SUBGRAPH

MASSBROWSER PRELIM FINDINGS REPORT REV 2

MassBrowser

March 5, 2020

Prepared for: MassBrowser

Subgraph Technologies, Inc.
642 Rue de Courcelle, Suite 309
Montreal, Quebec
https://subgraph.com

Testing Parameters

Testing Environment

For the purpose of testing, Subgraph deployed an isolated MassBrowser network comprised of:

- 1 Operator (Python/Django, Celery workers, Redis, ZMQ)
- Multiple relays (NodeJS, Vue.js, Electron)
- Multiple clients (NodeJS, Vue.js, Electron, Linux only)

There was also limited testing of the development and pre-prod QA environment in operation on the Internet from the perspective of hostile relays and clients. For the latter the MassBrowser team provided Subgraph with invitation codes. Subgraph only directly tested the relays and clients that we operated ourselves, though we could observe the behavior of the Operator remotely.

Design of MassBrowser Architecture and Protocol

The audit performed by Subgraph was limited in scope to the implementation and configuration of the various MassBrowser architectural components. There was also some consideration of the system's design with respect to any significant negative contribution to the technical security posture i.e., identification of severe technical design faults.

Subgraph did not assess the effectiveness of MassBrowser in meeting its broad objective of providing a feasible and robust censorship circumvention mechanism. MassBrowser is an ongoing research project conducted within the Secure, Private Internet (SPIN) Research Group at the University of Massachusetts Amherst and the efficacy of its design vs. total objectives will be comprehensively examined during the course of academic review.

Rather our goals were to test resistance to attacks against the software components as they exist today. This included attacks by malicious participants operating within the system, as well as identifying more passive weaknesses in components and their interaction with one another.

CacheBrowser

Analysis and review of the CacheBrowser protocol was considered out of scope as this component originated in a prior independent project and was then incorporated into MassBrowser. Subgraph did perform a limited review of the JavaScript implementation within MassBrowser which was distinct from the original.

Notes and Observations

Production Environment Configuration State

Subgraph observed that the production environment was not configured as securely as it could be, e.g. running in debug mode. While we note these as findings, we acknowledge that the "production" environment is not yet fully production as there is no general enrollment for clients at this time, and there is ongoing development.

Codebase Maturity

A few of the issues we have identified suggest that the code is not yet mature enough to be considered robust. For example, the two denial of service issues are both related to unhandled exceptions. Neither of those correspond to especially severe or rare events and could be handled gracefully at runtime rather than terminating the application. Subgraph recommends more testing, broadly across the entire application, to locate and fix these issues prior to production deployment. One of these issues caused a network-wide outage (though the health monitor did detect this, alerting the administrators who manually intervened).

Client / Relay Session Transport

The client attempts to create an encrypted tunnel using AES GCM. We assume the primary motivation is avoidance of detection by Deep Packet Inspection rather than transport security. Clients can and do use HTTPS when required and this is supported end to end without involvement of relays or the Operator. For this reason, Subgraph did not critically examine the cryptographic security of this transport.

Summary

No.	Title	Severity	Remediation
V-001	Relay IP Address Denial of Service	High	In Progress
V-002	Adversary Procurement of Trusted Weak Certificates for Arbitrary Websites	High	In Progress
V-003	API Cross-Site Request Forgery	High	In Progress
V-004	User Identification and Tracking by Silent Remote CacheBrowser CA Certificate Retrieval	High	In Progress
V-005	CacheBrowser Weak Server Certificate Generation	Medium	In Progress
V-006	Client Denial of Service	Medium	In Progress
V-007	Proxy Services Unnecessarily Bind to INADDR_ANY	Medium	In Progress
V-008	Internal Endpoints CORS Exposure	Medium	In Progress
V-009	Insecure Electron Policies	Medium	In Progress
V-010	Debug Mode Enabled in Production	Medium	In Progress
V-011	Naive Reachability Check	Medium	In Progress
V-012	API Information Disclosure	Low	In Progress

Details of Findings

V-001: Relay IP Address Denial of Service

Severity	Remediation
High	In Progress

Discussion

The MassBrowser Operator server is prone to a severe denial of service attack from a malicious relay. Subgraph was able to cause a network-wide denial of service using an unauthenticated hostile relay. The denial of service persists until manual intervention.

Relays can publish host/port endpoint pairs that are distinct from the network origin of their connection to the Operator. This is to facilitate situations where some port forwarding configuration makes them reachable when they would otherwise not be, i.e. behind NAT. This is expected to be validated by the Operator performing a reachability check to ensure that there is an endpoint at the advertised address.

When a client requests a session, the Operator makes an attempt to identify a relay that satisfies the client's website categories and has optimal proximity to the client. The proximity of the relay to the client is in part determined based on data from IP address lookups in a GeoIP database.

An exception is thrown by the GeoIP library when an IP address is queried that does not exist in its database. This exception is passed up the call stack and is caught in the loop where all relays are evaluated for suitability for assignment, causing the loop to exit prematurely. Consequently, a relay will fail to be assigned, and an error reporting this will be communicated to the client. This condition will affect all clients as this loop runs for each new session and will fail every time for every user if a triggering relay is in the database.

Impact Analysis

This issue can be exploited when a relay registers with the network. There are a few simple ways to send a bad relay IP when registering:

1. Modify the client to send the bad IP address
2. Intercept requests between the relay and the Operator and change the relay IP address
3. Send the relay registration request manually via the API

It should be noted that the MassBrowser network has a health monitor in place that detected this issue. This enabled the administrators to manually remove the bad IP addresses and restore service to the network.

The denial of service condition will persist until the relay is removed from the database manually, by an intervening administrator.

Remediation Recommendations

There are two measures that can be implemented to prevent this issue. Firstly, relay IP addresses should be validated before they are submitted. Addresses such as *127.0.0.1*, *0.0.0.0*, and RFC1918 addresses should not be allowed. Nonsense or impossible values such as these should be filtered in general: a hostile relay should not be able to specify obviously false values such as those that correspond to locations in non-routeable address space.

While that may prevent this instance of denial of service, the core issue is in fact is simply failing to handle exceptions. The Operator must handle this and other exceptions gracefully, allowing for (in this case) a bad relay to be skipped over during the loop.

Exceptions raised by code from third-party libraries must be handled, especially in critical operations. The error condition that produced the exception was a minor one but caused a service outage for all users due to the location in code where it occurred.

Remediation Status

The MassBrowser developers are aware of the issue and are developing a fix to address it.

Additional Information

The following exception from Subgraph's Operator server logs is indicative of the issue manifesting:

```
Internal Server Error: /api/client/XXXXXX/sessions
Traceback (most recent call last):
  File "/srv/uois/venv/lib64/python3.5/dist-
↪  packages/django/core/handlers/exception.py", line 41, in
↪  inner
    response = get_response(request)
  File "/srv/uois/venv/lib64/python3.5/dist-
↪  packages/django/core/handlers/base.py", line 187, in
↪  _get_response
    response = self.process_exception_by_middleware(e, request)
  File
↪  "/srv/uois/venv/lib64/python3.5/dist-packages/channels/handler.py",
↪  line 240, in process_exception_by_middleware
    return super(AsgiHandler,
     ↪  self).process_exception_by_middleware(exception, request)
  File "/srv/uois/venv/lib64/python3.5/dist-
↪  packages/django/core/handlers/base.py", line 185, in
↪  _get_response
    response = wrapped_callback(request, *callback_args,
↪  **callback_kwargs)
```

```
  File "/srv/uois/venv/lib64/python3.5/dist-
↪  packages/django/views/decorators/csrf.py", line 58, in
↪  wrapped_view
    return view_func(*args, **kwargs)
  File "/srv/uois/venv/lib64/python3.5/dist-
↪  packages/rest_framework/viewsets.py", line 83, in
↪  view
    return self.dispatch(request, *args, **kwargs)
  File "/srv/uois/venv/lib64/python3.5/dist-
↪  packages/rest_framework/views.py", line 483, in
↪  dispatch
    response = self.handle_exception(exc)
  File "/srv/uois/venv/lib64/python3.5/dist-
↪  packages/rest_framework/views.py", line 443, in
↪  handle_exception
    self.raise_uncaught_exception(exc)
  File "/srv/uois/venv/lib64/python3.5/dist-
↪  packages/rest_framework/views.py", line 480, in
↪  dispatch
    response = handler(request, *args, **kwargs)
  File "/srv/uois/webserver/session/views.py", line 51, in create
    relay, session_type =
↪  LessDummyDistanceBasedRelayAssigner().get_relay(client, categories)
  File "/srv/uois/webserver/session/relay_assigners.py", line 78, in
↪  get_relay
    score = 1.0 / (1.0 + self.get_distance(client, relay))
  File "/srv/uois/webserver/session/relay_assigners.py", line 62, in
↪  get_distance
    client_loc = GEODB.city(client.ip).location
  File
↪  "/srv/uois/venv/lib64/python3.5/dist-packages/geoip2/database.py",
↪  line 114, in city
    return self._model_for(geoip2.models.City, 'City', ip_address)
  File
↪  "/srv/uois/venv/lib64/python3.5/dist-packages/geoip2/database.py",
↪  line 194, in _model_for
    (record, prefix_len) = self._get(types, ip_address)
  File
↪  "/srv/uois/venv/lib64/python3.5/dist-packages/geoip2/database.py",
↪  line 190, in _get
    "The address %s is not in the database." % ip_address)
geoip2.errors.AddressNotFoundError: The address 127.0.0.1 is not in the
↪  database.
```

```
2020-02-19 18:29:01,886 - ERROR - worker - Error processing message with
↪  consumer uois.channels.WSConsumer:
Traceback (most recent call last):
  File "/srv/uois/venv/lib64/python3.5/dist-
↪  packages/kombu/utils/functional.py", line 43, in
↪  __call__
    return self.__value__
AttributeError: 'ChannelPromise' object has no attribute '__value__'
```

V-002: Adversary Procurement of Trusted Weak Certificates for Arbitrary Websites

Severity	Remediation
High	In Progress

Discussion

The CacheBrowser proxy built into the MassBrowser client performs TLS interception for supported domains. When a domain is requested at the client SOCKS5 proxy that fits the policy for handling by the CacheBrowser proxy, that incoming connection request is passed to the CacheBrowser proxy service listening on TCP port 6425.

CacheBrowser relies on the Server Name Indication (SNI) TLS extension, extracting the *Server Name* field of the client's TLS *Client Hello* to identify the domain the user intends to access. Incoming client requests cause a certificate to be created dynamically for that domain. This certificate is signed by the Certificate Authority (CA) certificate that users add to their browser during installation of the MassBrowser client.

A certificate and RSA-1024 keypair will be created for *any* domain when a client connects directly to the CacheBrowser proxy service. The check for domain support for CacheBrowser is expected to occur in the SOCKS5 proxy policy logic. These certificates are based on an RSA-1024 keypair and have an expiry set to 2 years from the moment they're created. This is risky as the CacheBrowser effectively becomes a universal CA for the client that prints arbitrary trusted weak certificates, though they are expected to be used only internally and exist ephemerally.

Due to **V-007** this is not the case. The CacheBrowser proxy will bind to *0.0.0.0/0* (INADDR_ANY) by default. Therefore a host with network reachability to the client can connect to the proxy service and procure the trusted certificate that is generated for the domain specified in the SNI field. To reiterate, these certificates are generated using a weak RSA modulus and have an expiry date of 2 years. It should be noted that most clients will be on a LAN where they aren't reachable by the Internet, such as their home, work, or school network.

Impact Analysis

If the adversary is able to complete the entire attack, including factoring the RSA private key, man-in-the-middle (MITM) attacks against the user and target website are possible if the user is still using the same browser with the MassBrowser CA certificate installed.

Certain types of adversaries may be in a position to plausibly exploit this vulnerability to procure weak certificates for specific target domains. DSL and cable modem devices provided by Internet providers often perform NAT for the user LAN, and control over these devices would provide such reachability into internal networks.

As this certificate is signed by a trusted CA configured by the user when they installed MassBrowser, it could be used later in an active MITM attack against the user if the RSA-1024 is broken; the 2-year expiry gives the adversary time to attempt the prime factorization that would be required to exploit the weak certificate. While this is a theoretical attack, factoring RSA-1024 is considered feasible for adversaries with state-level capabilities.

Remediation Recommendations

The CacheBrowser certificate manager should not be so generous with printing certificates on-demand. These are dangerous artifacts to create and utilize so casually; a CacheBrowser policy check must be performed on the domain prior to runtime certificate generation and only those certificates that would actually used for CacheBrowser sessions should be generated.

If a domain in the SNI isn't found to be supported for CacheBrowser sessions, a default, perhaps even invalid, X.509 certificate for a non-routeable domain can be produced by the TLS server.

The CacheBrowser proxy should not be bound to *0.0.0.0/0*, there is no apparent reason for this that Subgraph could identify.

The CacheBrowser certificate manager should consider 2048 RSA modulus size for website certificates.

The MassBrowser team should consider a shorter expiry than 2 years; there is no reason for such a long life as these certificates are anyways intended to be used for a single browser session per the current design.

Remediation Status

In progress.

Additional Information

This finding can be demonstrated by running the following command (note: the servername argument corresponds to the SNI field):

```
openssl s_client -verifyCAfile ~/.massbrowser/certs/ca.pem -servername
↪  subgraph.com --connect localhost:6425
```

```
(realm-pentest):client ▸  openssl s_client -verifyCAfile ~/.massbrowser/certs/ca.pem -servername subgraph.com --connect localhost:6425
CONNECTED(00000003)
depth=0 CN = subgraph.com, C = US, ST = MA, L = Amherst, O = MassBrowser, OU = MassBrowser
verify error:num=66:EE certificate key too weak
verify return:1
depth=1 CN = massbrowser.cs.umass.edu, C = US, ST = MA, L = Amherst, O = MassBrowser, OU = MassBrowser
verify return:1
depth=0 CN = subgraph.com, C = US, ST = MA, L = Amherst, O = MassBrowser, OU = MassBrowser
verify return:1
---
Certificate chain
 0 s:CN = subgraph.com, C = US, ST = MA, L = Amherst, O = MassBrowser, OU = MassBrowser
   i:CN = massbrowser.cs.umass.edu, C = US, ST = MA, L = Amherst, O = MassBrowser, OU = MassBrowser
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDdjCCAl6gAwIBAgIQfAzRF8JXHi5O0uvBVNGPYjANBgkqhkiG9w0BAQsFADB7
MSEwHwYDVQQDExhtYXNzYnJvd3Nlci5jcy51bWFzcy5lZHUxCzAJBgNVBAYTAlVT
MQswCQYDVQQIEwJNQTEQMA4GA1UEBxMHQW1oZXJzdDEUMBIGA1UEChMLTWFzc0Jy
b3dzZXIxFDASBgNVBAsTC01hc3NCcm93c2VyMB4XDTIwMDMwMjE3MjUyMloXDTIy
MDMwMzE3MjUyMlowbzEVMBMGA1UEAxMMc3ViZ3JhcGguY29tMQswCQYDVQQGEwJV
UzELMAkGA1UECBMCTUExEDAOBgNVBAcTB0FtaGVyc3QxFDASBgNVBAoTC01hc3NC
cm93c2VyMRQwEgYDVQQLEwtNYXNzQnJvd3NlcjCBnzANBgkqhkiG9w0BAQEFAAOB
jQAwgYkCgYEAgWLnH6QM1cubftBwvmqr9z/FUqBUe5/GU3vmIbUlmXnjVe9U//iq
GKGK+Vgi+6KEtSn5o6oOusZtDfadC1HyeOREUKre1O6t34gU83CPL8kKBLqW2o9q
J3g10YJm6KwVpkkNQFjfDH16X7LzsFQxfHgjwTbKXyniV/BeHPfjDy8CAwEAAaOB
hTCBgjAJBgNVHRMEAjAAMAsGA1UdDwQEAwIEsDAdBgNVHSUEFjAUBggrBgEFBQcD
AQYIKwYBBQUHAwIwEQYJYIZIAYb4QgEBBAQDAgbAMB0GA1UdDgQWBBS0j56MvkzR
Rmxc+RWG3jeJUuPbBzAXBgNVHREEEDAOggxzdWJncmFwaC5jb20wDQYJKoZIhvcN
AQELBQADggEBAJ+Wtyc6MoGHl2g2CiKZ23q30d8lBadPlstLrqxMzfO6sHwk8YHS
RuJeTlw6BAw3CbZjKrAR2aMbrE+ny+g/OBvuWTymuIT3kkt1pxLgw2DFKPjEvkpv
cr+95A+6+hihX2KzL6XoHFfBLDgTJqJAQ8rv2mDrMsXxW9t9UJuBumlE894xiOk1
RRjvaemb5N2x0Snh9UX4QTU0ISYSE9gd6fu56cTdnM6J7BE7qOQ1o9yFu6NOX9NY
vRqPl4Z5oV8qrHs/wv2WrunjHrRW4+2b2RIVRcfgCJ2dJpkySJ9wzTVxwBxTYf+v
+LhGu78tj5KOu4UqC5aMS/RHQm886Ue5drg=
-----END CERTIFICATE-----
subject=CN = subgraph.com, C = US, ST = MA, L = Amherst, O = MassBrowser, OU = MassBrowser

issuer=CN = massbrowser.cs.umass.edu, C = US, ST = MA, L = Amherst, O = MassBrowser, OU = MassBrowser

---
No client certificate CA names sent
Peer signing digest: SHA512
Peer signature type: RSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 1435 bytes and written 397 bytes
Verification error: EE certificate key too weak
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 1024 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID: 596515B9C058644FC68F0768AA0CA26F52C1F0793030280CBA0A05521B7DF0A8
    Session-ID-ctx:
    Master-Key: 2186AD8F6C556E1FEA20DCC1D686A8B0531B837EB9CBC017516C3C98A3B53906D7CB5A2075A57BE0AC5CFF536A277D07
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 7200 (seconds)
    TLS session ticket:
    0000 - aa 13 4f 82 c5 99 f3 d8-67 e3 49 49 98 33 3c d9   ..O.....g.II.3<.
    0010 - d7 cb 8d e0 b4 58 b2 e1-df cf 1e 69 09 63 bf 13   .....X.....i.c..
    0020 - fb 01 94 45 ac b6 85 1b-21 d2 02 56 c4 25 0e b1   ...E....!..V.%..
    0030 - 6d 10 5b 89 c7 ac 6b 16-01 27 2a 71 4c 1c 9f 01   m.[...k..'*qL...
    0040 - 00 60 0a 12 9b 8f 27 91-cf 3c 2f 21 2d 8c bb 86   .`....'..</!-...
    0050 - 3a cf e6 d3 5c dc 24 f7-e7 bf 6a be 47 fb da 3c   :...\.$...j.G..<
    0060 - 17 aa 63 fe ce 4e 3e 81-c4 f9 2c 55 2a 6f 48 9b   ..c..N>...,U*oH.
    0070 - bb 56 3f 82 d1 62 0d c9-79 49 80 9b bd 36 18 17   .V?..b..yI...6..
    0080 - 99 0f 49 e6 d6 58 62 38-b7 d4 71 6a 02 0d 2e 36   ..I..Xb8..qj...6
    0090 - d8 a0 96 74 55 03 60 81-a1 53 9c 79 17 dc 40 72   ...tU.`..S.y..@r
    00a0 - 08 71 9a 8c 68 5f 6b 8a-fc 5b ee 94 03 7d 7e 72   .q..h_k..[...}~r
    00b0 - d3 a9 29 c3 3b 49 d1 63-61 45 a0 98 29 e0 26 3d   ..).;I.caE..).&=
    00c0 - c2 33 51 1f 52 4d 15 7e-62 cb ac 68 0f aa f0 8a   .3Q.RM.~b..h....
```

V-003: API Cross-Site Request Forgery

Severity	Remediation
High	In Progress

Discussion

Subgraph discovered that the Operator server API is prone to cross-site request forgery.

Cross-site Request Forgery (CSRF) is a vulnerability that lets forms be submitted from a different origin other than the site hosting the API. The forms will be submitted with the cookie-based credentials of logged in users.

This issue occurs specifically because the CSRF prevention mechanism of Django is disabled by default in the Operator server API (at least in our deployment). This causes the `csrfmiddlewaretoken` form field to be ignored instead of validated. When the prevent mechanism is enabled, this field will contain a unique token that must be submitted with any request that modifies data through the API (POST, PUT, DELETE). Requests that omit or contain invalid tokens will be rejected under this policy.

This issue only affects endpoints for the `/api` path and not the `/admin` path. However, the most likely avenue of attack is against a logged in administrator since the same cookie-based credentials are used for both `/admin` and `/api`.

The CSRF prevention appears to be disable in the `uauth` module via the following lines of code:

```
from rest_framework.authentication import SessionAuthentication

class CsrfExemptSessionAuthentication(SessionAuthentication):

    def enforce_csrf(self, request):
        return
```

Impact Analysis

To exploit the issue, the attacker will create a malicious form that is designed to perform actions via the API. This might include creating, updating, or deleting data in the Operator. The form will also be designed to be submitted in the background without user interaction.

The attacker then entices administrators to visit their site that hosts the malicious form. If the victim is logged in, the form will be submitted with their cookie-based credentials and upon successful submission the malicious actions will be performed.

One possible way to entice an administrator to visit the form is via the feedback mechanism of the application.

A possible attack is to perform actions in the API that can cause a denial of services – such as deleting data or populating the Operator with bad data.

Remediation Recommendations

Keeping the admin console safe from CSRF style attacks while sharing a back-end REST API with non-browser consumers will require enhancement of the authorization model used.

Roles intended for human / browser console users and for API consumers such as clients and relays must be partitioned.

The REST endpoints can then differentiate between the browser-based access, where Django CSRF token validation is mandatory, and non-browser access, where not mandatory.

Remediation Status

In progress.

Additional Information

N/A

V-004: User Identification and Tracking by Silent Remote CacheBrowser CA Certificate Retrieval

Severity	Remediation
High	In Progress

Discussion

Users can be identified by their uniquely generated CA certificates which can be retrieved trivially by any website or active adversary who is in a MITM position relative to the target user, or by a malicious website that the user visits by chance or through some other means.

Certificates are remotely retrievable due to the open Cross-Origin Resource Sharing (CORS) Allow-Origin policy set on the endpoint '/cert' exposed by the client's local management interface:

See *app/src/client/services/webPanelService.js*:

```
app.use('/cert', function (req, res, next) {
  res.setHeader('Access-Control-Allow-Origin', '*')
  res.setHeader('Content-Type', 'application/x-x509-ca-cert')
  fs.readFile(path.join(getDataDir(), 'certs/ca.pem'))
  .then(f => {
    console.log(f)
    res.end(f)
    next()
  })
})
```

More information about CORS can be found here:

- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Impact Analysis

An adversary can uniquely identify MassBrowser users. The unique and long-lived certificates would serve as a kind of 'super-cookie' that will permit persistent fingerprinting and tracking.

A more worrisome thought is that an adversary would be able to match a user identified on a network to the fingerprint stored on their computer system for a positive physical identification.

This attack can be performed most effectively by an attacker with MITM interception capability. If the attack does not possess this capability, the user must be convinced or coerced into visiting a HTTP site under the control of the attacker.

The purpose of this application, ie: circumvention, increases the risk that this issue poses.

Remediation Recommendations

Universal cross-origin access to internal endpoints should not be permitted.

Remediation Status

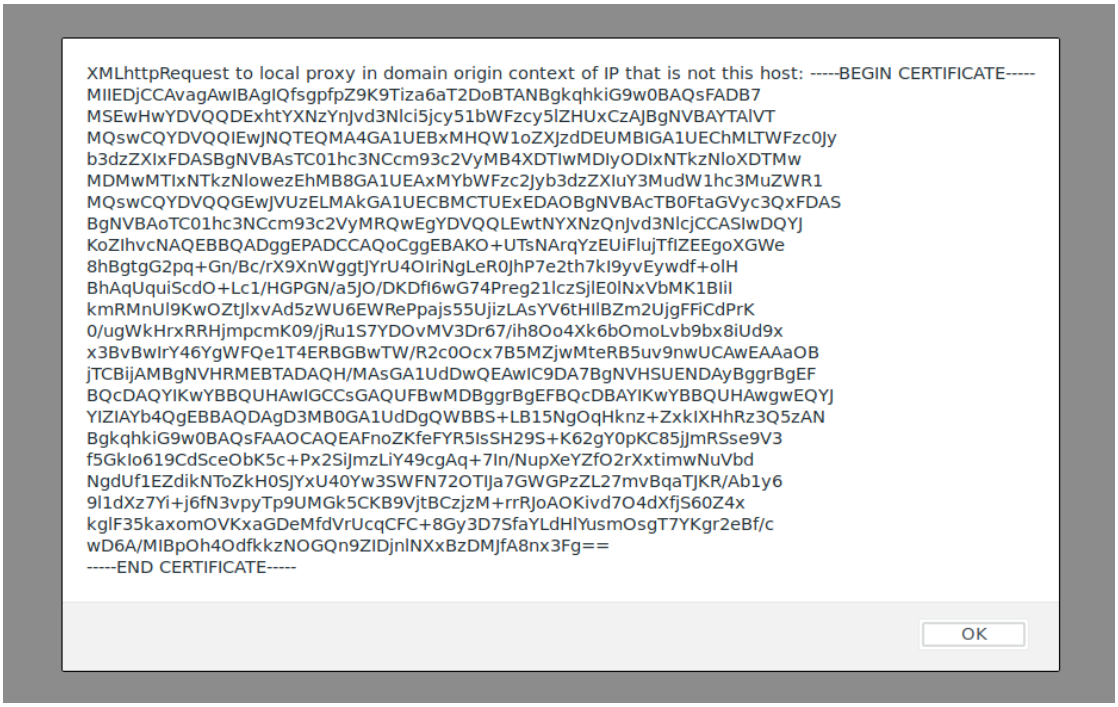
In progress.

Additional Information

The following proof of concept will demonstrate the issue if hosted on an HTTP server and visited by a browser configured to use the MassBrowser proxy:

```
<html>
        <body>
        <script>
                var xmlhttp = new XMLHttpRequest();
                var method = 'GET';
                var url = 'http://localhost:6423/cert';

                xmlhttp.open(method, url, true);
                xmlhttp.onerror = function () {
                        console.log("** An error occurred during the
                        ↪  transaction");
                };
                xmlhttp.onreadystatechange = function () {
                        if (xmlhttp.readyState === XMLHttpRequest.DONE
                        ↪  && xmlhttp.status === 200) {
                                alert("Cert: "+xmlhttp.responseText);
                        }
                };
                xmlhttp.send();
        </script>
        </body>
</html>
```


XMLhttpRequest to local proxy in domain origin context of IP that is not this host: -----BEGIN CERTIFICATE-----
MIIEDjCCAvagAwIBAgIQfsgpfpZ9K9Tiza6aT2DoBTANBgkqhkiG9w0BAQsFADB7
MSEwHwYDVQQDExhtYXNzYnJvd3Nlci5jcy51bWFzcy5lZHUxCzAJBgNVBAYTAlVT
MQswCQYDVQQIEwJNQTEQMA4GA1UEBxMHQW1oZXJzdDEUMBIGA1UEChMLTWFzc0Jy
b3dzZXIxFDASBgNVBAsTC01hc3NCcm93c2VyMB4XDTIwMDIyODIxNTkzNloXDTMw
MDMwMTIxNTkzNlowezEhMB8GA1UEAxMYbWFzc2Jyb3dzZXIuY3MudW1hc3MuZWR1
MQswCQYDVQQGEwJVUzELMAkGA1UECBMCTUExEDAOBgNVBAcTB0FtaGVyc3QxFDAS
BgNVBAoTC01hc3NCcm93c2VyMRQwEgYDVQQLEwtNYXNzQnJvd3NlcjCCASIwDQYJ
KoZIhvcNAQEBBQADggEPADCCAQoCggEBAKO+UTsNArqYzEUiFlujTflZEEgoXGWe
8hBgtgG2pq+Gn/Bc/rX9XnWggtJYrU4OIriNgLeR0JhP7e2th7kI9yvEywdf+olH
BhAqUquiScdO+Lc1/HGPGN/a5JO/DKDfl6wG74Preg21lczSjlE0lNxVbMK1BiiI
kmRMnUl9KwOZtJlxvAd5zWU6EWRePpajs55UjizLAsYV6tHllBZm2UjgFFiCdPrK
0/ugWkHrxRRHjmpcmK09/jRu1S7YDOvMV3Dr67/ih8Oo4Xk6bOmoLvb9bx8iUd9x
x3BvBwIrY46YgWFQe1T4ERBGBwTW/R2c0Ocx7B5MZjwMteRB5uv9nwUCAwEAAaOB
jTCBijAMBgNVHRMEBTADAQH/MAsGA1UdDwQEAwIC9DA7BgNVHSUENDAyBggrBgEF
BQcDAQYIKwYBBQUHAwIGCCsGAQUFBwMDBggrBgEFBQcDBAYIKwYBBQUHAwgwEQYJ
YIZIAYb4QgEBBAQDAgD3MB0GA1UdDgQWBBS+LB15NgOqHknz+ZxkIXHhRz3Q5zAN
BgkqhkiG9w0BAQsFAAOCAQEAFnoZKfeFYR5IsSH29S+K62gY0pKC85jJmRSse9V3
f5GkIo619CdSceObK5c+Px2SiJmzLiY49cgAq+7In/NupXeYZfO2rXxtimwNuVbd
NgdUf1EZdikNToZkH0SJYxU40Yw3SWFN72OTIJa7GWGPzZL27mvBqaTJKR/Ab1y6
9l1dXz7Yi+j6fN3vpyTp9UMGk5CKB9VjtBCzjzM+rrRJoAOKivd7O4dXfjS60Z4x
kglF35kaxomOVKxaGDeMfdVrUcqCFC+8Gy3D7SfaYLdHlYusmOsgT7YKgr2eBf/c
wD6A/MIBpOh4OdfkkzNOGQn9ZIDjnlNXxBzDMJfA8nx3Fg==
-----END CERTIFICATE-----
OK

V-005: CacheBrowser Weak Server Certificate Generation

Severity	Remediation
Medium	In Progress

Discussion

CacheBrowser dynamically generates certificates for client requested domains that are routed to it per the client policy. These certificates are produced using an RSA key with a 1024-bit modulus generated randomly at the time of the website request.

1024-bit RSA is considered weak and is being phased out of all use globally. For example, OpenSSL as configured system-wide by default on Debian 10 will not validate a certificate for this reason.

The following client code is used to generate the certificate dynamically:

```
var keysServer = pki.rsa.generateKeyPair(1024)
var certServer = pki.createCertificate()
certServer.publicKey = keysServer.publicKey
certServer.serialNumber = this.randomSerialNumber()
certServer.validity.notBefore = new Date()

↪  certServer.validity.notBefore.setDate(certServer.validity.notBefore.getDate()
↪  - 1)
certServer.validity.notAfter = new Date()

↪  certServer.validity.notAfter.setFullYear(certServer.validity.notBefore.getFull
↪  + 2)
```

Impact Analysis

The impact of this issue would be low if these certificates were truly ephemeral, but they are not, as they can be generated by other parties due to the exposure of the CacheBrowser proxy port and their 2-year expiry. This highlights the risk of using trusted X.509 certificates so casually.

Remediation Recommendations

- Generate 2048 RSA keypairs
- Only generate certificates for qualifying domains
- Reduce the expiry time

Remediation Status

In progress.

Additional Information

N/A

V-006: Client Denial of Service

Severity	Remediation
Medium	In Progress

Discussion

Subgraph has identified a vulnerability that makes it possible for a malicious website to crash the MassBrowser client. This attack can be triggered by an adversary with MITM capability at any time, or passively, if a target user visits a malicious website. An attacker with point-to-point network proximity to the target client can also exploit this vulnerability.

The issue is due an unhandled exception thrown when there is a direct client connection to the CacheBrowser proxy port that does not correspond to a connection registered by the primary proxy interface. The CacheBrowser proxy attempts to query internal metadata about such connections without any consideration for the possibility that the metadata wasn't stored. The exception in that case is fatal when it is eventually caught.

Affected code from *app/src/client/cachebrowser/cacheProxy.js*:

```
handleCacheSocket (socket) {
  [..]
  let cachesocketoptions = {
    host: this.connectionlist[socket.remotePort][0],
    port: this.connectionlist[socket.remotePort][1],
    servername: this.connectionlist[socket.remotePort][2],
    rejectUnauthorized: false
  }
  [..]
```

The following exception is thrown because *this.connectionlist* does not contain the data that's expected due to the connection being made directly rather than setup by the SOCKS5 proxy interface:

```
err uncaught Exception  :  TypeError: Cannot read property '0' of
↪  undefined
   at CacheProxy.handleCacheSocket
↪  (/home/user/mb/MassBrowser/app/dist/client/client.js:17777:53)
   at Server.<anonymous>
↪  (/home/user/mb/MassBrowser/app/dist/client/client.js:17727:17)
   at Server.emit (events.js:198:13)
   at TLSSocket.onSocketSecure (_tls_wrap.js:745:29)
   at TLSSocket.emit (events.js:198:13)
   at TLSSocket._finishInit (_tls_wrap.js:636:8)
   at TLSWrap.onhandshakedone (_tls_wrap.js:103:9)
```

This can be triggered by a third-party website simply by it directing the client to make an HTTPS request to the loopback service (itself).

As the service listens on *0.0.0.0/0*, a host with network reachability to the client can also trigger the crash by simply connecting to it.

Impact Analysis

A third-party can cause the MassBrowser client to crash, requiring a manual restart.

Remediation Recommendations

The exception should be handled gracefully rather than allowed to propagate higher up the call stack.

Remediation Status

In progress.

Additional Information

The following HTML page demonstrates the issue being exploited by a website. The page will cause the client to crash when visited by a MassBrowser proxied browser:

```
<html>
    <body>
        <img src="https://localhost:6425/sup.png">
    </body>
</html>
```

V-007: Proxy Services Unnecessarily Bind to INADDR_ANY

Severity	Remediation
Medium	In Progress

Discussion

Subgraph observed that the client proxy services bind to *0.0.0.0/0* (INADDR_ANY) on Linux systems and likely elsewhere. This means that the proxy ports are exposed to other hosts on a network that has end-to-end network reach to the client. On the Internet, this would mean any host on the Internet could communicate with the client proxy services. In practice this is highly unlikely to ever occur. The exposure is almost always going to be limited to hosts in the same LAN address space as the client: home network, office network, school network, etc.

This is due to the use of NodeJS library functions such as *tls.Server.listen()* which will bind to *0.0.0.0/0* by default. An example from *app/src/client/cacheBrowser/cacheProxy.js*:

```
[..]
      this.proxyserver = tls.createServer(this.proxyoptions, (socket) =>
      ↪  {
        // console.log('new Connection')
        this.handleCacheSocket(socket)
      })

      this.proxyserver.listen(config.cachebrowser.mitmPort, () => {
        started = true
        debug('Initializing certificate manager')
        certificateManager.initializeCA().then(() => {
          resolve()
        })
      })
[..]
```

Subgraph doesn't see any reason for most of the local services used within the client to bind to INADDR_ANY in most of the following cases:

```
./cachebrowser/cacheProxy.js:
↪  this.proxyserver.listen(config.cachebrowser.mitmPort, () => {
./net/NetworkManager.js:      server.listen({port: publicPort, host:
↪  '0.0.0.0', exclusive: false}, () => {
./net/NetworkManager.js:            server.listen({port: publicPort,
↪  host: '0.0.0.0', exclusive: false}, () => {
./net/clientSocks.js:          server.listen(PORT, HOST)
```

```
./net/clientSocks.js:          second_server.listen(SECOND_PORT, HOST)
./net/clientSocks.js:    server.listen(PORT, HOST)
./net/clientSocks.js:    second_server.listen(SECOND_PORT, HOST)
./services/noHostHandlerService.js:    this.server.listen(port, () =>
↪  debug(`No-host handler started on port ${port}`))
```

```
electron 20850 user   89u  IPv6 1217972      0t0  TCP *:6425 (LISTEN)
electron 20850 user   92u  IPv4 1217975      0t0  UDP *:16788
electron 20850 user   94u  IPv4 1217976      0t0  UDP *:44655
electron 20850 user   97u  IPv4 1217978      0t0  TCP *:15233 (LISTEN)
electron 20850 user  103u  IPv6 1215048      0t0  TCP *:6423 (LISTEN)
```

Interestingly there is one instance where the host is explicitly supplied as an option to the *listen()* method. This is for the client listener that is intended to be publicly exposed in the case that it is routeable on the Internet.

NodeJS documentation for the server.listen() methods:

- https://nodejs.org/api/net.html#net_server_listen

Impact Analysis

Two potential impacts of this have been identified:

- An adversary with network reachability can crash the GUI client
- An adversary with network reachability can forge weak but trusted certificates for arbitrary domains

Adversaries with network reachability to clients would be:

- The entire Internet, if the client is at a routeable endpoint, though this is unlikely
- Other hosts on the same LAN
- The provider of network equipment that controls NAT, as is the case for many home/small business cable and DSL modems

There may be other ways to impact the client adversely.

Fundamentally, we could not identify any good reason why most of these ports need to face the client's external network since all incoming proxy connection requests should come from the same host.

Remediation Recommendation

Ensure that the proxy services bind to the loopback interface address on all supported platforms. This would be achieved by ensuring that the loopback address is passed as an option to *server.listen()* or *tls.Server.listen()*.

Remediation Status

In progress.

Additional Information

N/A

V-008: Internal Endpoints CORS Exposure

Severity	Remediation
Medium	In Progress

Discussion

Several of the MassBrowser internal client API endpoints are exposed to third-party websites who can access and interact with them. This is due to the CORS Allow-Origin policy set for them.

The endpoints affected are the following:

- The "No host handler" that responds for any non-hostname URI
- http://localhost:6422/v2
- http://localhost:6423/check-proxy
- http://localhost:6423/cert
- http://localhost:6423/settings-complete

Each of these sends CORS headers in response similarly, e.g.:

```
initializeApp(app) {
  app.use('/check-proxy', function (req, res) {
    res.setHeader('Access-Control-Allow-Origin', '*')
    res.setHeader('Content-Type', 'text/plain')
    res.end('active')
  })
```

This means that a third-party website can use the browser to issue XMLHTTPRequest requests and read the response body without violating the *same-origin policy*.

Impact Analysis

An attack described as **V-004** leverages this issue to uniquely fingerprint and track MassBrowser users.

Adversaries can also silently detect that MassBrowser is being used and enabled.

Remediation Recommendations

The MassBrowser team should re-evaluate the decision to set such a permissive CORS policy for these endpoints, and then narrow the policy to the minimum required.

Remediation Status

In progress.

Additional Information

A proof of concept to check the user's MassBrowser proxy status remotely follows:

```
<html>
    <body>
    <script>
        var xmlhttp = new XMLHttpRequest();
        var method = 'GET';
        var url = 'http://localhost:6423/check-proxy';

        xmlhttp.open(method, url, true);
        xmlhttp.onerror = function () {
            console.log("** An error occurred during the transaction");
        };
        xmlhttp.onreadystatechange = function () {
            if (xmlhttp.readyState === XMLHttpRequest.DONE &&
            ↪  xmlhttp.status === 200) {
                    alert("XMLhttpRequest to local proxy in domain
                    ↪  origin context of IP that is not this host:
                    ↪  "+xmlhttp.responseText);
            }
        };
        xmlhttp.send();
    </script>
    </body>
</html>
```

V-009: Insecure Electron Policies

Severity	Remediation
Medium	In Progress

Discussion

The client and relay Electron user interfaces have insecure security policies.

If a cross-site scripting vulnerability is discovered in the Electron applications, these insecure security policies could be exploited to execute arbitrary code on the client or relay host. Subgraph audited the `Vue.js` user interface code for cross-site scripting issues but did not find any. However, unless the insecure policies are addressed, they present a serious risk if cross-site scripting vulnerabilities are introduced at a later time or are discovered via some mechanism that Subgraph is not aware of.

The following specific issues are present:

- Node.js Integration with Remote Content
- Insecure Resources
- Insecure Content-Security-Policy

The most serious concern is "Node.js Integration with Remote Content" as this will permit remote content to call into Node.js libraries – which can be exploited to execute arbitrary code on the local computer if a cross-site scripting issue is present.

"Insecure Content-Security-Policy" is also potentially serious because the lack of a policy permits `unsafe-eval` – which can introduce cross-site scripting vectors via JavaScript `eval` and similar methods such as `setTimeout`, `setInterval`, etc.

"Insecure Resources" indicates that some resources are loaded from non-HTTPS origins – in this case JS/CSS are loaded from a local application service. The affected resources are not risky in themselves but since mixed secure and non-secure content is allowed then it may be theoretically possible for an adversary to get malicious remote content loaded in the application – which is worsened by the other insecure policies.

This issue was discovered in the Linux version that we built from source. The Mac OS X and Windows versions were not tested.

Impact Analysis

These issues present latent security problems to the application. If a cross-site scripting vulnerability is discovered or an attacker is able to get malicious remote content loaded in the application, it may be possible to execute arbitrary code.

Remediation Recommendations

Guidelines on how to address these issues can be found here:

- https://www.electronjs.org/docs/tutorial/security

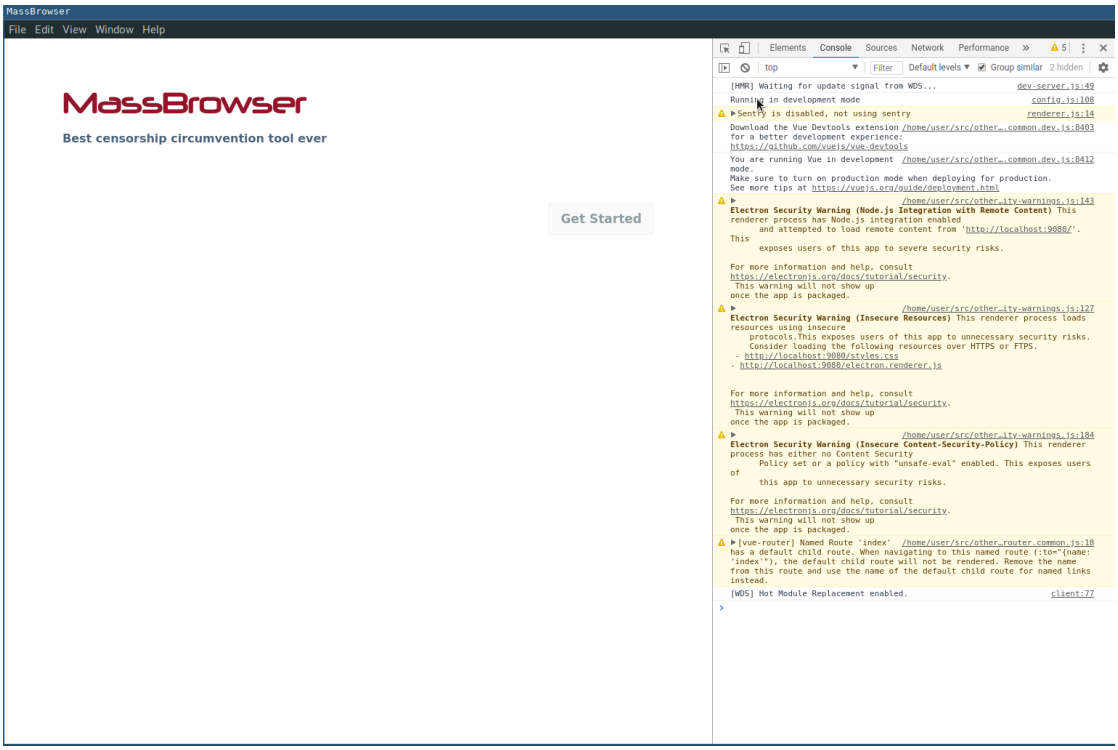
One unique feature of MassBrowser is that it creates a certificate authority and creates certificates in relation to the CacheBrowser feature. To avoid loading insecure local resources, it may be possible to host the local services via HTTPS using the certificate authority. There also may be other ways of embedding the content in the application other than loading it over HTTP.

Remediation Status

In progress.

Additional Information

The following screenshot depicts the Electron security warnings:



V-010: Debug Mode Enabled in Production

Severity	Remediation
Medium	In Progress

Discussion

Debug mode is enabled in the production instance of the Operator server running at `backend.yaler.co`. This can reveal sensitive information about the application and its host environment that can be used in further attacks.

Impact Analysis

Attackers can gain access to sensitive information about the application and its host environment, particularly when submitting data that causes exceptions.

Remediation Recommendations

Disable debug mode in production.

Remediation Status

In progress.

Additional Information

The following screenshot shows full debug information in production:


ValueError at /api/website/request
Cannot assign "<user: models.UAnonymousUser object at 0x7f0fcc714940>": "WebsiteRequest.user" must be a "User" instance.
Traceback
Request information
USER
GET
POST
FILES
COOKIES
META

V-011: Naive Reachability Check

Severity	Remediation
Medium	In Progress

Discussion

Clients and relays can publish arbitrary Internet-routeable endpoints. This is to facilitate situations where the client and relay listeners are situated behind NAT but have port forwarding of some kind enabled allowing for incoming point-to-point connections from the Internet.

The MassBrowser Operator will run a Celery worker task that performs a simple TCP connect to establish whether an endpoint is reachable or not. Reachability is determined solely by the success of this connection. This is fairly weak: a malicious client or relay can supply an address and port that are known to be open, which is sufficient for the Operator to believe that the endpoint is reachable.

Example of the reachability check performed for clients who advertise reachable endpoints from *web-server/client/tasks.py* in the Operator source tree:

```
@shared_task
def check_reachability(client_pk,uid=-1):
    try:
[..]
        client.tcp_reachable= canConnect(client.ip,client.port)
        client.save()
        check_reachability.apply_async(args=(client.pk,uid),
↪   countdown=CHECKCONNECT_PERIOD)

        print ('session Created')
    except Exception as E:
        print (E,'Cannot Check Connectivity')

[..]
def canConnect(ip,port):
    s=socks.socksocket()
    # s.set_proxy(socks.SOCKS5,SOCKS_HOST,SOCKS_PORT)
    print ("Connecting to %s:%d"%(ip,port))
    try:
        s.connect((ip,port))
        s.close()
    except:
        print("Cannot connect to %s:%d" % (ip, port))
```

```
        return False
    print("Connected to %s:%d" % (ip, port))
    return True
[..]
```

A similar reachability check is performed for relays, which is what Subgraph originally tested. This is because relays are intuitively expected to be reachable more often. It's worth noting that this code appears to be still under development and/or testing, as evident elsewhere in the codebase, and slightly different behavior was observed related to this functionality in the running Operator than what appeared to be specified in the code reviewed.

Subgraph was exploring the consequences of fooling the reachability check for a hostile relay that would publish endpoints at *127.0.0.1:5558* and *127.0.0.1:22*, both assumed to pass this check. That bad information from a single relay triggered a network-wide denial of service due to a separate issue documented as **V-001**.

Impact Analysis

This may aid in other attacks that depend on an endpoint being considered reachable.

Relays in particular are high-risk participants in the MassBrowser network, as they are not authenticated and are open to public enrollment.

We foresee other issues related to trusting information such as this from relays and clients. We recommend that the MassBrowser team further examine the risks and always design with the assumption that either participants may behave in unexpected ways.

Remediation Recommendations

The Operator should use a stronger mechanism to confirm that it is in communication with a real MassBrowser endpoint (client or relay). MassBrowser endpoints should be expected to provide proof that they are what they are advertised to be, and not random services/hosts on the Internet.

A MassBrowser protocol enhancement could address this issue.

Remediation Status

In progress.

Additional Information

N/A

V-012: API Information Disclosure

Severity	Remediation
Low	In Progress

Discussion

The API is configured to run in a test mode that allows users to make requests directly to the API web page. This includes forms for POST requests, etc. In these forms are drop-downs that include a list of possible values from the database such as user identifiers, invitation codes, etc.

This information may aid in further attacks – such as the CSRF issue described elsewhere in this report.

Impact Analysis

Attackers may abuse the information in other attacks against the Operator server and its users.

Remediation Recommendations

More information about the potential cause of this issue and some resolutions can be found here:

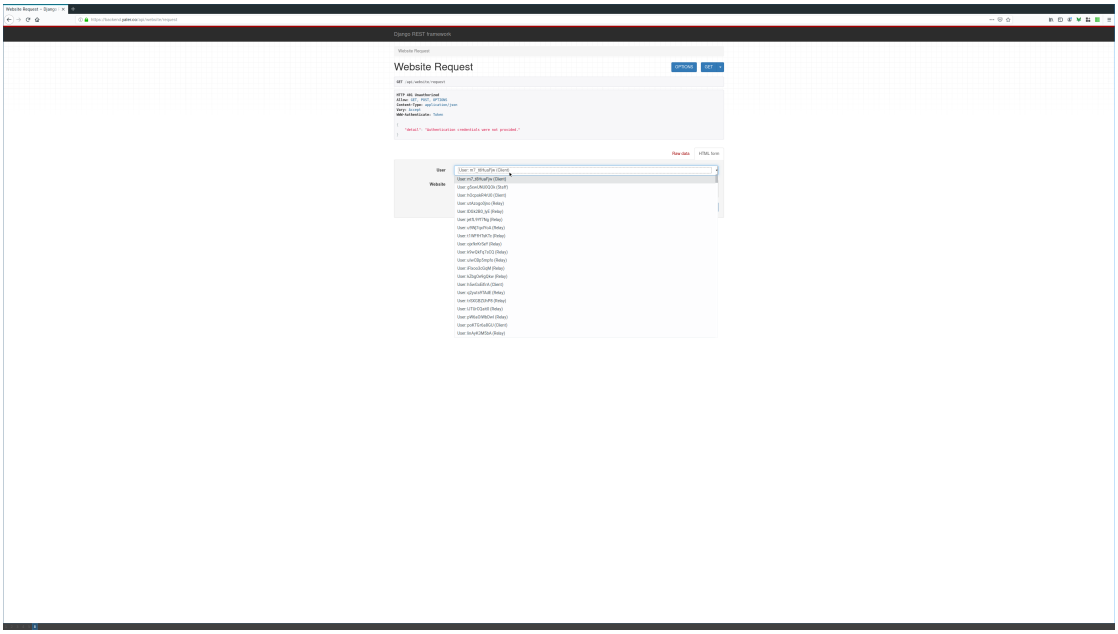
https://github.com/encode/django-rest-framework/issues/3905

Remediation Status

In progress.

Additional Information

The following screenshot shows an API drop-down with user identifiers:


Website Request - Django
Django REST framework
Website Request
Website Request
OPTIONS
GET
GET /api/website/request
HTTP 401 Unauthorized
Allow: GET, POST, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Token
{
"detail": "Authentication credentials were not provided."
}
Raw data
HTML form
User
Website

Appendix

Methodology

Our approach to testing is designed to understand the design, behavior, and security considerations of the assets being tested. This helps us to achieve the best coverage over the duration of the test.

To accomplish this, Subgraph employs automated, manual and custom testing methods. We conduct our automated tests using the industry standard security tools. This may include using multiple tools to test for the same types of issues. We perform manual tests in cases where the automated tools are not adequate or reveal behavior that must be tested manually. Where required, we also develop custom tools to perform tests or reproduce test findings.

The goals of our testing methodology are to:

- Understand the expected behavior and business logic of the assets being tested
- Map out the attack surface
- Understand how authentication, authorization, and other security controls are implemented
- Test for flaws in the security controls based on our understanding
- Test every point of input against a large number of variables and observe the resulting behavior
- Reproduce and re-test findings
- Gather enough supporting information about findings to enable us to classify, report, and suggest remediations

Description of testing activities

Depending on the type and scope of the engagement, our methodology may include any of the following testing activities:

1. **Information Gathering:** Information will be gathered from publicly availble sources to help increase the success of attacks or discover new vulnerabilities
2. **Network discovery:** The networks in scope will be scanned for active, reachable hosts that could be vulnerable to compromise
3. **Host Vulnerability Assessment:** Hosts applications and services will be assessed for known or possible vulnerabilities
4. **Application Exploration:** The application will be explored using manual and automated methods to better understand the attack surface and expected behavior
5. **Session Management:** Session management in web applications will be tested for security flaws that may allow unauthorized access
6. **Authentication System Review:** The authentication system will be reviewed to determine if it can be bypassed

7. **Privilege Escalation:** Privilege escalation checks will be performed to determine if it is possible for an authenticated user to gain access to the privileges assigned to another role or administrator
8. **Input Validation:** Input validation tests will be performed on all endpoints and fields within scope, including tests for injection vulnerabilities (SQL injection, cross-site scripting, command injection, etc.)
9. **Business Logic Review:** Business logic will be reviewed, including attempts to subvert the intended design to cause unexpected behavior or bypass security controls

---

Reporting

Findings reports are peer-reviewed within Subgraph to produce the highest quality findings. The report includes an itemized list of findings, classified by their severity and remediation status.

Severity ratings

Severity ratings are a metric to help organizations prioritize security findings. The severity ratings we provide are simple by design so that at a high-level they can be understood by different audiences. In lieu of a complex rating system, we quantify the various factors and considerations in the body of the security findings. For example, if there are mitigating factors that would reduce the severity of a vulnerability, the finding will include a description of those mitigations and our reasoning for adjusting the rating.

At an organization's request, we will also provide third-party ratings and classifications. For example, we can analyze the findings to produce *Common Vulnerability Scoring System* (CVSS)[1] scores or *OWASP Top 10*[2] classifications.

The following is a list of the severity ratings we use with some example impacts:

Critical
Exploitation could compromise hosts or highly sensitive information

High
Exploitation could compromise the application or moderately sensitive information

Medium
Exploitation compromises multiple security properties (confidentiality, integrity, or availability)

Low
Exploitation compromises a single security property (confidentiality, integrity, or availability)

[1] https://www.first.org/cvss/
[2] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Informational

Finding does not pose a security risk or represents suspicious behavior that merits further investigation

The severity of a finding is often a product of the impact to general security properties of an application, host, network, or other information system.

These are:

Compromise of confidientiality	Exploitation results in authorized access to data
Compromise of integrity	Exploitation results in the unauthorized modification of data or state
Compromise of availability	Exploitation results in a degradation of performance or an inability to access resources

The actual severity of a finding may be higher or lower depending on a number of other factors that may mitigate or exacerbate it. These include the context of the finding in relation to the organization as well as the likelihood of exploitation. These are described in further detail below.

Contextual factors

Confidentiality, integrity, and availability are one dimension of the potential risk of a security finding. In some cases, we must also consider contextual factors that are unique to the organization and the assets tested.

The following is a list of those factors:

Financial compromise	Exploitation may result in financial losses
Reputation compromise	Exploitation may result in damage to the reputation of the organization
Regulatory liability	Exploitation may expose the organization to regulatory liability (e.g. place them in a state of non-compliance)
Organizational impact	Exploitation may disrupt the operations of the organization

Likelihood

Likelihood measures how probable it is that an attacker exploit a finding.

This is determined by numerous factors, the most influential of which are listed below:

Authentication required	Whether or not the attack must be authenticated
Privilege level	Whether or not an authenticated attacker requires special privileges
Public exploit	Whether or not exploit code is publicly available
Public knowledge	Whether or not the finding is publicly known
Exploit complexity	How complex it is for a skilled attacker to exploit the finding
Local vs. remote	Whether or not the finding is exposed to the network
Accessibility	Whether or not the affected asset is exposed on the public Internet
Discoverability	How easy it is for the finding to be discovered by an attacker
Dependencies	Whether or not exploitation is dependant on other findings such as information leaks

Remediation status

As part of our reporting, remediation recommendations are provided to the client. To help track the issues, we also provide a remediation status rating in the findings report.

In some cases, the organization may be confident to remediate the issue and test it internally. In other cases, Subgraph works with the organization to re-test the findings, resulting in a subsequent report reflecting remediation status updates.

If requested to re-test findings, we determine the remediation status based on our ability to reproduce the finding. This is based on our understanding of the finding and our awareness of potential variants at that time. To reproduce the results, the re-test environment should be as close to the original test environment as possible.

Security findings are often due to unexpected or unanticipated behavior that is not always understood by the testers or the developers. Therefore, it is possible that a finding or variations of the finding may still be present even if it is not reproducible during a re-test. While we will do our best to work with the organization to avoid this, it is still possible.

The findings report includes the following remediation status information:

Resolved
Finding is believed to be remediated, we can no longer reproduce it

In progress
Finding is in the process of being remediated

Unresolved
Finding is not remediated – may indicate the initial report, a finding under investigation, or one that the organization has chosen not to address

Not applicable
There is nothing to resolve, this may be the case with informational findings