Asserting Membership in OpenStreetMap

Roland M. Olbricht

September 19, 2018

User authentication in OpenStreetMap does not only serve to login into the website osm.org. The accounts are also used in a widespread field of related services. This starts with editors [4] that submit the map data on behalf of the human mappers. Other services (e.g. [1], [3]) mostly rely on the confirmation of which user is logged and do not interact otherwise with the API. Finally, with the advent of the GDPR, there is now a class of services that need to know that a confirmed OSM user is requesting but that even do not want to know which one - because this just would generate unwanted extra personal data.

OpenStreetMap has started with the well-known concept of user name and password. However, as users shall be discouraged to enter OSM credentials on third party sites, an authentication concept called OAuth [8] has been adopted. This redirects from the third party website to the original osm.org login page and then redirects back to the login requesting third party site.

As the notions and definitions of OAuth often cause confusion, we first define a set of roles that reflect more closely the needs of the OpenStreetMap ecosystem.

1 Glossary

Credential Checker: The remote system that can tell whether a pair of user name and password constitutes a well-known user. May return other cryptographic information to start a session.

End User: The person or system that is triggering the data exchange. Most often this is a human that wants to get some information or do something. But this could as well be a system that updates its downstream database once a day or so.

Human User: An end user user that is definitely a human user and not a server or script.

Remote Storage: The remote system that holds data which is intended only for known users. In case of OpenStreetMap this is the main database.

User Agent: The software that makes the connection on behalf of the end user. This may be a browser, but it can also be a smartphone app, a script, or the JavaScript bundle of a website. In case of a non-human user there is usually no distinction between the user agent and the end user.

Web Service: A service that relies on the Credential checker to verify somebody is a known user. It may communicate with the remote storage. But in the OpenStreetMap ecosystem, many services have prefetched their data and need no further communication.

2 Editing tools

As the OpenStreetMap ecosystem is diverse, I would like to discuss a couple of use cases as they exist today. I aspire to be comprehensive but I cannot warrant that.

We start with the case of the website osm.org itself. There is one user agent involved, the browser. The credential checker, the web service, and the remote storage are all in one site. Thus, with or without OAuth, the user fills in the credentials into a form supplied by the web service. The security relies on the security of osm.org plus the security of the used connection.

In addition, the security of the connection between the user agent and the credential checker is crucial. On the one hand, up-to-date configurations of HTTPS are considered secure enough at the time of writing this document. On the other hand, this connection is required sometimes or always in any discussed scenario. Thus, this connection mode is assumed to be secure enough for the purpose of this document.

There are at least three other common scenarios:

The user can edit data in the main database by using an online editor like *Level0* [7]. Technically, the iD editor would be similar but it is now integrated in osm.org. The user logs in to be able to supply added or edited data that is then associated to his user account. In this case, the end user is a human user and the credential checker is osm.org. For the web service and user agent, it is more difficult. Technically, the user agent is the JavaScript code of the website in the end user's browser and no web service is involved.

The user interface is designed to show otherwise: The user agent is the browser and the web service is the loaded website. The rationale behind this is to inhibit *phishing* [16] and *pharming* [15] by educating users to supply their credentials nowhere else than on the credential checker's website.

Another situation are standalone editors, both on the desktop like JOSM [5] and on the smartphone like Vespucci [18]. The end user still is a human user and the credential checker still is osm.org. But there is no longer any independent browser that can act as a trusted user agent. While there is a browser on the system, a reasonably secure browser will not leak information to an unrelated application. Hence, the standalone editor cannot work with the intended user interface of OAuth. What happens instead behind the scenes in JOSM is screen scraping of the osm.org website. While this works this is technically as secure as supplying user name and password by HTTPS POST to osm.org but worse because it gives the impression of security where no security is.

3 Data Publishing

The third scenario is publishing of the OpenStreetMap data, the mission of OpenStreetMap [11]. OpenStreetMap uses a distributed approach, i.e. the data is eagerly copied to many mirror services. Examples for such mirror services are *Geofabrik Downloads* [2] and *Overpass API* [13] but also the *Planet.osm* service [17] on a subdomain of osm.org.

The web service is in this case identical to the remote storage. It is different from the credential checker.

The end user is often a human user and using the web browser as user agent. But there are other legit constellations:

- JOSM [5] can fetch data from Overpass API. Here we have a human end user and a distinct piece of software as user agent.

- *Overpass Turbo* [14] is a website that fetches and displays data from Overpass API. In this case, the JavaScript code of Overpass Turbo acts as user agent and the end user is a human user.

- Geofabrik and Overpass API themselves fetch the necessary data to keep up to date from Planet.osm. In this case, the end user and user agent are the same and it is a server script.

- *OSMCha* [12] is a website that displays results of its own database derived from OpenStreetMap data. In this case the two user agent models mentioned above do apply again. But the remote storage is still identical to the web service and separated from the credential checker.

Many other services fall into one of these categories. An inventory of the OpenStreetMap ecosystem is beyond the scope of this paper.

Note that also the amount of information about the logged-in user varies substantially. JOSM does not need at all user information to download data. User data is only needed to upload edited data. Overpass API and Geofabrik do not need to know anything about the user. By contrast, OSMCha has a legit reason to know the name and uid of the logged-in user.

4 Features and Limitations of OAuth

OpenStreetMap makes use of OAuth version 1.0 [8]. There exists also a backwards incompatible successor, OAuth version 2.0 [9]. Given that OAuth 2.0 is way too complex [10], there are no plans to switch to the newer version. The downside of this is that the underlying assumptions and the chosen cryptography is stuck with the state-of-the-art of 2010. In all of the following, OAuth always refers to OAuth 1.0.

OAuth is based on a three-party model:

- The remote storage is forced to be identical to the credential checker. This party is called *service provider* or *server*.

- The end user is expected to be a human user and the user agent to be a browser. This party is called *user* or *resource owner*.

- The web service has no relevant data, and it is called *consumer* or *client*.

We will check in a moment whether our constellations match the requirements.

Given that the requirements are met, OAuth warrants the following:

- The web service and its connections only know as little data as possible to fulfill their tasks on behalf of the end user.

- The credentials only travel between the user agent and the credential checker. In particular, the security model of a standard conformant browser ensures that no JavaScript from any other website than the credential checker sees the credentials.

In case of the online editor, the benefit is that the editor never sees the password. Having only a HTTP connection instead of HTTPS for the online editor does not jeopardize the credentials. HTTPS needs a CA signed certificate, and such certificates have been expensive in 2010. The advantage of not needing HTTPS is now obsolete because *Let's Encrypt* [6] offers certificates for free and tailored to the respective requirements of each website.

In case of the standalone editor, the credentials are inherently unsafe to any component of the editor. While the legit editor itself might do some kind of isolation, a tampered-with version may store or distribute credentials within the full software, and no other mechanism has any channel to inform the user about that. There is no third party backend a standalone editor needs to talk to, hence there is no benefit to protect information from that non-existing party.

Similarly, whenever the end user is not a human user the user name and password need anyway to be stored in or regularly resent to the user agent. This again defeats the mentioned first benefit.

In the third case we care for the identity of the remote storage. It is distinct from the credential checker and identical to the web service. This implies first that no information needs to flow from the credential checker to the web service beside the user name. In most of the mentioned scenarios, this means that the web service needs to implement the full OAuth framework including security as of 2010 just to get an information that it should not get at all with regard to the GDPR. Second, the web service has anyway all information because it is the remote storage. Thus, the second benefit again does not apply.

The only execption to this is a service like OSMCha that needs to know the user name. Here, the connection between credential checker and web service has at least some advantage.

5 Projected Alternative

The basic considerations behind the alternative design are as follows: Security is always only as strong as the weakest point in the system. Thus, added authentication workflows must not fall behind the security level of OAuth. HTTPS can now assumed to be a commodity and required everywhere. That obsoletes the need to obfuscate information on the connection. In particular, the technical burden on the involved systems should be as low as possible. We want to encourage and not to discourage third party tools. Finally, the design must allow for the different role combinations that actually exist in the OpenStreetMap ecosystem.

The user agent on behalf of the end user can have two different requirements: It may want to fetch information from the remote storage. And it may want to talk to osm.org to submit edited data.

The latter requires that osm.org is used as endpoint for the payload. Thus, osm.org is forcibly the identity provider.

The former does not require any communication to any other party than the remote storage except to check that the end user is a legit user. Thus we have the latitude to set up a proxy for the moment being that talks to the end user and remote storage on the front side and to osm.org on the back side. That proxy can and shall be migrated to osm.org once the system is working reliable. We refer to this for the rest of the section as the credential checker.

The credential checker requires any web service to register. It then exposes to each registered application a feed of API keys that are only visible to that web service. Those API keys can be built from nonces, i.e. purely random data.

To the end user, the credential checker offers a web interface. That web interface allows the user to enable an API key per application the user wants to use.

The end user then supplies the API key to its user agent. The user agent sends the API key on each request to the web service. The web service just has to match the sent API key to the list of its known keys and is not confined to a particular cryptographic standard.

The cryptographic protection stems from that the connections are all secured by compulsory HTTPS, and that the API keys are unrelated to anything else and random thus cheap to replace. As an improvement, the user name is not exposed to the web service.

We now have all freedom to adapt that system to the needs of the respective web service. For example, extra parameters can be added to the API key to allow the credential checker to grant or refuse user rights on the web service. As the application is set up to consume the feed from the credential checker there is a first-class channel to revoke keys.

Or, key renewal could be automated by allowing the user agent to renew the key with user name plus old API key. The web service does not need to know and cannot figure out whether the old and new key belong to the same user. In addition, the transmission of the password is clamped down to one single occasion as opposed to OAuth that needs to transmit the password on every session initiation.

6 Conclusion

OAuth had been in 2010 the best-in-class solution to run an online editor on osm.org managed credentials. As of 2018 and with the GDPR in force, the OpenStreetMap ecosystem has different requirements.

A better alternative is possible but not yet done. A prototype can use OAuth with screen scraping for an intermediate period to show that the setup works in practice. A migration to osm.org as additional authentication service is highly recommended.

Whether OAuth should be retracted due to its age as opposed to its fitness for online editors is out of scope for this report.

References

[1] https://forum.openstreetmap.org/

[2] https://download.geofabrik.de/

[3] https://help.openstreetmap.org/

[4] https://github.com/openstreetmap/iD

[5] https://josm.openstreetmap.de/

[6] https://letsencrypt.org/

[7] http://level0.osmz.ru/

[8] https://tools.ietf.org/html/rfc5849

[9] https://tools.ietf.org/html/rfc6749

[10] https://en.wikipedia.org/wiki/OAuth#OAuth_2.0

[11] https://wiki.osmfoundation.org/wiki/Mission_Statement

[12] https://osmcha.mapbox.com/

[13] https://wiki.openstreetmap.org/wiki/Overpass_API

[14] https://Overpass-turbo.eu/

[15] https://en.wikipedia.org/wiki/Pharming

[16] https://en.wikipedia.org/wiki/Phishing

[17] https://planet.openstreetmap.org/

[18] https://vespucci.io/