Network Security

Security Analysis of Github

Sean Smith, Kyle Holzinger, Amalia Safer

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 invalided 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 getting a correct cookie is so low that it's not a reasonable attack.

$$Pr[] \approx$$

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

Figure 1: Browser Fingerprint

CSP

Content security policy or CSP is a way of whitelisting domains that github.com is allowing to load into into the page. This whitelist contains the.

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%
divshot.github.com	59,400	0.39%
guides.github.com	56,700	0.37%
fortawesome.github.com	44,500	0.29%

Figure 2: Subdomains

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" }
```

Git Commit 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.

Advisor: Sharon Goldberg

Network Security

Security Analysis of Github

Sean Smith, Kyle Holzinger, Amalia Safer

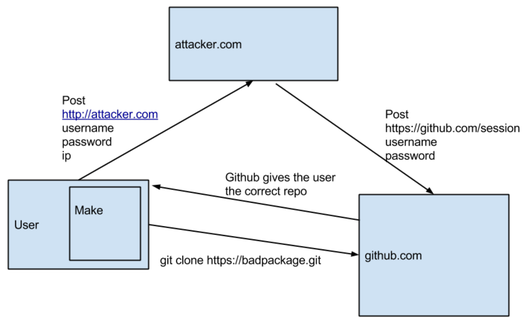

Figure 3: Diagram of our attack

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 on any operating system.
   `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.

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

XSS

Github has two major protections against cross-site-scripting (XSS) attacks.

Advisor: Sharon Goldberg