Download Spam Protection for Firefox

Personal Details

Name	Sagar Bharadwaj KS
E-mail	sagarbharadwaj50@gmail.com
IRC Nick	sagarb_97
Telephone	+91 9663959969
Other contact methods	sagar.bharadwaj@yahoo.com **Linked in :** sagar-bharadwaj-k-s
Country of residence	India
Timezone	IST (UTC +5:30)
Primary language	English
Github ID	SagarB-97
Bugzilla Profile	Sagar Bharadwaj
Website	http://sagarbharadwaj.me/

Project Proposal

The project mainly involves implementing download spam protection in Firefox. The goal is to prevent sites from bombarding the browser with multiple non user initiated downloads.

This would mean that a mechanism to differentiate between user initiated and automatically initiated downloads has to be established as it is undesirable to annoy the user with unnecessary warning and permission prompts even when the user himself has requested for a download.

There are a number of bugs filed on Mozilla's bug tracking system, Bugzilla regarding this issue. There are mainly two proposed approaches. Each approach can be implemented by making certain changes either in the backend framework or in the front end code. I have described both the proposed approaches and the possible front end and backend alterations required.

Approach 1

Maintain a state machine for every open page.

- Initially the state is 'ALLOW_COUNT_DOWNLOADS'.
- After 'count' number of downloads are triggered, the state is transitioned to 'PROMPT_BEFORE_DOWNLOAD'. The value of count is to be determined. In chromium browser, the count is set to 1.
- Whenever a page in 'PROMPT_BEFORE_DOWNLOAD' state triggers a download, the user is shown a permission prompt, where it is confirmed if the user wants to permit multiple downloads.
- The page's state is set to 'ALLOW_DOWNLOADS' whenever currentDocument -> UserHasInteracted() transitions to true. This state allows all downloads as the user is now interacting with the page and is probably aware of all the downloads.

Backend alteration

If backend alteration is a feasible alternative, the nsIDocument object of every open page is associated with the state the page is currently in with respect to downloads.

A Sample code for the above approach would be :

```
const int ProtectionDownloadLimit = 1; // A global parameter. A Knob

class nsIDocument {
    ....
    enum DownloadProtectionState {
        ALLOW_COUNT_DOWNLOADS,
        PROMPT_BEFORE_DOWNLOAD,
        ALLOW_DOWNLOADS
    } downloadProtectionState ;

    // Number of downloads triggered by interacting with any child of this document object
    uint32_t downloadsTriggered;

    ...
}



nsIDocument:nsIDocument()
    : mUserHasInteracted(false),
    downloadProtectionState(ALLOW_COUNT_DOWNLOADS),
    downloadsTriggered(0)
{
    ...
}

void
nsIDocument::SetUserHasInteracted(bool aUserHasInteracted)
{
  mUserHasInteracted = aUserHasInteracted;
  if(mUserHasInteracted){
      downloadProtectionState = ALLOW_DOWNLOADS;
  }
}
```

Pseudo code when a download is triggered can be :

```
// document : a document object one of whose elements triggered a download

// A function called when download is triggered
// or a function that invokes nsDownloader::OnStartRequest()


void fn(...) {

    if(document->downloadProtectionState == ALLOW_DOWNLOADS){
        document->downloadsTriggered++;
        //Initiate the download without any prompt
    }
    else if(document->downloadProtectionState == PROMPT_BEFORE_DOWNLOAD){
        // Display warning or permission prompt if permission not already granted.
        // Take appropriate action based on user choice
    }

    else if(document->downloadsTriggered < ProtectionDownloadLimit &&
        document->downloadProtectionState == ALLOW_COUNT_DOWNLOADS){
            document->downloadsTriggered++;
            //Initiate the download without any prompt
    }
    else if(document->downloadsTriggered == downloadLimit) {
        document->downloadProtectionState = PROMPT_BEFORE_DOWNLOAD;
        // Display warning or permission prompt if permission not already granted.
        // Take appropriate action based on user choice
    }
}
```

However, after discussion with the mentor, it was realised that it is not advisable to alter the backend framework as the project is mostly front end oriented.

Thus an alternative approach is proposed.

Frontend alteration

- Create a new javascript module, 'DownloadSpamProtection.jsm'
- The module would mainly contain a map from the DownloadSource's referrer (DownloadSource.referrer) to a state : ALLOW_COUNT_DOWNLOADS, PROMPT_BEFORE_DOWNLOAD and ALLOW_DOWNLOADS and some basic functions (insertion, deletion, state change etc) associated with the map.
- The association between a referrer and its state is to ensure that one referrer cannot 'refer' more than 'count' downloads without user interaction. In cases where a Download object was initialised without DownloadSource object, an alternative will have to be discussed to determine the origin of the download.
- The Download.start() function which is main entry point to actually start a download would update the map object in DownloadSpamProtection
    - A new map entry will be created if referrer not already in the map.
    - If it is, associated count will be incremented or associated state change will happen. (ALLOW_COUNT_DOWNLOADS -> PROMT_BEFORE_DOWNLOAD). If state is PROMPT_BEFORE_DOWNLOAD, corresponding permission prompt is shown
    - Whenever EventStateHandling::isHandlingUserInput transitions to true, the state associated with that referrer is set to ALLOW_DOWNLOADS. (Similarly after user answers in the affirmative to the prompt)

Most permissions associated with a download such as parental controls and runtime permissions are resolved before actually starting a download in Download.start(). However, other user prompts such as the file explorer (where the user chooses the path to save a downloaded file) is shown while the actual download continues uninterrupted in the background.

This project being protection against spam, it would be better to follow the approach that's used in resolving parental controls and runtime permissions. This would provide the users protection against websites that intentionally try hogging the user's bandwidth. Thus, this is the approach that's proposed above.

Approach 2

I can observe two parallel but interdependent requirements in this approach:

- Implementing a mechanism to count the number of downloads that has been initiated from a particular origin - a webpage where the downloads are initiated

- Implementing a mechanism to differentiate between user initiated and automatic downloads. This is to prevent displaying warnings and permission dialogs when the user is aware of multiple downloads.

Corresponding to the above implementations, certain mensurations have to be instrumented based on the data collected to determine the effectiveness of the implementation. That is, depending on how many times the user allows or disallows a permission and also how many downloads are marked as user initiated, certain implementation decisions have to made in the future.

Implementing a mechanism to count the number of downloads

Backend alteration

In order to associate each session with the number of downloads triggered by it, a new property can be associated with the document object of that page.

```
const int ProtectionDownloadLimit = 1; // A global parameter

class nsIDocument {
    ....
    // Number of downloads triggered by interacting with any child of this document object
    uint32_t downloadsTriggered;
    ...
}
```

Note that the attribute is associated with the document of that tab and not with the origin. Only the permission request is associated with the domain to retain permanently for all tabs from the same domain.

Whenever a download is triggered :

```
// document : a document object one of whose elements triggered a download

// A function called when download is triggered
// or a function that invokes nsDownloader::OnStartRequest()
void fn(...) {

    if(document->downloadsTriggered < ProtectionDownloadLimit){
        document->downloadsTriggered++;
        //Initiate download without any prompt
    }
    else {
        // Display warning or permission prompt if permission not already granted.
        // Take appropriate action based on user choice
    }
}
```

The required document object can be accessed by traversing up the dom tree.

Front end alteration

The map object in DownloadSpamProtection described above can be used to associate the download's referrer to the number of downloads that has been initiated from this referrer.

Implementing a mechanism to differentiate between user initiated and automatic downloads

A number of approaches have been discussed in the comment thread regarding implementation of this mechanism.

There is **EventStateManager::IsHandlingUserInput()** which is used before granting Permission requests to determine if the request was user triggered or not. Bug 1345424 adds an attribute isHandlingUserInput() to nsIContentPermissionRequest. A similar approach can be used to differentiate user triggered and auto triggered downloads.

After implementing the feature that can bring about this differentiation, the above pseudo code can be altered to accommodate this feature.

```
    if(document->downloadsTriggered < ProtectionDownloadLimit){
        if(!isHandlingUserInput())
            document->downloadsTriggered++;
        //Initiate download without any prompt
    }
```

After some careful consideration, reading comments on associated bugs and some consultation with the project mentor, it seems **Approach 1** with **Front End alteration** is the best approach to be followed.

However there are some **crucial pointers/limitations** to be considered :

1. The state machine will have to be reset every time the user navigates out of a page. Not doing so will have the effect of granting a site temporary permission for the entire session to download multiple files just because of user's interaction
2. The approach suggested here is perhaps blindly trusting the credibility of a site once the user starts interacting with it. That is, all downloads initiated after user's first interaction is considered user initiated. As there is no direct method to mark downloads as user initiated or auto initiated, this work around is used. A better approach needs to be invented to protect from sites that try and auto initiate downloads in spite of user's constant interaction. This will be done in the community bonding period.

Schedule of Deliverables

April 23 - March 13 (Community bonding period)

This period will be used for conducting further research on the project to come up with a concrete implementation plan. Some possible research that will be done during this period includes :

- Possible solutions to differentiate between user initiated and auto initiated downloads. That is, further investigation of platform features such as EventStateManager::IsHandlingUserInput()
- Possible solutions to find the origin of a download if not for DownloadSource.referrer
- Deciding upon appropriate places to plug in the state change code. That is, determining where the state associated with a download will have to changed. Also deciding the place where the condition that determines if user interaction has taken place is evaluated.
- Implementation specifics of instrumenting various parts of the project will be discussed

This period will also be utilized to better acquaint myself with Mozillians and the Firefox code base

May 14 - May 21

Implementing the Map in DownloadSpamProtection described above.

This would include implementing all the associated functions such as adding a new entry into the map, deleting an entry from the map, changing the state of an entry in the map etc. The exact interface of the module will be finalised after discussions in the community bonding period

May 22 - May 24

Document the DownloadSpamProtection module.

I would also be writing some basic tests that use the APIs of this module in its raw form if required

May 24

Submit the DownloadProtectionModule for review.

Work on the review suggestions will be done alongside the remaining tasks.

May 25 - May 31

Plug in state changing code into the download flow

This period would involve discovering places where it would be appropriate to change the state associated with a download and implementing the state changing code. It would also include determining if user interaction has taken place and changing the state associated with a download if it has.

June 1 - June 6

Write automated tests to test the functionality of the implemented map

Write tests with some auto initiated and user initiated downloads to test if the state transitions as specified.

June 7

Submit the entire implementation of the flow for review

Work on review suggestions will be done in the following buffer period.

June 7 - June 10

Buffer period. Clean up code and prepare for Phase 1 evaluation. Wait for reviews on previously submitted code.

This period is a buffer period to complete any unfinished tasks. Complete documentation of DownloadSpamProtection module and cleaning up of test cases is done in this phase. The buffer is provided to wait for and work on review suggestions.

June 11 - June 15

Phase 1 evaluations. Read and research on implementing permissions and UI features in Firefox

Work on and wait for review suggestions on previous two submissions.

June 16 - June 23

Implementation of 'multiple downloads' Permission

Implement a permission which contains information about user preferences for downloads from a particular domain. This is preferably a temporary permission that is valid for that particular session.

June 23 - June 30

Implementation of Permission prompt UI

The UI for permission prompt is to be implemented according to the specs here.

The permission prompt can be shown using an nsIPrompt instance. There's already a module DownloadUIHelper to display prompts related to downloads. A new method 'allowMutlipleDownloads' will have to be added to accommodate for multiple downloads permission prompt.

From the UX specs for the multiple download permission prompt, it seems to me that prompts generated by nsIPromptService::confirmCheck() or nsIPromptService::confirmEx() would suffice. It seems

DownloadUIHelper module currently does not have provisions to generate permission style prompts. All the prompts currently shown are warnings. Some help will have to be taken from other modules such as PopupNotifications.jsm to generate the required permission style prompts.

Implementation details regarding the permission prompt UI will have to decided.

July 1 - July 4

Rewrite automated tests to verify the permission prompt functionality

July 4

Submit permission prompt implementation for Review.

Work on review suggestions will be done in the following buffer period.

July 5 - July 8

Buffer period. Complete any unfinished tasks. Wait for review suggestions on previous submissions.

Prepare for phase 2 evaluation.

July 9 - July 13

Phase 2 evaluation.

Read and research about Telemetry and data collection in Firefox. Wait for and work on review suggestions for previous three code submissions.

July 14 - July 21

Instrument the code that detects downloads started by user

July 21

Submit the instrumentation code for review.

July 22 - July 28

Instrument the user interaction with Permission prompt for multiple downloads

The implementation specifics of how instrumentation is carried out will be discussed in the community bonding period

July 28

Submit instrumentation code for review.

July 29 - August 6

Buffer period. Wait for code review

Complete any unfinished tasks. Complete the entire documentation and testing process. Prepare for final phase of evaluation. Wait for and work on any review suggestions on previous five code reviews.

Thus the schedule is divided into a total of five code reviews. The work on code review suggestions will be done alongside other stages of implementations.

A buffer period is given immediately after every code review submission in case the reviewer suggests a change in the basic interface in the initial stages of the review which might affect further stages of implementation.

Open Source Development Experience

I have some experience in Open source contribution. I am familiar with Mozilla's contribution workflow as I have resolved a few bugs. I am comfortable using Mercurial and Bugzilla.

Some Firefox bugs I've resolved:

https://bugzilla.mozilla.org/show_bug.cgi?id=1402485

https://bugzilla.mozilla.org/show_bug.cgi?id=1421491

https://bugzilla.mozilla.org/show_bug.cgi?id=1425724

https://bugzilla.mozilla.org/show_bug.cgi?id=1424234

Some of my other open source contributions can be found on my Github profile : SagarB-97

Some examples of my contributions:

https://github.com/sympy/sympy/pull/12108

https://github.com/sympy/sympy/pull/12112

https://github.com/sympy/sympy/pull/12167

https://github.com/sympy/sympy/pull/12176

Work experience and examples

I write code almost every day and many of my projects are hosted online as open source repositories. I have experience programming in a number of languages including javascript, C++, C and Python.

Some examples of my open source projects :

Javascript:

https://github.com/SagarB-97/Online-Quiz-Platform

https://github.com/Handwritten-Equation-Solver/Handwritten-Equation-Solver

https://github.com/OS-Simulator/OS-Simulator

C++:
https://github.com/SagarB-97/Low-Rate-TCP-DoS-Attack
https://github.com/SagarB-97/Sparse-Matrix-Vector-Multiplication
https://github.com/SagarB-97/CUDAcodes

Android Java:
https://github.com/SagarB-97/ArduinoController
https://github.com/SagarB-97/Grape-withAR
https://github.com/SagarB-97/Ghost

Python:
https://github.com/SagarB-97/PythonAutomations

Internship Experience

I interned in **Samsung R&D** in the Summer of 2017 (my sophomore year of undergraduation) .

Post : Software Engineering Intern

- Studied part of the code base implementing Layer 1 of 3G communication
- Unified Time Division Duplex (TDD) and Frequency Division Duplex(FDD) code bases written for Shannon chipsets and achieved 30% code size reduction
- Automated the process of unification by writing C parser scripts in Perl

This internship helped me realize the importance and seriousness of writing industry level production code, automated tests and proper documentation.

Academic Experience

I'm currently pursuing my third year of undergraduate studies in **National Institute of Technology, Karnataka Surathkal** (NITK Surathkal) in India. I'm in the **Computer Science** department.

Why I chose Computer Science?

I've always wanted to pursue my immense interest in the field of Computer Science since primary school. I, therefore, jumped at the opportunity of studying computer science in this renowned university.

How I'm doing?

I have a cumulative GPA of **9.28/10.0**.

Why Me?

I have a considerable amount of experience writing and contributing to open source resources. I'm already familiar with Mozilla's workflow and therefore will not need to spend time making myself comfortable with it. I'm familiar with Mozilla's bug tracking system, firefox's multiple repositories, interacting with the review board, commit authoring and other conventions.

I have all the skills required to complete this project. I've worked quite a bit with javascript as mentioned in the 'Work experience' section. I have a good understanding of how the web works as I've taken Data Communication, Computer networks and Advanced Computer networks courses through the period of three semesters. These courses included Web technologies in sufficient detail.

My webpage has sufficient detail about my skills, projects and also hosts my resume.

Why Mozilla?

I am inspired by the Mozilla community and their commitment to an open source culture. Mozilla is one of the biggest open source organizations and Mozilla Firefox if one of the most used browsers. I'm always excited to contribute to something that caters to the needs of millions of people.