Ship Public Suffix List (PSL) over Remote Settings

Mozilla

Project Name: Ship Public Suffix List (PSL) over Remote Settings

Project Mentor: Mathieu Leplatre **[leplatrem]**

Personal Details

Name: Arpit Bharti

Email: arpitbharti73@gmail.com

IRC nickname: arpit

Phone: +91-9810020853

Country of residence: India

Primary Language: English

Time Zone: UTC+05:30

Project Abstract

An update to the public suffix list (PSL) is shipped bundled with Firefox releases. The current system creates the dafsa binary file at build time. The list has to be manually updated with every new release and cannot be updated for people running old versions of Firefox. I will be working to create an update system using the remote-settings API in Firefox to deliver an updated list to the client periodically, the period is yet to be determined. The list will be published on the mozilla remote-settings servers and be shipped as a compressed DAFSA binary file. This binary format allows for shipping the list quickly over the network as it is orders of magnitude smaller in size.

Project Tasks

The project can be divided into two halves based on where the code will run. The preprocessing of the list and publishing will happen on the server side. After the list has been published and approved, only then will it be downloaded by clients using firefox. The task to be accomplished may be summarised as the following.

On the server side :

- Create a script to download the PSL
- Create a script to remove extra symbols and whitespace and comments
- Port the python script make_dafsa.py that creates the DAFSA binary file from firefox to run on server
- Request access to Stage/Prod servers
- Create a script to upload the DAFSA to a remote-settings server and request review and signoff

The above can be achieved using kinto using cURL if I use bash or kinto-http.py if using python for everything.

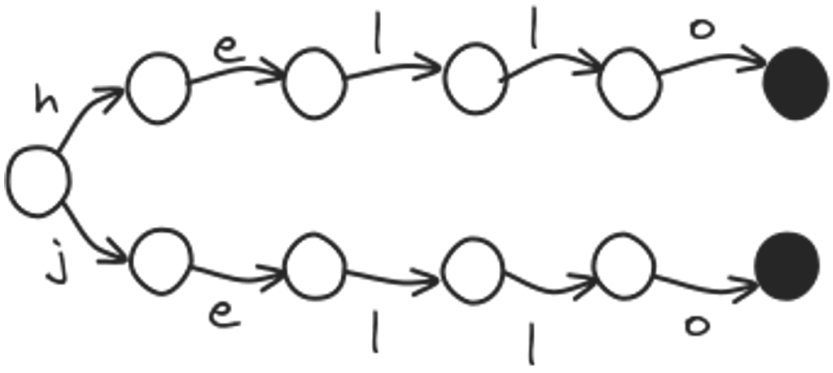
On the client side :

- Download the compressed dafsa file using the remote-setting api. Handle errors that may arise.
    - Checking for available space on client's disk
    - Resuming the download in case of failure
    - Verify integrity (md5sum) regularly
    - Redownload the file in case it was corrupt
- Store the new download somewhere in the profile folder
- Hook up the previous system to read from the new location
- Remove the old system that creates the dafsa binary at build time

Technical Details

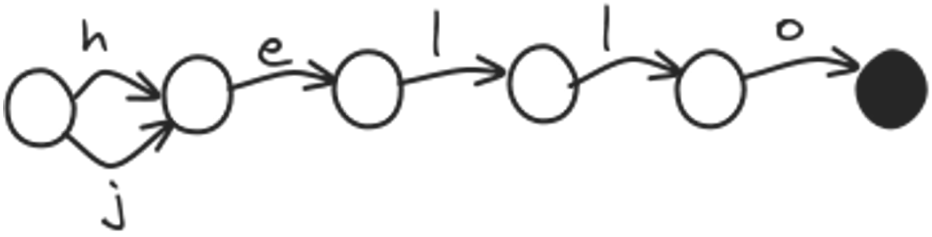
What is a Trie?

Trie, also called a prefix tree, is a kind of search tree, an ordered tree data structure can be used to store strings to create fast lookup structures, like a dictionary. In the below example we can see a structure to represent the words 'hello' and 'jello', each letter has a different node and there is a lot of repetition happening.



What is a DAFSA?

A deterministic acyclic finite state automaton (DAFSA) can be understood as a compressed trie. The goal is to compress the structure and optimise for space.



Each finite language is generated by a trie automaton, and each trie can be compressed into a deterministic acyclic finite state automaton.
By allowing the same vertices to be reached by multiple paths, a DAFSA may use significantly fewer vertices than a trie. The primary difference between DAFSA and trie is the elimination of suffix and infix redundancy in storing strings. The trie eliminates prefix redundancy since all common prefixes are shared between strings, such as between doctors and doctorate the doctor prefix is shared. In a DAFSA common suffixes are also shared, for words that have the same set of possible suffixes as each other. For dictionary sets of common English words, this translates into major memory usage reduction.

Because the terminal nodes of a DAFSA can be reached by multiple paths, a DAFSA cannot directly store auxiliary information relating to each path, e.g. a word's frequency in the English language. However, if for each node we store the number of unique paths through that point in the structure, we can use it to retrieve the index of a word, or a word given its index. The auxiliary information can then be stored in an array.

To summarise, a trie is a data structure that can be used to make dictionaries and validate whether a string belongs to a particular set or not. A dafsa is a data structure that achieves the same by compressing the repeated nodes into a single node and instead defining new paths that lead out of it, It is conscious about space and and is orders of magnitude smaller than a trie.

Sources:

- Tries -
    - https://en.wikipedia.org/wiki/Trie
    - https://www.geeksforgeeks.org/trie-insert-and-search/
    - https://xlinux.nist.gov/dads/HTML/directedAcyclicWordGraph.html
- Dafsa
    - https://en.wikipedia.org/wiki/Deterministic_acyclic_finite_state_automaton
    - http://stevehanov.ca/blog/?id=115
    - http://www.wutka.com/dawg.html
    - https://xlinux.nist.gov/dads/HTML/directedAcyclicWordGraph.html

What is the Public Suffix List?

According to publicsuffix.org, a "public suffix" is one in which Internet users can register names directly (or in the past). Examples of public suffixes are .com, .co.uk and pvt.k12.ma.us. The Public Suffix List is a list of all known public suffixes.

The Public Suffix List is a multi-vendor initiative to provide an accurate list of domain name suffixes, which is maintained by Mozilla volunteers. The usefulness of this is shown by the example of cookies. In the past, browsers used an algorithm that only refused to set far-reaching cookies for top-level domains without dots (for example, com or org). However, this did not work for top-level domains where only third level registrations are allowed (eg co.uk). In these cases, websites could set a cookie for .co.uk, which will be shared with every website registered under co.uk.

Since there exists no method of finding the highest level at which a domain may be registered for a particular top-level domain (the policies differ with each registry), the only method is to create a list. This is the aim of the Public Suffix List.

- Software using the Public Suffix List will be able to determine where cookies may and may not be set, protecting the user from being tracked across sites

- As well as this, the Public Suffix List can also be used to support features such as site grouping in browsers. By knowing where the user-controlled section of the domain name begins and ends, browsers can group cookies and history entries by site in a way that couldn't easily be done before

- PSL can be used to determine what is a valid domain name and what isn't

You can view the list at https://publicsuffix.org/list/public_suffix_list.dat

List Format

A public suffix is a set of DNS names or wildcards concatenated with dots. It represents the part of a domain name which is not under the control of the individual registrant.

Specification

- The list is a set of rules, with one rule per line.
- Each line is only read up to the first whitespace; entire lines can also be commented using `//`.
- Each line which is not entirely whitespace or begins with a comment contains a rule.
- Each rule lists a public suffix, with the subdomain portions separated by dots (.) as usual. There is no leading dot.
- The wildcard character * (asterisk) matches any valid sequence of characters in a hostname part. (Note: the list uses Unicode, not Punycode

forms, and is encoded using UTF-8.) Wildcards are not restricted to appear only in the leftmost position, but they must wildcard an entire label. (I.e. *.*.foo is a valid rule: *bar.foo is not.)

- Wildcards may only be used to wildcard an entire level. That is, they must be surrounded by dots (or implicit dots, at the beginning of a line).
- If a hostname matches more than one rule in the file, the longest matching rule (the one with the most levels) will be used.
- An exclamation mark (!) at the start of a rule marks an exception to a previous wildcard rule. An exception rule takes priority over any other matching rule.

Example

Here is an example (incomplete) list section. The rules are numbered, but the numbers would not appear in the real file:

```
            1. com
            2. *.jp
            // Hosts in .hokkaido.jp can't set cookies below level 4...
            3. *.hokkaido.jp
            4. *.tokyo.jp
            // ...except hosts in pref.hokkaido.jp, which can set cookies
at level 3.
            5. !pref.hokkaido.jp
            6. !metro.tokyo.jp
```

The example above would be interpreted as follows, in the case of cookie-setting, and using "foo" and "bar" as generic hostnames:

1. Cookies may be set for foo.com.
2. Cookies may be set for foo.bar.jp.
   Cookies may not be set for bar.jp.

3. Cookies may be set for foo.bar.hokkaido.jp.
   Cookies may not be set for bar.hokkaido.jp.
4. Cookies may be set for foo.bar.tokyo.jp.
   Cookies may not be set for bar.tokyo.jp.
5. Cookies may be set for pref.hokkaido.jp because the exception overrides the previous rule.
6. Cookies may be set for metro.tokyo.jp, because the exception overrides the previous rule.

How is Firefox using DAFSAs and the Public Suffix List?

Currently the public suffix list (PSL) is shipped as .dat file name effective_tld_names.dat this file is compiled into a DAFSA at build time. This is done by the python script called make_dafsa.py. According to the documentation:

```
This python program converts a list of strings to a byte array in C++.
This python program fetches strings and return values from a gperf file
and generates a C++ file with a byte array representing graph that can be
used as a memory efficient replacement for the perfect hash table.

The input strings are assumed to consist of printable 7-bit ASCII
characters
and the return values are assumed to be one digit integers.

In this program a DAFSA is a diamond shaped graph starting at a common
source node and ending at a common sink node. All internal nodes contain
a label and each word is represented by the labels in one path from
the source node to the sink node.
```

This script generates the a byte array that encodes a dictionary of strings and values. This file is parsed at runtime to check for which values belong to the PSL when user types a top level domain in the url bar.

Firefox is currently using the public list for:

- Restricting cookie setting
- Restricting the setting of the document.domain property
- Sorting in the download manager
- Sorting in the cookie manager
- Searching in history
- Domain highlighting in the URL bar

Short Description of Timeline

In the starting weeks of GSoC, I will be working on the **DAFSA** creation and publishing scripts. This will include writing the python script that compiles the PSL and creates an deliverable binary, also bash or python scripts for publishing the new binaries to the remote-settings server.
One or two week before the mid term evaluation I will start working on the code to download the new binaries from the remote-settings server and store them locally.

The motive of this project will also be to allow solving of bugs that currently depend on the existence of this functionality eg. The bugs :
Don't offer to open non-linked text that looks like urls such as file names also
Use PSL to do a search for foo.bar URL bar entries which aren't known domains, with the same infobar as for single-word searches

Proposed Detailed Timeline

Pre GSoC

- Learn about DAFSA data structure
- Learn more about publishing mozilla remote-settings servers and fetching that data in Firefox
- Understand how PSL is shipped currently and find appropriate locations where the PSL can be stored
- Learn more about remote-settings api in firefox
- Review code that is currently being using the some of the required functionality
- If possible try to contribute to the Firefox project

Community Bonding

- Seek guidance from mentor about the workflow and how things work at Mozilla
- Enrich knowledge about the uses of the PSL currently and how the new changes will affect other systems
- If possible start coding in this phase, to get a head-start

Week 1

Day	Tasks
Day 1	● Create the bash script to start using the remote-settings server. ● Set it up to upload attachments, test with some binary files.
Day 2	● Setup a bash or python script to download the PSL with CURL. ● Start work on the cleanup script that strips the raw .dat file of special symbols and whitespaces.
Day 3	● Test the cleanup script and finish if incomplete.
Day 4	● Continue working on the cleanup script.
Day 5	● Write a blog detailing the week's works.

Week 2

Day	Tasks
Day 1	● Test the output of the cleanup script. ● Start work on the **DAFSA** creation program.
Day 2	● Continue working on the dafsa creation script.
Day 3	● Continue working on the dafsa creation script.
Day 4	● Continue working on the dafsa creation script.

Day 5	● Continue working on the dafsa creation script. ● Write a blog detailing the week's works.

Week 3

Day	Tasks
Day 1	● Continue working on the dafsa creation script.
Day 2	● Request Access to the Stage/Prod Server. ● Meanwhile setup a local server to use the stage/prod functionality. ● Modify the upload script for multi sign off workflow.
Day 3	● Retest the previous work. ● Start creating a final unified script for download, processing and upload.
Day 4	● Create a script to calculate md5sum for the dafsa, tag it with current version number.
Day 5	● Write a blog detailing the week's works.

Week 4 (Evaluation Week)

Day	Tasks
Day 1	● Package the dafsa into an object with metadata used for verification and versioning. ● Test the script on open remote-settings server.
Day 2	● Testing the entire download, processing, upload script with multi user sign off on prod/stage server.
Day 3	● Buffer day for in order to accommodate any pending/overdue tasks from previous weeks. Head towards next day work as soon as done.
Day 4	● Buffer day for in order to accommodate any pending/overdue tasks from previous weeks. Head towards next day work as soon as done.
Day 5	● Buffer day for in order to accommodate any pending/overdue tasks from previous weeks. Head towards next day work as soon as done. ● Write a blog detailing the week’s works.

Week 5

Day	Tasks
Day 1	● Start working on the client side operations of the project. ● Setup prefs define the server to download from. ● Create a new javascript file to download from the server.
Day 2	● Improve the simple function to download test if download works.
Day 3	● Setup functions for both scenarios: ○ If the file exists ○ If the file does not exist
Day 4	● A new function to calculate the md5sum of the file downloaded and verify by matching against metadata.
Day 5	● Write a blog detailing the week’s works.

Week 6

Day	Tasks
Day 1	● Setup function to check the available space on user's device.
Day 2	● Setup function to read the version number of current download and redownload if new update available.
Day 3	● Function to retrigger download if download fails.
Day 4	● Reroute the download to the correct location in profiles.
Day 5	● Write a blog detailing the week's works.

Week 7

Day	Tasks
Day 1	● Clean up the previous build system in mozbuild file.
Day 2	● Cleanup the previous files used for dafsa creation, including the previous .dat file and make_dafsa.py
Day 3	● Locate the c++ files that read the dafsa. ● Make changes to read the binary from the new location.
Day 4	● Testing day to check if that no issues arise from last day's work.
Day 5	● Write a blog detailing the week's works.

Week 8 (Evaluation week)

Day	Tasks
Day 1	● Buffer day for in order to accommodate any pending/overdue tasks from previous weeks. Head towards next day work as soon as done.
Day 2	● Buffer day for in order to accommodate any pending/overdue tasks from previous weeks. Head towards next day work as soon as done.
Day 3	● Buffer day for in order to accommodate any pending/overdue tasks from previous weeks. Head towards next day work as soon as done.
Day 4	● Make the code cleaner and add documentation.
Day 5	● Write a blog detailing the week’s works.

Week 9

Day	Tasks
Day 1	● Make the code cleaner and add documentation.
Day 2	● Make the code cleaner and add documentation.
Day 3	● Make the code cleaner and add documentation.
Day 4	● Make the code cleaner and add documentation.
Day 5	● Write a blog detailing the week’s works.

Week 10

Day	Tasks
Day 1	● Testing day, test everything.
Day 2	● Testing day, test everything.
Day 3	● Testing day, test everything.
Day 4	● Testing day, test everything.
Day 5	● Write a blog detailing the week’s works.

Week 11 and Week 12

Buffer week in order to accommodate any pending/overdue tasks from previous weeks.

Post GSoC

Work on the other issues that were left for the making the GSoC project more consistent and manageable.
Write a blog detailing the work that I did and my experience in the last three months.

Personal Availability

I have reviewed the GSoC 2019 Timeline and I am agreeing that I will be fully available without any constraints during the time mentioned, I will be working 40 hours a week for the except in the last three weeks I will be partially off because the new semester starts in August. I will surely cover that up by working on weekends.

Personal Projects

- Wasm game of life (WebAssembly | Javascript | Rust)
- Oil Shipment POC (Solidity | Web3 JS)
  More Projects

Contributions to Open Source

Relevant projects:

- Add page title to about: pages that do not have a title
- Remote-settings

WebAssembly Studio
More Bugs

Internship Experience

In my last summer internship I worked as a blockchain developer at Splisys Consulting where I learned to work with Ethereum(solidity/web3js) and Node JS. I worked to create a supply chain management system that would be used by the company to track oil shipments from loading on the origin to unloading on the destination. The goal was to permanently store the data that is being provided by the various parties involved, this included simple data like dates of events and data about the shipment itself. The data would later be used for calculation of damage costs and late fees in case of dispute.

Academic Experience

I am majoring in Information Technology from Guru Gobind Singh Indraprastha University, New Delhi, I am a 3rd year student will be graduating next year. As part of my syllabus, I have so far studied majority of the subjects taught to undergraduates in computer science course. The subjects relevant to this project that I have studied are, Data Structures, Algorithms, Theory of Computation, Graph Theory.

Extracurriculars:

I have also finished a full stack web development classroom course from Coding Blocks. I have attended multiple national and state level hackathons and as a result have become adept at working in high stress environments and communicating well with teams.

Why Me?

As evident by my application above, I have a very good understanding of what the goals of this project are. As seen in the timeline and tasks sections, I am aware of the current Firefox codebase that I would be dealing with and the code that needs to be patched. The knowledge I have gained about the project and the tools required has been in a short amount of time. This all due to my strong work ethic and my eagerness to ask questions, the Mozilla IRC channels have helped me a lot in that regard. I am proficient in Python, Javascript and Bash the primary languages required for this project, I will have to deal with some C++ as well in the later leg of the project and I pledge to become more proficient in this language. I am comfortable working with the Mozilla infrastructure and tools as I have contributed to Firefox previously and wish to continue in the future.

Why Mozilla ?

While looking for GSoC projects for 2019, I first filtered the projects on the brainstorming page based on my skill set, ie. Javascript and Python. I tried to understand the projects well but this project caught my eye because this is an issue I myself have experienced while using Firefox. I did some research and after gaining a reasonable level of understanding of the project, I decided to go with this project. I had looked at many organisations. I even contributed to mozilla's WebAssembly Studio project and would have liked to contribute to it further but I left it in favour of working on Firefox. What excites me the most about this project is that I will be writing code for Firefox that will ship to millions of users across the world. This possibility alone is enough to get me pumped up.