$36k Google App Engine RCE

TL;DR

In early 2018 I got access to a non-production Google App Engine deployment environment, where I could use internal APIs and it was considered as Remote Code Execution due to the way Google works. Thanks to this I got a reward of $36,337 as part of Google Vulnerability Rewards Program.

Note

You can try an example of a few concepts I mention in this Google App Engine application. You can find the source code of that application, the source code of the gRPC C++ client, and every Protocol Buffer definition I got in this GitHub repository.

Some time ago, I noticed every Google App Engine (GAE) application replied to every HTTP request with a "`X-Cloud-Trace-Context`" header, so I assumed any website returning that header is probably running on GAE. Thanks to that, I learned "appengine.google.com" itself runs on GAE, but it can perform some actions that cannot be done anywhere else and common user applications cannot perform, so I tried to discover how was it able to do those actions.
Obviously, it has to make use of some API, interface or something only available to applications ran by Google itself, but maybe there was a way to access them, and I looked for that.

First, I began learning how GAE apps perform internal actions (Such as writing logs or getting an OAuth token), and I discovered that, in the Java 8 environment, it did so by sending Protocol Buffer (PB) messages (In binary wire format) to an internal HTTP endpoint located in `http://169.254.169.253:10001/rpc_http`.
The HTTP request would look like this:

```
POST /rpc_http HTTP/1.1
Host: 169.254.169.253:10001
X-Google-RPC-Service-Endpoint: app-engine-apis
```

About me
I'm an 18-year-old Uruguayan student interested in computer security.
Email: eze2307@gmail.com

```
X-Google-RPC-Service-Method: /VMRemoteAPI.CallRemoteAPI
Content-Type: application/octet-stream
Content-Length: <LENGTH>

<PROTO_MESSAGE>
```

And the PB message would be an "apphosting.ext.remote_api.Request" message with:

`service_name` = Name of the API to call

`method` = Name of the API's method to invoke

`request` = Bytes of the inner PB request (Encoded in binary wire format)

`request_id` = Security ticket (Given to the app with every GAE request), this is required even though it is marked as optional

The response from the HTTP request would be the corresponding PB message that represents the reply from the API, or an error message.

The security ticket can be obtained (In the Java 8 runtime) with these lines of code:

```
import com.google.apphosting.api.ApiProxy;
import java.lang.reflect.Method;

Method getSecurityTicket =
ApiProxy.getCurrentEnvironment().getClass().getDeclaredMethod("getSecurityTicket");
getSecurityTicket.setAccessible(true);
String security_ticket = (String)
getSecurityTicket.invoke(ApiProxy.getCurrentEnvironment());
```

An example of this process: If I want to get a Google OAuth token with the "https://www.googleapis.com/auth/xapi.zoo" scope (A test scope without real use), I would follow these steps:

1. Generate a "apphosting.GetAccessTokenRequest" message with:
   `scope` = `["https://www.googleapis.com/auth/xapi.zoo"]`
2. Generate a "apphosting.ext.remote_api.Request" message with:
   `service_name` = `"app_identity_service"` (The API that provide access to the GAE Service Account)
   `method` = `"GetAccessTokenRequest"`

    `request` = The bytes of the PB message generated in the previous step, encoded in binary wire format
    `request_id` = Security ticket
3. Send the HTTP request
4. Decode the response, which should be a "apphosting.GetAccessTokenResponse" message

Since this endpoint has access to some internal stuff, I was sure this must be related to whatever "appengine.google.com" uses for performing internal actions, but I could not find anything in the HTTP endpoint.
At first I guessed it might be using some other endpoint located in the same server (`169.254.169.253`), so I uploaded a statically linked version of Nmap to GAE and ran it against the server (For running binaries in GAE I upload them with the app, then during runtime I copy them to `/tmp` and give them execution permission, since the rest of the file-system is read-only). Here is a live example.
I found that the port 4 was open, so I sent stuff to it. It replied with a weird mess of data, but it had some legible strings and after looking them up on-line I found it was a gRPC service.

I tried to build a Java gRPC client that runs on GAE, but I was having troubles since the built-in gRPC library seemed to be incomplete and whenever I uploaded a complete one it still tried to use the built-in library.
So I built a C++ client instead and ran it on GAE.

After some trial and error I discovered the gRPC service was just like the HTTP endpoint, running a "apphosting.APIHost" API. There was a difference though, it had the option for JSON encoding of the PB messages instead of just binary, so it made testing much easier.
Here is a live example of this client.

Since I did not find anything else in the server, I assumed the actions "appengine.google.com" does internally either contact a different server, or use the RPC services (HTTP/gRPC) for invoking some hidden APIs/methods.
I tried finding any other server with Nmap, but I only found the Metadata server, which was not useful, so I went with the idea that it must use hidden APIs, but, how to find them?

First, I collected every Protocol Buffer definition I could find (Extracting them from .CLASS files found in .JAR files, and from binaries found in the runtime) and searched in them anything that could point to some hidden API (If you are curious, all the PB definition files I extracted can be found here).

I found promising the "apphosting/base/appmaster.proto" file, it had several PB messages that seemed like internal methods for modifying internal settings of App Engine, and an API called "AppMaster" with some methods defined in it, but after several trials I could not find the way to perform any call to those methods.

Since I did not find any of the hidden APIs/methods in the PB definitions, I had to look somewhere else.
I tried looking in the binaries, they were huge and full of stuff that was either useless or I did not understand (Also, I was exploring them using a combination of `strings` + `grep`, I do not know much about reverse engineering), but after noticing the main binary, "`java_runtime_launcher_ex`", had a lot of command line parameters, I had the idea of looking at what parameters did it receive when running in the GAE environment.

Getting the parameters was quite difficult at first because I tried to connect every Java variable I could find to its corresponding parameter, it was impossible.
Then I tried something smarter: Creating a Java library in C++ with a method that reads the arguments passed to the launcher and returned them.
Doing so was easy to do, thanks to this Stack Overflow post, retrieving the information with these lines of code:

```
int argc = -1;
char **argv = NULL;

static void getArgs(int _argc, char **_argv, char **_env) {
  argc = _argc;
  argv = _argv;
}

__attribute__((section(".init_array"))) static void *ctr = (void*) getArgs;
```

And then a simple method that converted the arguments to a Java array. Here is a live example.

After running the code, I got lots of arguments, among them was this one (I divided it into multiple lines for readability):

```
--api_call_deadline_map=
  app_config_service:60.0,
  blobstore:15.0,
  datastore_v3:60.0,
  datastore_v4:60.0,
  file:30.0,
  images:30.0,
  logservice:60.0,
  modules:60.0,
  rdbms:60.0,
  remote_socket:60.0,
  search:10.0,
  stubby:10.0
```

I quickly noticed the APIs I had already used, like "`logservice`" (For writing logs), so I deduced that these were APIs available through the internal HTTP endpoint.
I also noticed "`stubby`", which I had already seen mentioned before in error messages from some Google products (While bug-hunting) and I had read about it in the SRE, so I knew it was a RPC infrastructure, and it might be a way for "appengine.google.com" to perform internal actions.

Great, now I know the name of an internal API, but, what methods does it have?
I tried several method names with my C++ gRPC client, but all of them returned an error saying they do not exist, so instead I looked up in Google.
I somehow found this 2010 post with an error message reading:

```
  The API call stubby.Send() took too long to respond and was cancelled.
```

So, I tried the "`Send`" method. It did not exist.

I was sure it must exist, so the error message was probably just hiding the fact that it does exists but I do not have access to it.
I tried to verify it by finding any difference between a real "not-exist" error (Example) and a fake one (Example), and I found it: If in my gRPC client I made a request without setting the "apphosting.APIRequest.pb" field (Which is marked optional but I always set it to at least an empty string or "`{}`" in JSON), it would return a "not-exist" error for a non-existent method (Example), and a "incomplete request" error to a real method (Example) (Even if it supposedly did not exist). Therefore, "`stubby.Send`" does in fact exist.

Now, how to access it?
I could not come up with a way for accessing it in the production GAE deployment environment, but then I remembered I had gotten access to the staging (staging-appengine.sandbox.googleapis.com) and the test (test-appengine.sandbox.googleapis.com) GAE deployment environments thanks to this bug (Normally, common Google users should not have access to non-production deployment environments).
Thanks to some little research in those deployment environments, I knew how to perform a call to an app that runs in them:

1. Upload a version with manual scaling (It did not work otherwise, for some weird reason, returning `403 Forbidden`)
2. Perform a request to "www.appspot.com" but change the `Host` header to "`<PROJECT-NAME>.prom-<qa/nightly>.sandbox.google.com`"
   If your app would normally run on "save-the-expanse.appspot.com", you should replace "`<PROJECT-NAME>`" with "`save-the-expanse`", and if you uploaded your app to the staging GAE environment, you should replace "`<qa/nightly>`" with just "`qa`", if you uploaded it to the test GAE environment instead, you should replace it with "`nightly`".
   For example: I tested on "`the-expanse.prom-nightly.sandbox.google.com`" (Without the "save", since The Expanse had not been canceled back then).

The bug

Once I uploaded my application with the gRPC client, I quickly discovered that, in the non-production (staging/test) GAE environments, I had access to "`stubby.Send`"!
After some quick testing (Mostly reading error messages and guessing how to fix them), I found how to perform a simple Stubby call:

1. Call "`stubby.GetStubId`" with the following JSON PB message:
   ```
   {
     "host": "<HOST>"
   }
   ```
   With `<HOST>` set to where the method you want to call is hosted (For instance, "`google.com:80`", "`pantheon.corp.google.com:80`", "`blade:monarch-cloud_prod-streamz`").
   "`blade:<SERVICE>`" seems to be like an internal DNS system Google uses, for instance, "`blade:cloudresourcemanager-project`" internally is "cloudresourcemanager.googleapis.com" externally (Some, like "`blade:monarch-cloud_prod-streamz`", do not have an external counterpart).
2. The previous request will return a JSON PB message with "stub_id" as its only field, store its value
3. Call "`stubby.Send`" with the following JSON PB message:
   ```
   {
     "stubby_method": "/<SERVICE>.<METHOD>",
     "stubby_request": "<PB>",
   ```

```
  "stub_id": "<STUB_ID>"
}
```

For finding what values can "stubby_method" be, you can set it to "/ServerStatus.GetServices" with an empty "`stubby_request`" and it will return a nice "rpc.ServiceList" listing all the services (And their methods) the target supports.
<PB> are the PB message bytes (In binary wire format).

4. If successful, the call will return a JSON PB message with "`stubby_response`" as its only field, it'll have the response PB message bytes (In binary wire format).

After discovering this, I did some testing, but I was not able to find any Stubby call that I considered dangerous. Nevertheless, I reported this to Google and it got a P1 priority.

After the initial report, I looked over everything I've done again, trying to find some variation that could be successfully used for an attack, and I noticed that, besides "`stubby`", there was "`app_config_service`" in the arguments I got from the Java launcher binary, it was another hidden API.
Looking in the PB definitions I had gotten before, I couldn't find its methods directly, nor on Google Search, but I later found them mentioned in "apphosting/base/quotas.proto".
For example, it says "APP_CONFIG_SERVICE_GET_APP_CONFIG", and a little testing revealed "`app_config_service.GetAppConfig`" is a real hidden method.

The "`app_config_service`" has several interesting methods, but the most interesting methods for me were the "`app_config_service.ConfigApp`" and the "`app_config_service.SetAdminConfig`" methods, because they allowed me to set internal settings such as the allowed email senders, the app's Service Account ID, ignore quota restrictions, and set my app as a "SuperApp" (I don't know what that means, but sounds super) and give it "FILE_GOOGLE3_ACCESS" (I think Google3 is a part of Piper, with files related to Google's APIs and services).
The "`app_config_service.SetAdminConfig`" method has "apphosting.SetAdminConfigRequest" as its request message, and "`app_config_service.ConfigApp`" has "apphosting.GlobalConfig" as its request message.

I also found some other APIs/methods thanks to "apphosting/base/quotas.proto", like "basement.GaiaLookupByUserEmail".

After discovering this, I reported the new findings to Google and they bumped the priority of the internal ticket and said:

```
 Please stop exploring this further, as it seems that you could easily break something
using these internal APIs.
```

Then the issue was CC'd to several employees:

mo...@google.com <mo...@google.com> #16 Mar 6, 2018 10:3

```
Thanks for the update. I bumped up the severity of the bug filed to the product
team.

Please stop exploring this further, as it seems that you could easily break
something using these internal APIs. When issuing the reward, we'll take into
account what you could've achieved with this access if you wanted to.
```
1(

-Hotlist: 702027 1(

mo...@google.com <mo...@google.com> Mar 6, 2018 02

+CC: sp...@google.com 02
+CC: fr...@google.com 02
+CC: gd...@google.com 02

mo...@google.com <mo...@google.com> Mar 6, 2018 04

+CC: an...@google.com 04
+CC: cp...@google.com 04
+CC: no...@google.com 04

mo...@google.com <mo...@google.com> Mar 7, 2018 04

+CC: wh...@google.com 04
+CC: bk...@google.com 04
+CC: ap...@google.com 04

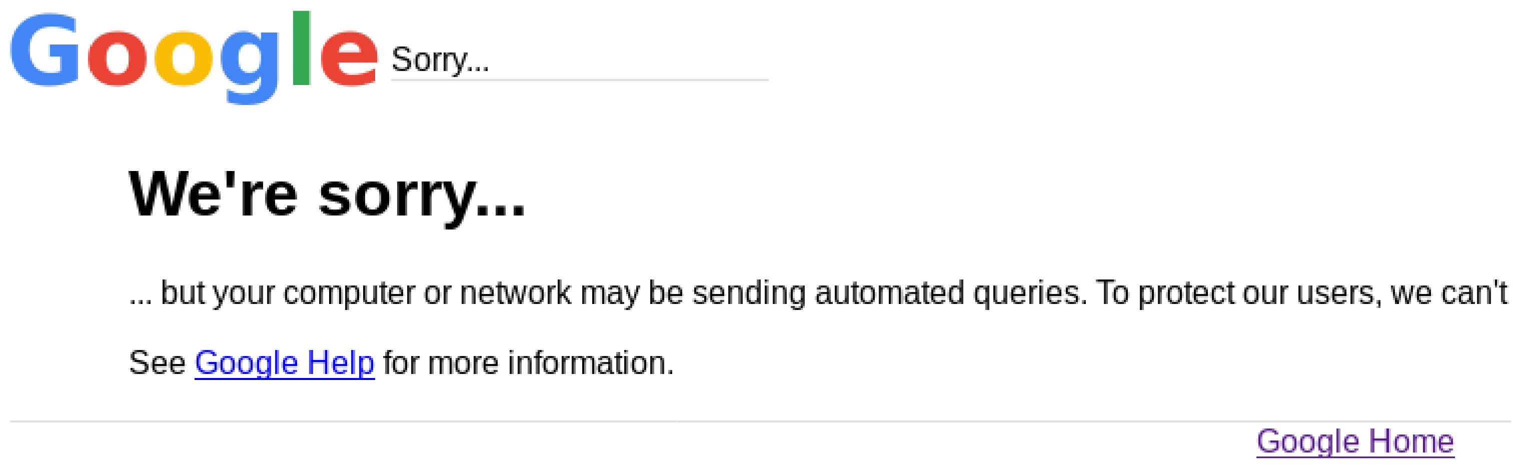
mo...@google.com <mo...@google.com> Mar 8, 2018 08

+CC: ha...@google.com 08

au...@google.com <au...@google.com> Mar 8, 2018 06

+CC: sa...@google.com 06

A few days later, the access to non-production GAE APIs and environments was blocked with this error page (With status "`429 Too Many Requests`").
You can still see this message in "staging-appengine.sandbox.googleapis.com" and "test-appengine.sandbox.googleapis.com".



Google Sorry...

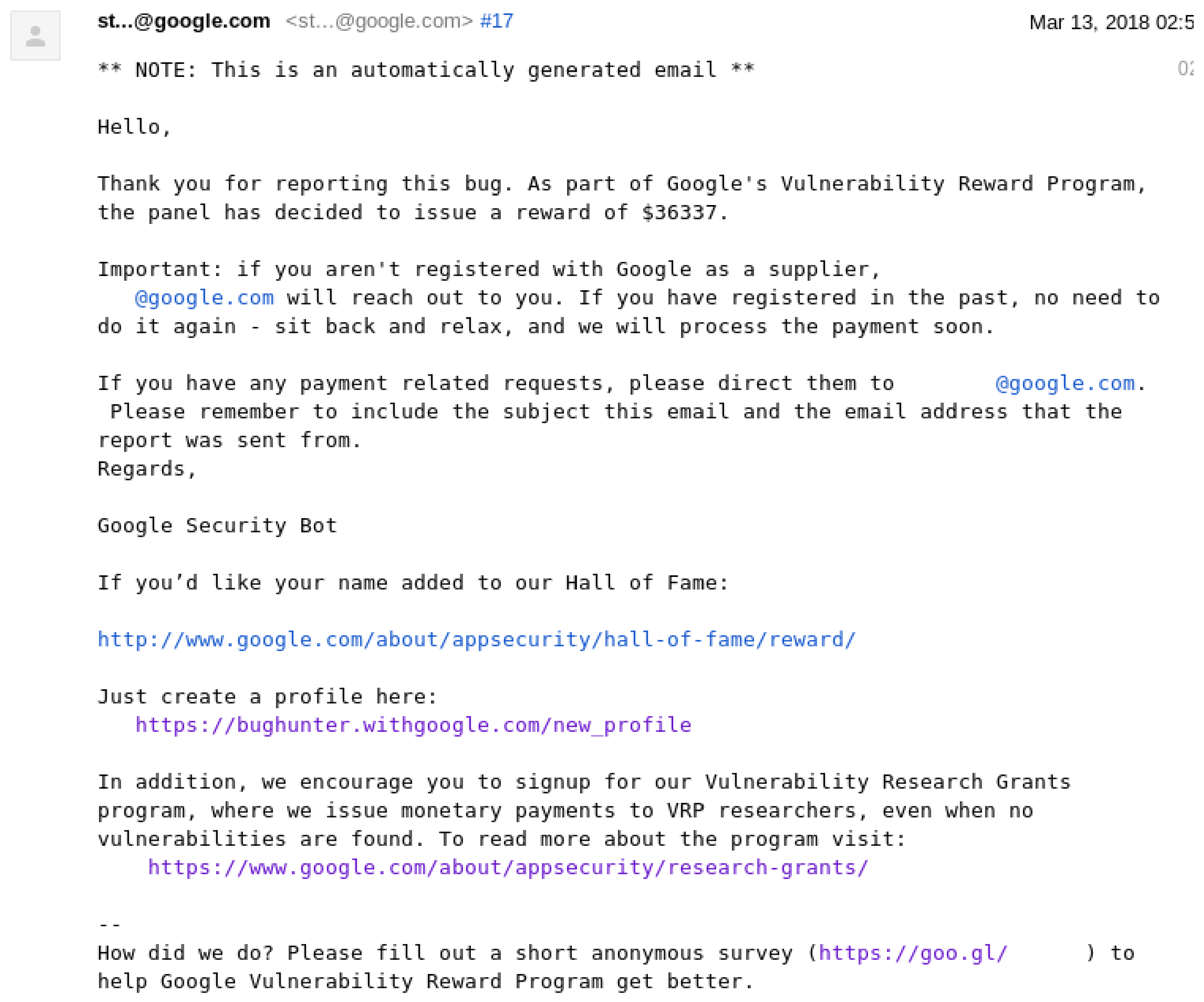
We're sorry...

... but your computer or network may be sending automated queries. To protect our users, we can't

See Google Help for more information.

Google Home

And later I got the following message:


st...@google.com <st...@google.com> #17 Mar 13, 2018 02:5

```
** NOTE: This is an automatically generated email **                02

Hello,

Thank you for reporting this bug. As part of Google's Vulnerability Reward Program,
the panel has decided to issue a reward of $36337.

Important: if you aren't registered with Google as a supplier,
   @google.com will reach out to you. If you have registered in the past, no need to
do it again - sit back and relax, and we will process the payment soon.

If you have any payment related requests, please direct them to        @google.com.
 Please remember to include the subject this email and the email address that the
report was sent from.
Regards,

Google Security Bot

If you’d like your name added to our Hall of Fame:

http://www.google.com/about/appsecurity/hall-of-fame/reward/

Just create a profile here:
   https://bughunter.withgoogle.com/new_profile

In addition, we encourage you to signup for our Vulnerability Research Grants
program, where we issue monetary payments to VRP researchers, even when no
vulnerabilities are found. To read more about the program visit:
    https://www.google.com/about/appsecurity/research-grants/

--
How did we do? Please fill out a short anonymous survey (https://goo.gl/      ) to
help Google Vulnerability Reward Program get better.
```

I was rewarded **36,337** dollars!

I was not aware until then that this was regarded as Remote Code Execution (The highest tier for bugs), it was a

very pleasant surprise.
I asked to one of the Googlers in the reward panel about it, and he told me it is RCE for the way Google works (And suggested reading the SRE) and also that the extra $5k (Since they pay $31,337 for RCE bugs) was for a lesser bug.

Timeline

- *February 2018*: Issue found
- *February 25th, 2018*: Initial report (Only the "`stubby`" API)
- *March 4th and 5th, 2018*: The "`app_config_service`" API discovered and reported
- *March between 6th and 13th, 2018*: The access to non-prod GAE environments was blocked with a 429 error page
- *March 13th, 2018*: Reward of $36,337 issued
- *May 16th, 2018*: Issue confirmed as fixed

About me

I am 18-year-old student at the University of the Republic (A public university in Uruguay) interested in computer security, you can contact me at eze2307@gmail.com.