vectors. The platform will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The Ground Truth framework provided a number of positive impressions during this assignment that must be mentioned here:

- 7ASecurity was unable to identify any vulnerability on the official Ground Truth website. The reason for this is that Ground Truth leverages hardened third party components which offer little attack surface for this part of the project. Specifically, the web server is powered by the *al-folio*[85] theme with *Jekyll* and is created automatically from source code using *Github Actions*[86], which explains why no web server or web attack vector vulnerabilities could be identified during this assignment.
- The proxy server and VPN connections are constantly checked for service downtime and connectivity status.
- SSH is guarded using SSH keys rather than a simple password. Furthermore, in some cases the port is not even exposed to the Internet.
- Regarding the privacy audit, it was confirmed that Ground Truth does not collect sensitive information or track users, as it only deals with censorship statistics.

The security of the Ground Truth solution will improve substantially with a focus on the following areas:

- **Reduction of the Attack Surface:** A number of critical *Remote Code Execution* (RCE) issues identified during this assignment occurred because the platform concatenates tainted input into operating system commands that are executed later (GRT-01-003, GRT-01-004, GRT-01-019). All these issues should be resolved utilizing alternative implementations that do not rely on operating system commands, such as using the *requests* library[87] instead of *cURL* commands, as well as the available python functions for safe execution of operating system commands[88]. Furthermore, strong consideration should be given to implement a container-based version of the framework that includes sandboxing to drastically limit the potential for similar attacks in the future.
- **TLS Hardening**: A number of scripts employ insecure clear-text protocols for

[85] https://github.com/alshedivat/al-folio
[86] https://github.com/features/actions
[87] https://pypi.org/project/requests/
[88] https://docs.python.org/3/library/subprocess.html

network communications, which facilitates the exploitation of critical RCE vulnerabilities substantially (GRT-01-019). An effort should be made to stick to hardened TLS communications to protect users from Man-In-The-Middle (MitM) attacks. In situations where clear-text protocols are required for testing data, exposure to attacks should be limited by implementing sandboxing on a pivoting service designed to route inbound and outbound traffic.

- **Input Validation:** It is important to perform input validation to ensure that only correctly formed data enters the application workflow. Strong input validation and sanitization ought to be in place when using user-supplied data or dynamic configuration details in operating system commands, as well as any other sensitive sinks.
- **Supply Chain Hardening:** The Ground Truth framework should take advantage of a number of Github features to easily improve its Supply Chain security posture against the SLSA standard (WP5).
- **Software Patching:** The platform should implement appropriate software patching procedures which regularly apply security patches in a timely manner (GRT-01-010, GRT-01-015). In a day and age when most lines of code come from underlying software dependencies, regularly patching these becomes increasingly important to avoid unwanted security vulnerabilities. Possible automation for this could include tools like *Snyk.io*[89] or *Renovate Bot*[90].
- **Secret Management** should be improved to ensure application secrets are not disclosed via hardcoded credentials or the commit history (GRT-01-018). Instead, these ought to be stored outside of the source code to reduce the potential for leaks and privilege escalation throughout the infrastructure. Special care should be taken to ensure credentials are also removed from the github history. The development team should then perform global searches and educate developers to avoid similar issues in the future. More broadly, adequate IT security and DevSecOps procedures are needed at the infrastructure level. Insecure storage of secrets was found at different steps of CI/CD pipeline, which strongly suggests the whole process should be reviewed holistically and improved.
- **Exception Handling** ought to be enhanced to avoid censorship spoofing (GRT-01-001) and DoS (GRT-01-002) via crashes, as well as potential logic errors (GRT-01-017).
- **Automation Improvements:** A number of issues identified during this iteration could have been found by improving the automation of CI/CD pipelines in the cloud environments (GRT-01-006). It is strongly recommended to leverage automation (GRT-01-009), such as infrastructure as code (i.e. Terraform), to develop a self-documented and repeatable multi-cloud infrastructure, as well as

[89] https://snyk.io/
[90] https://github.com/renovatebot/renovate

uncover and resolve security issues in a timely manner. The approach should include the configuration of tools that regularly scan the relevant repositories. Additionally, it is important to ensure deployment processes are reviewed and improved so that at least two members of staff are required to make any modification in the production environment.

- **Least Privilege:** Several of the spotted weaknesses during this exercise had to do with the potential for privilege escalation due to excessive privileges (GRT-01-007, GRT-01-013). It is crucial to implement the least privilege security principle, thoroughly reviewing all permissions to ensure privileges are always strictly the minimum possible for the solution to operate.
- **Implementation of Security Standards**: An effort should be made to employ well-known security standards to harden the cloud environment. A good starting point in this regard would be to implement the *CIS Critical Security Controls*[91] and then test security controls against the *CIS Benchmarks*[92].
- **Test Environment:** Other potential future improvements could include the implementation of a proof-of-concept Ground Truth infrastructure that acts as a censor and simulates attacks on the bi-lateral communication with the framework to test for edge cases.

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the platform significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another code audit, is highly recommended to ensure adequate security coverage of the platform.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing Ground Truth resources as well as perhaps an Azure cloud configuration audit, as access to that could not be granted on time during this iteration.

[91] https://www.cisecurity.org/insights/white-papers/cis-controls-cloud-companion-guide
[92] https://www.cisecurity.org/insights/blog/foundational-cloud-security-with-cis-benchmarks

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank Shuai Hao, Xiaoqin Liang and the rest of the Ground Truth team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the *Open Technology Fund (OTF)* for sponsoring this project.