Abhinav Vemulapalli

Weeks of March 7 and March 21

Goals for this Week

1. Work on JS fuzzer
    a. Finish function to randomly generate string input
    b. Finish adding the messages to fuzz
2. Research into CVE-2022-26486, a new sandbox bypass using the WebGPU IPC Framework
3. Work on and finish Journal 4
    a. Add the research I've done about CVE-2022-26486
    b. Explain the research I accomplished about the Firefox source code (trying to get rid of Firefox autocorrecting errors)
    c. Include my accomplishment of getting more of my fuzzer complete (include any research I completed to finish the fuzzer)

My Research and What I Learned

While this paper won't contain as much research I've had in previous papers, and the goals are also not plentiful, I've gotten a lot done (including more than my goals). To begin I researched something I didn't foresee happening but, I thought it would be helpful. Just recently, about a few months ago, Firefox patched two vulnerabilities: CVE-2022-26485 and CVE-2022-26486, both of which were being actively used together to perform full-chain exploits. From an earlier journal, full-chain exploits are when an attacker uses one exploit to compromise the JS interpreter in a child shell and then uses another exploit to bypass the sandbox protecting the rest of the system and gaining access to the system. After researching the CVE, I spent a lot of time just looking into the Firefox source code and understanding how different things work in the code and not just at a high level.

CVE-2022-26486

Although researchers found two vulnerabilities, only one of them is a sandbox bypass exploit. According to the Mozilla Foundation Security Advisory, CybersecurityHelp, and Zorz, CVE-2022-26585 is a use-after-free attack that occurs when processing the XSLT parameter. In addition, the three sources cite CVE-2022-26586 also being a use-after-free attack but this time occuring in the WebGPU IPC framework.

REMINDER: A use-after-free attack usually happens when the program creates an object in memory but then fails to properly remove that object, and continues to access the memory where that object was present. Therefore, artifacts of the object remain in memory and an attacker could later control what is present in that object and control what the program executes. For a more detailed explanation, refer back to Journal 1.

Zorz states that the “vulnerability affects Firefox, Firefox Focus, and Thunderbird” with attackers “exploit[ing] the vulnerabilities in the wild.” In addition, the Mozilla Foundation Security Advisory expands a little bit more on the vulnerability stating “an unexpected message in the WebGPU IPC framework could lead to a use-after-free and exploitable sandbox escape.” Lastly, attackers can remotely attack people with this vulnerability by tricking victims to “open a specially crafted web page [which] trigger[s] a use-after-free error, and execute[s] arbitrary code on the system” (“Remote Code Execution in Mozilla Firefox”). The brand new nature of the vulnerability means that there is no proof of concept available yet. Proof of concept just means that someone has published code that exploits the bug and proves that the bug is exploitable to the extent everyone is saying it is. Regardless, I decided I could try and poke around the source code and see if maybe the issue stands out to me. Given I knew the bug was around the WebGPU IPC framework, I opened the Firefox source code directory in VSCode and searched for all occurences of WebGPU. From there I found files titled `WebGPUChild.cpp` and `WebGPUParent.cpp` in the `dom/webgpu/ipc` folder. I tried going through the file to see if I could identify the use-after-free bug, however, I’m not experienced enough with C++ to identify exactly what’s going on and whether or not there is a use-after-free present. Given all my efforts, I couldn’t even find the bug let alone understand how the WebGPU IPC code was working. With that, I decided to spend more time on understanding other parts of the source code. Namely, the sections concerning JSActors and those IPC methods.

JSActors Internal Implementation

Quick disclaimer, while most of the content from here on was just revelations and understandings I came up with as I went through the source code, I’m including it under the research section because understanding the source code was comparable to researching how Firefox works so I could patch different parts of it. Understanding the source code would also deepen my comprehension of the IPC framework works and allows me to potentially write a better fuzzer. Most of my understanding of the source code also came from my prior computer science knowledge and research I’ve done. Regardless, I set out to understand the source code with two primary goals in mind: view everytime an IPC message was being sent with the data and patch Firefox to remove GUI elements. This section will focus on the former with the latter being explained in a later section. From my research in Journal 2, I knew that the way to interact with JSActors and such was through JavaScript (JS) with most Actors themselves being written in JS (there are a couple which aren’t. Most notable of which is the WebGPU, which is written in C++). However, I knew that the JSActors themselves and how they functioned were not implemented in JS. Following my intuition led me to a file title `JSActor.cpp` in the `dom/ipc/` folder. The file contained the code for how Mozilla implemented JSActors in C++ and all the underlying functionality of them. It also meant I had access to viewing the IPC calls and messaging deep within Firefox itself. The fact I had also compiled my own custom version of Firefox meant I could add my own code to this section, re-compile and build a new version of Firefox that uses the code I added. I go over what I did specifically in the accomplishments section, but I was able to succeed in my first goal. The lines of interest start around 198 and 220. Before explaining and showing the code at those lines, it’s important to remember that

Mozilla documented two primary methods for sending messages between the parent and child process. To be specific, JSActors rely on the `sendAsyncMessage` and `sendQuery` methods to send messages between parent and child actors. In fact, one of the most common pieces of code I've been using all this time, and showcased in multiple journals, uses the `sendQuery` method. As a refresher, here is how I used the `sendQuery` method to create alert prompts:

```
prom = await actor.sendQuery("Prompt:Open", {
    promptType: "dialog",
    title: "👻",
    modalType: 1,
    promptPrincipal: null,
    inPermutUnload: false,
    _remoteID: "id-lol"
});
```

As you can tell from the above snippet, the `sendQuery` method takes two parameters, the message name and the data. The `sendAsyncMessage` also takes the same two parameters and it's important to remember that both these methods take those as parameters. Another thing to keep in mind is that the above code has all been JavaScript. Going back to the code in JSActor.cpp, here is the code around line 198 and line 220 respectively:

```
void JSActor::SendAsyncMessage(JSContext* aCx, const nsAString&
              aMessageName, JS::Handle<JS::Value> aObj, ErrorResult& aRv) {
  Maybe<ipc::StructuredCloneData> data{std::in_place};
  if (!nsFrameMessageManager::GetParamsForMessage(
          aCx, aObj, JS::UndefinedHandleValue, *data)) {
    aRv.ThrowDataCloneError(nsPrintfCString(
        "Failed to serialize message '%s::%s'",
        NS_LossyConvertUTF16toASCII(aMessageName).get(), mName.get()));
    return;
  }

  JSActorMessageMeta meta;
  meta.actorName() = mName;
  meta.messageName() = aMessageName;
  meta.kind() = JSActorMessageKind::Message;

  SendRawMessage(meta, std::move(data), CaptureJSStack(aCx), aRv);
}
```

```
already_AddRefed<Promise> JSActor::SendQuery(JSContext* aCx, const
                                             nsAString& aMessageName,
                                             JS::Handle<JS::Value> aObj,
                                             ErrorResult& aRv) {
  Maybe<ipc::StructuredCloneData> data{std::in_place};
  if (!nsFrameMessageManager::GetParamsForMessage(
          aCx, aObj, JS::UndefinedHandleValue, *data)) {
    aRv.ThrowDataCloneError(nsPrintfCString(
        "Failed to serialize message '%s::%s'",
        NS_LossyConvertUTF16toASCII(aMessageName).get(), mName.get()));
    return nullptr;
  }

  nsIGlobalObject* global = xpc::CurrentNativeGlobal(aCx);
  if (NS_WARN_IF(!global)) {
    aRv.ThrowUnknownError("Unable to get current native global");
    return nullptr;
  }

  RefPtr<Promise> promise = Promise::Create(global, aRv);
  if (NS_WARN_IF(aRv.Failed())) {
    return nullptr;
  }

  JSActorMessageMeta meta;
  meta.actorName() = mName;
  meta.messageName() = aMessageName;
  meta.queryId() = mNextQueryId++;
  meta.kind() = JSActorMessageKind::Query;

  mPendingQueries.InsertOrUpdate(meta.queryId(),
                                 PendingQuery{promise,
meta.messageName()});

  SendRawMessage(meta, std::move(data), CaptureJSStack(aCx), aRv);
  return promise.forget();
}
```

Looking at the code for both, you can see that they both have a `meta.actorName()` and `meta.messageName()` which are both useful information to have, especially when debugging why certain IPC messages are working or not. At other times, it will also help with understanding exactly how Firefox is functioning when certain actions happen. In addition, the names of the methods are almost the exact same as the ones I've been using in JavaScript. The similarities hint at the fact that under the hood, the JS functions I've been using, essentially call the above two functions. That was about it for my research into JSActors internals but it proved very useful for achieving my first goal of understanding the source code. One quick note, because I'm using a custom built version of Firefox, I can run it from the command line with `./mach run` (if you look at my previous journal where I attempt to build Firefox, I use a similar but different command of `./mach build`). When I run the custom built version of Firefox from the command line like that, Firefox outputs tons of information into the terminal from errors to warnings to whatever code I could put. This has fairly significant implications which will be more apparent in my accomplishments section. Before I get there though, I do need to cover one last area of research I completed and that was regarding specific JSActors and their methods.

Specific JSActors and their Methods of Functionality

I first began my research efforts with PromptParent primarily because I was also trying to get rid of the GUI element. The reason why I'm trying to get rid of the GUI element is because when I try to run the fuzzer with random information, the program will hang because it needs someone to react to the prompt. Therefore, if I'm able to remove the prompt from the picture, I won't have to worry about that. This was also another one of my advisor's tips which will hopefully prove helpful. I started with going through the code for `PromptParent.jsm` in the `browser/actors/` folder. The first place I started was where it would check what kind of message the parent Actor received and made a decision about what to show depending on that.

```
if (
      (args.modalType === Ci.nsIPrompt.MODAL_TYPE_CONTENT &&
          !contentPromptSubDialog) ||
    (args.modalType === Ci.nsIPrompt.MODAL_TYPE_TAB &&
        !tabChromePromptSubDialog) ||
    this.isAboutAddonsOptionsPage(this.browsingContext)
        ) {
        return this.openContentPrompt(args, id);
      }
      return this.openPromptWithTabDialogBox(args);
}
```

In ths case, if the modal type being requested is the same as one of the above constants, or we're on the About:Addons page in Firefox, it will call the `openContentPrompt` method and display an alert that way. Otherwise, it will call the `openPromptWithTabDialogBox` method and make an alert prompt that way. The `openContentPrompt` method is as follows:

```
openContentPrompt(args, id) {
  let browser = this.browsingContext.top.embedderElement;
  if (!browser) {
    throw new Error("Cannot tab-prompt without a browser!");
  }
  let window = browser.ownerGlobal;
  let tabPrompt = window.gBrowser.getTabModalPromptBox(browser);
  let newPrompt;
  let needRemove = false;

  // If the page which called the prompt is different from the the top
context
  // where we show the prompt, ask the prompt implementation to display the
origin.
  // For example, this can happen if a cross origin subframe shows a
prompt.
  args.showCallerOrigin =
    args.promptPrincipal &&
    !browser.contentPrincipal.equals(args.promptPrincipal);

  let onPromptClose = () => {
    let promptData = gBrowserPrompts.get(this.browsingContext);
    if (!promptData || !promptData.has(id)) {
      throw new Error(
        "Failed to close a prompt since it wasn't registered for some
reason."
      );
    }

    let { resolver, tabModalPrompt } = promptData.get(id);
    // It's possible that we removed the prompt during the
    // appendPrompt call below. In that case, newPrompt will be
    // undefined. We set the needRemove flag to remember to remove
    // it right after we've finished adding it.
    if (tabModalPrompt) {
      tabPrompt.removePrompt(tabModalPrompt);
    } else {
      needRemove = true;
    }

    this.unregisterPrompt(id);

    PromptUtils.fireDialogEvent(
      window,
      "DOMModalDialogClosed",
      browser,
      this.getClosingEventDetail(args)
    );
    resolver(args);
    browser.maybeLeaveModalState();
  };

  try {
    browser.enterModalState();
    PromptUtils.fireDialogEvent(
      window,
      "DOMWillOpenModalDialog",
      browser,
      this.getOpenEventDetail(args)
    );

    args.promptActive = true;

    newPrompt = tabPrompt.appendPrompt(args, onPromptClose);
    let promise = this.registerPrompt(newPrompt, id);

    if (needRemove) {
      tabPrompt.removePrompt(newPrompt);
    }

    return promise;
  } catch (ex) {
    Cu.reportError(ex);
    onPromptClose(true);
  }

  return null;
}
```

The beginning appears to be a lot of checks to see if there is already a prompt open or not, set up necessary variables and functions for later use. For example, the above `onPromptClose` is a function they define for use later in the `openContentPrompt` function. The internal function's job is just like it name says, to run certain code when a prompt closes. However, the important part, the part that looks like it actually creates the prompt, is the the `try-catch` block. In that block, it tries to get the browser to enter a modal state which basically means to darken everything and ensure you can't use the rest of the browser until you interact with the prompt. Later on it goes to append the prompt to a queue and registers it which makes me think that behind the scenes, Firefox queues the different prompts so the browser can display them at appropriate times. In addition to that here is the code for the `openPromptWithTabDialogBox` method:

```
async openPromptWithTabDialogBox(args) {
  const COMMON_DIALOG = "chrome://global/content/commonDialog.xhtml";
  const SELECT_DIALOG = "chrome://global/content/selectDialog.xhtml";
  let uri = args.promptType == "select" ? SELECT_DIALOG : COMMON_DIALOG;

  let browsingContext = this.browsingContext.top;

  let browser = browsingContext.embedderElement;
  let promptRequiresBrowser =
    args.modalType === Services.prompt.MODAL_TYPE_TAB ||
    args.modalType === Services.prompt.MODAL_TYPE_CONTENT;
  if (promptRequiresBrowser && !browser) {
    let modal_type =
      args.modalType === Services.prompt.MODAL_TYPE_TAB ? "tab" :
"content";
    throw new Error(`Cannot ${modal_type}-prompt without a browser!`);
  }

  let win;

  // If we are a chrome actor we can use the associated chrome win.
  if (!browsingContext.isContent && browsingContext.window) {
    win = browsingContext.window;
  } else {
    win = browser?.ownerGlobal;
  }

  // There's a requirement for prompts to be blocked if a window is
  // passed and that window is hidden (eg, auth prompts are suppressed if
the
  // passed window is the hidden window).
  // See bug 875157 comment 30 for more..
  if (win?.winUtils && !win.winUtils.isParentWindowMainWidgetVisible) {
    throw new Error("Cannot open a prompt in a hidden window");
  }

  try {
    if (browser) {
      browser.enterModalState();
      PromptUtils.fireDialogEvent(
        win,
        "DOMWillOpenModalDialog",
        browser,
        this.getOpenEventDetail(args)
      );
    }

    args.promptAborted = false;
    args.openedWithTabDialog = true;

    // Convert args object to a prop bag for the dialog to consume.
    let bag;

    if (promptRequiresBrowser && win?.gBrowser?.getTabDialogBox) {
      // Tab or content level prompt
      let dialogBox = win.gBrowser.getTabDialogBox(browser);

      if (dialogBox._allowTabFocusByPromptPrincipal) {
        this.addTabSwitchCheckboxToArgs(dialogBox, args);
      }

      bag = PromptUtils.objectToPropBag(args);
      await dialogBox.open(
        uri,
        {
          features: "resizable=no",
          modalType: args.modalType,
          allowFocusCheckbox: args.allowFocusCheckbox,
        },
        bag
      ).closedPromise;
    } else {
      // Ensure we set the correct modal type at this point.
      // If we use window prompts as a fallback it may not be set.
      args.modalType = Services.prompt.MODAL_TYPE_WINDOW;
      // Window prompt
      bag = PromptUtils.objectToPropBag(args);
      Services.ww.openWindow(
        win,
        uri,
        "_blank",
        "centerscreen,chrome,modal,titlebar",
        bag
      );
    }

    PromptUtils.propBagToObject(bag, args);
  } finally {
    if (browser) {
      browser.maybeLeaveModalState();
      PromptUtils.fireDialogEvent(
        win,
        "DOMModalDialogClosed",
        browser,
        this.getClosingEventDetail(args)
      );
    }
  }
  return args;
}
```

Like the other method, this one also starts off with declaring variables that it will need for later use such as the browser object itself to display the prompt within. It has other checks like ensuring that it is trying to add the prompt to a browser object that actually exists. Similar to the other method, the bulk of displaying the alert appears to happen in the `try-catch` block. However this time, it was blatantly apparent which lines of code are responsible for opening the prompt depending on the type. The first is how it appears to open all the prompts:

```
// Tab or content level prompt
      let dialogBox = win.gBrowser.getTabDialogBox(browser);

      if (dialogBox._allowTabFocusByPromptPrincipal) {
        this.addTabSwitchCheckboxToArgs(dialogBox, args);
      }

      bag = PromptUtils.objectToPropBag(args);
      await dialogBox.open(
        uri,
        {
          features: "resizable=no",
          modalType: args.modalType,
          allowFocusCheckbox: args.allowFocusCheckbox,
        },
        bag
      ).closedPromise;
```

The `dialogBox.open` is what opens the alert prompt and also passes some of the data that we pass as part of the `JSActor.sendQuery` method. Particularly, it uses the `args.modalType` and `args.allowFocusCheckbox` to determine what kind of modal to display and another setting to enable or disable. The other way Firefox attempts to display prompts and alerts is with the following code:

```
// Ensure we set the correct modal type at this point.
      // If we use window prompts as a fallback it may not be set.
      args.modalType = Services.prompt.MODAL_TYPE_WINDOW;
      // Window prompt
      bag = PromptUtils.objectToPropBag(args);
      Services.ww.openWindow(
        win,
        uri,
        "_blank",
        "centerscreen,chrome,modal,titlebar",
        bag
      );
    }

    PromptUtils.propBagToObject(bag, args);
```

It appears that this code is what happens when there's not TabDialogueBox and the prompt doesn't require the browser. While, I may not truly understand what that means, the important part is that this could be another potential factor or code that is what ends up displaying the prompt. I believe that this could play a role primarily from the fact that it does `Services.ww.openWindow` with "`modal`" as one of the options. This could also be the code responsible for why `modalType` always defaults to 3 if I enter a number too high.

In addition to my revelations about the Prompt Actors, I also noticed something interesting to keep in mind when looking at the other actors. When going through the code for the `PopupBlockingParent.jsm` in `toolkit/actors/`. Some of the Actors also uses other JSActors to send messages. For example, in the aforementioned file, on line 99 there is the following code:

```
let actor = windowGlobal.getActor("PopupBlocking");
let popups = await actor.sendQuery("GetBlockedPopupList");
```

Which attempts to get the same Actor that the file is for and sends a message to get the list of blocked popups. Some further digging reveals that the `PopupBlockingChild.jsm` file checks for the “GetBlockedPopupList” message name. In addition, on line 157, there is another interesting line:

```
let actor = browsingContext.currentWindowGlobal.getActor("PopupBlocking");
actor.sendAsyncMessage("UnblockPopup", { index: popupIndex });
```

Again, some more digging revealed that the `PopupBlockingChild.jsm` file also checks for the “UnblockPopup” message. The biggest takeaway from this was that I should start looking elsewhere for messages to send for Actors. Until now, I’ve only been looking at message names in the switch statement. For example, in the `PromptParent.jsm`, this was what I was looking at:

```
receiveMessage(message) {
    let args = message.data;
    let id = args._remoteId;

    switch (message.name) {
      case "Prompt:Open": {
        if (
          (args.modalType === Ci.nsIPrompt.MODAL_TYPE_CONTENT &&
            !contentPromptSubDialog) ||
          (args.modalType === Ci.nsIPrompt.MODAL_TYPE_TAB &&
            !tabChromePromptSubDialog) ||
          this.isAboutAddonsOptionsPage(this.browsingContext)
        ) {
          return this.openContentPrompt(args, id);
        }
        return this.openPromptWithTabDialogBox(args);
      }
    }

    return undefined;
  }
```

Almost, every Actor I was looking at had a `receiveMessage` function which had a switch statement for the different messages. That’s where I understood the Prompt Actor uses the “Prompt:Open” message to open a prompt and so on. So after looking at the `PopupBlockingChild.jsm` file, I realized I should be looking through the entire file because the Parent could be communicating with the Child as well and if I can find a bug there, maybe that will lead to a bypass as well.

Conclusion

In the end, my research into the newly found CVE didn't pan out like I was hoping it would, mainly because the exploit is so new and so severe that Mozilla might not want too much details leaking out. Either way, it was somewhat beneficial because it got me comfortable with the idea of looking through the C++ source code and not just the JS part. It also got me more comfortable with C++ in general because I had to try and understand the WebGPU IPC code, however unsuccessful. In addition, my deep foray into the Firefox source code, like mentioned in my timeline, was extremely fruitful. Although, I've looked at the source code prior to the psat couple weeks, this was the deepest I've dove, and I've learned lots of valuable information. Mainly, things that'll help me with altering Firefox to better suit my fuzzing needs as well as altering Firefox so that I can output more helpful debug information.

Accomplishments

- The three biggest accomplishments for the past couple weeks was successfully compiling and building a Firefox that outputted additional debug messages and that didn't show a prompt. In addition to those two, my other biggest accomplishment was adding most of the messages and actors to my fuzzer and adding the function to generate random input.
- First, changing Firefox to output more useful debugging information. The first place I began was in the `JSActor.cpp` file. Particularly near the `SendQuery` method and `SendAsyncMessage` method.
  - From exposure to C++, I knew that I could use `std::cout` to print information to the terminal.
  - In order to do that, I also had to import the package that uses `std` with `#include <iostream>`
  - With the library imported, I could use std::cout to output whatever information I wanted in the two methods of interest. So, I added the following line of code to that method, just to start testing and see if it works.

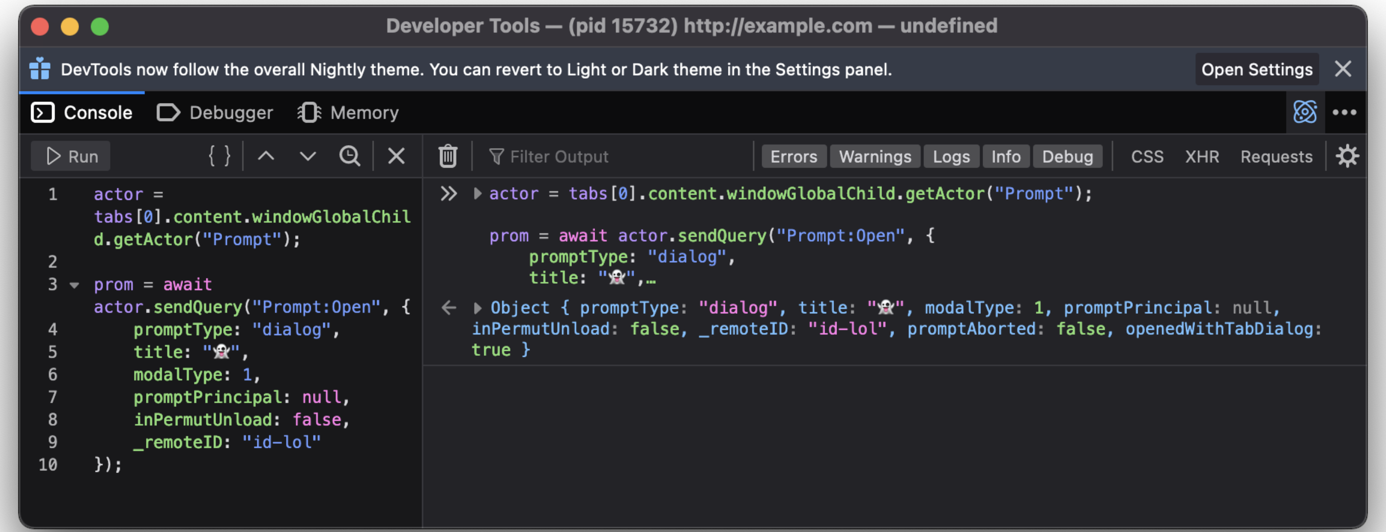
```
std::cout << "Message Received";
```

  - After adding that, I compiled like I usually do (`./mach build`), and then ran the custom built Firefox with `./mach run`. Once Firefox was running, I did my check of the IPC by sending a "Prompt:Open" message, and lo and behold, my terminal outputted "Message Received."
  - From there, it was time to step up the game and try outputting data. If you refer back to the code snippet above for the SendQuery method, you'll see two variables of interest: `mName` and `aMessageName`

- The former stores the Actor sending the message and the latter stores the actual message name being sent. So, to just start, I added the following line of code:

```
std::cout << "Sending Actor: " << mName << " with message: " <<
aMessageName << "\n";
```

- Once again, I built Firefox and ran it. I sent the message and got what I expected


Developer Tools — (pid 15732) http://example.com — undefined
DevTools now follow the overall Nightly theme. You can revert to Light or Dark theme in the Settings panel.
Open Settings
Console
Debugger
Memory
Run
Filter Output
Errors
Warnings
Logs
Info
Debug
CSS
XHR
Requests

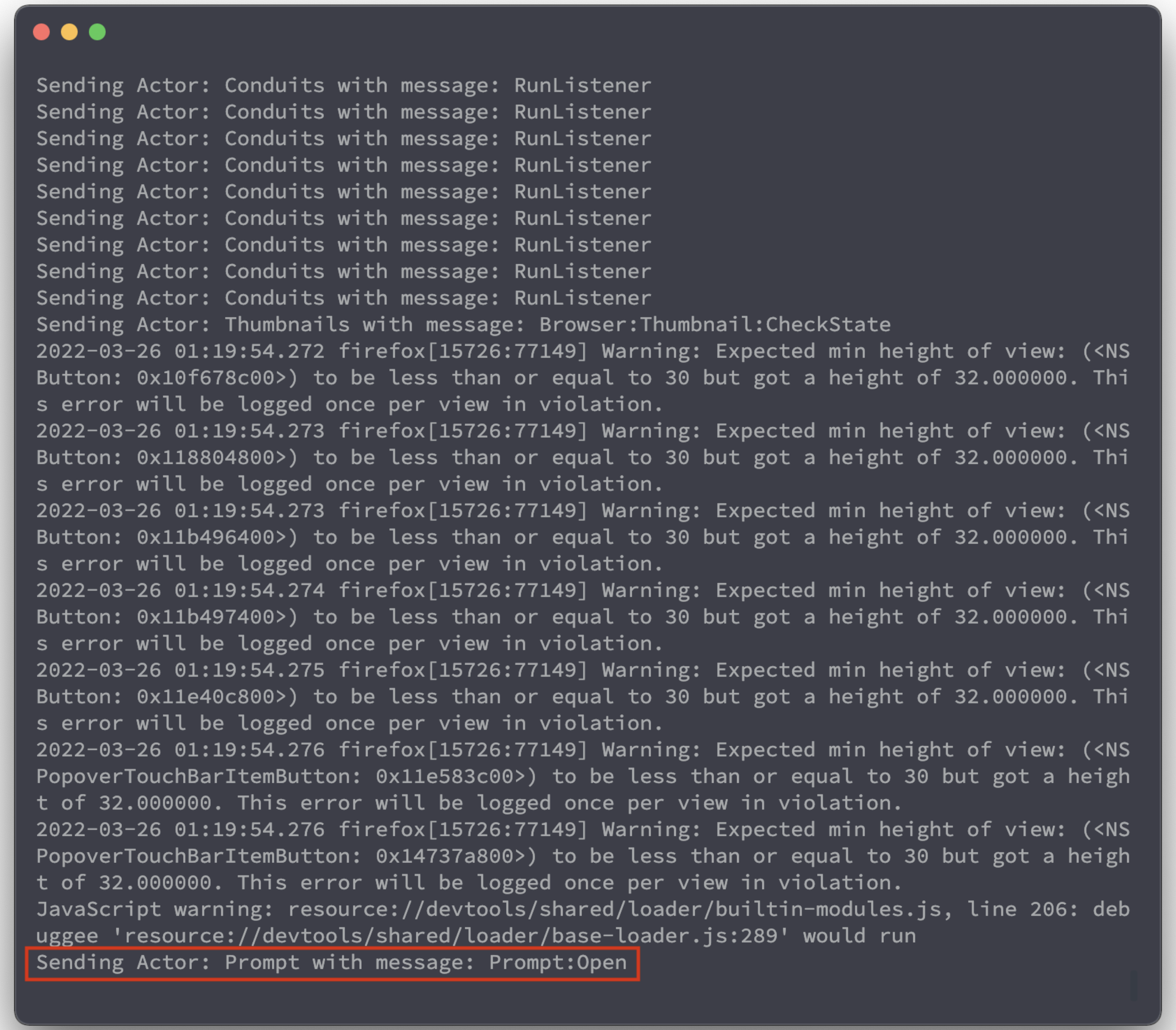
```
actor = tabs[0].content.windowGlobalChild.getActor("Prompt");

prom = await actor.sendQuery("Prompt:Open", {
    promptType: "dialog",
    title: "👻",
    modalType: 1,
    promptPrincipal: null,
    inPermutUnload: false,
    _remoteID: "id-lol"
});
```

```
» actor = tabs[0].content.windowGlobalChild.getActor("Prompt");

  prom = await actor.sendQuery("Prompt:Open", {
      promptType: "dialog",
      title: "👻",…
← Object { promptType: "dialog", title: "👻", modalType: 1, promptPrincipal: null,
  inPermutUnload: false, _remoteID: "id-lol", promptAborted: false, openedWithTabDialog:
  true }
```

Screenshot 1



```
Sending Actor: Conduits with message: RunListener
Sending Actor: Conduits with message: RunListener
Sending Actor: Conduits with message: RunListener
Sending Actor: Conduits with message: RunListener
Sending Actor: Conduits with message: RunListener
Sending Actor: Conduits with message: RunListener
Sending Actor: Conduits with message: RunListener
Sending Actor: Conduits with message: RunListener
Sending Actor: Conduits with message: RunListener
Sending Actor: Thumbnails with message: Browser:Thumbnail:CheckState
2022-03-26 01:19:54.272 firefox[15726:77149] Warning: Expected min height of view: (<NS
Button: 0x10f678c00>) to be less than or equal to 30 but got a height of 32.000000. Thi
s error will be logged once per view in violation.
2022-03-26 01:19:54.273 firefox[15726:77149] Warning: Expected min height of view: (<NS
Button: 0x118804800>) to be less than or equal to 30 but got a height of 32.000000. Thi
s error will be logged once per view in violation.
2022-03-26 01:19:54.273 firefox[15726:77149] Warning: Expected min height of view: (<NS
Button: 0x11b496400>) to be less than or equal to 30 but got a height of 32.000000. Thi
s error will be logged once per view in violation.
2022-03-26 01:19:54.274 firefox[15726:77149] Warning: Expected min height of view: (<NS
Button: 0x11b497400>) to be less than or equal to 30 but got a height of 32.000000. Thi
s error will be logged once per view in violation.
2022-03-26 01:19:54.275 firefox[15726:77149] Warning: Expected min height of view: (<NS
Button: 0x11e40c800>) to be less than or equal to 30 but got a height of 32.000000. Thi
s error will be logged once per view in violation.
2022-03-26 01:19:54.276 firefox[15726:77149] Warning: Expected min height of view: (<NS
PopoverTouchBarItemButton: 0x11e583c00>) to be less than or equal to 30 but got a heigh
t of 32.000000. This error will be logged once per view in violation.
2022-03-26 01:19:54.276 firefox[15726:77149] Warning: Expected min height of view: (<NS
PopoverTouchBarItemButton: 0x14737a800>) to be less than or equal to 30 but got a heigh
t of 32.000000. This error will be logged once per view in violation.
JavaScript warning: resource://devtools/shared/loader/builtin-modules.js, line 206: deb
uggee 'resource://devtools/shared/loader/base-loader.js:289' would run
Sending Actor: Prompt with message: Prompt:Open
```

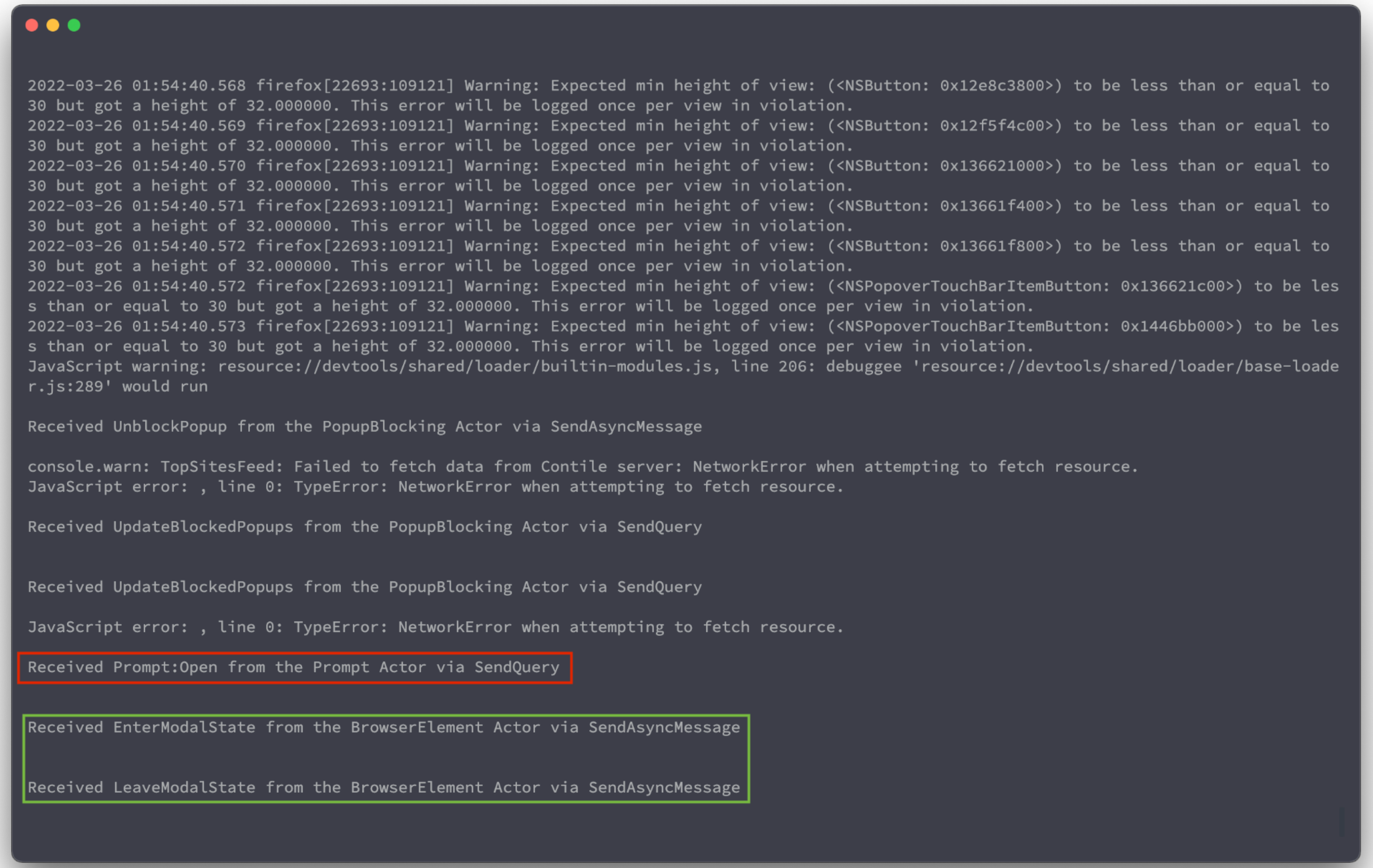
Screenshot 2

- In screenshot 1, you can see me doing what I've been doing for the past 3 journals, sending an IPC message via JSActors to open a prompt.
- In screenshot 2, you can see the output of my custom built Firefox and how there's a lot of debug info, but the red highlighted box is a result of the IPC message I sent. It also contains the information we expected. The actor I was using was Prompt and the message I send was Prompt:Open.
- My success here helped increase my confidence in writing C++ code, however minimal, so I can patch Firefox.

- My success here also meant that whenever I was struggling with a message not working in the Browser Content Toolbox (the place where I sent the Prompt:Open message in Screenshot 1), I could look at the terminal and see if the message was even being sent or if there were other errors.
- I wasn't happy with how it was being outputted so I just altered it to:

```
std::cout << "\nReceived " << aMessageName << " from the " << mName << "
Actor via SendQuery\n\n";
```

- I also added the same thing to `SendAsyncMessage` but replaced `SendQuery` in the code with the respective value.



```
2022-03-26 01:54:40.568 firefox[22693:109121] Warning: Expected min height of view: (<NSButton: 0x12e8c3800>) to be less than or equal to
30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-03-26 01:54:40.569 firefox[22693:109121] Warning: Expected min height of view: (<NSButton: 0x12f5f4c00>) to be less than or equal to
30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-03-26 01:54:40.570 firefox[22693:109121] Warning: Expected min height of view: (<NSButton: 0x136621000>) to be less than or equal to
30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-03-26 01:54:40.571 firefox[22693:109121] Warning: Expected min height of view: (<NSButton: 0x13661f400>) to be less than or equal to
30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-03-26 01:54:40.572 firefox[22693:109121] Warning: Expected min height of view: (<NSButton: 0x13661f800>) to be less than or equal to
30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-03-26 01:54:40.572 firefox[22693:109121] Warning: Expected min height of view: (<NSPopoverTouchBarItemButton: 0x136621c00>) to be les
s than or equal to 30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-03-26 01:54:40.573 firefox[22693:109121] Warning: Expected min height of view: (<NSPopoverTouchBarItemButton: 0x1446bb000>) to be les
s than or equal to 30 but got a height of 32.000000. This error will be logged once per view in violation.
JavaScript warning: resource://devtools/shared/loader/builtin-modules.js, line 206: debuggee 'resource://devtools/shared/loader/base-loade
r.js:289' would run

Received UnblockPopup from the PopupBlocking Actor via SendAsyncMessage

console.warn: TopSitesFeed: Failed to fetch data from Contile server: NetworkError when attempting to fetch resource.
JavaScript error: , line 0: TypeError: NetworkError when attempting to fetch resource.

Received UpdateBlockedPopups from the PopupBlocking Actor via SendQuery


Received UpdateBlockedPopups from the PopupBlocking Actor via SendQuery

JavaScript error: , line 0: TypeError: NetworkError when attempting to fetch resource.

Received Prompt:Open from the Prompt Actor via SendQuery


Received EnterModalState from the BrowserElement Actor via SendAsyncMessage


Received LeaveModalState from the BrowserElement Actor via SendAsyncMessage
```

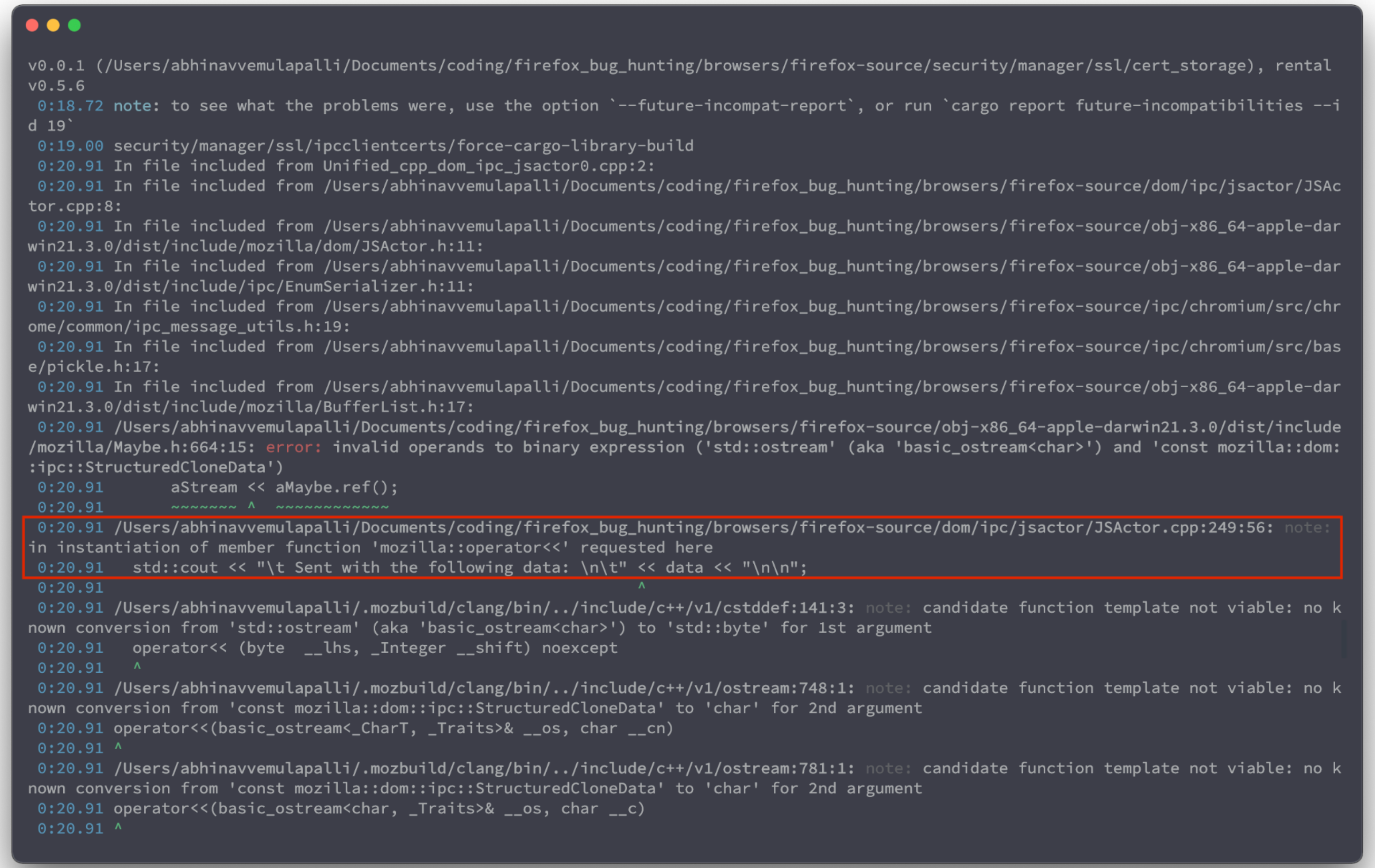
Screenshot 3

- As you can see from screenshot 3, after adding that new code that's how the output looked. The above screenshot also shows how even sending a message as simple as "Prompt:Open" also calls other internal messages as you can see from the green highlighted box.

- With my successes there, I noticed that there was another variable, `data` which, as the variable name says, contains the data that we send in `sendQuery`. I figured that being able to view that data would also prove immensely useful when debugging so I added the following code:

```
std::cout << "\t Sent with the following data: \n\t" << data << "\n\n";
```

- It does the same thing as the above code but tries to format it nicely by adding tabs (`\t`) and just outputting the contents of the data variable.
- However, when I tried to compile this version, it failed outputting the following error, highlighted in red.



```
v0.0.1 (/Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/security/manager/ssl/cert_storage), rental
v0.5.6
 0:18.72 note: to see what the problems were, use the option `--future-incompat-report`, or run `cargo report future-incompatibilities --i
d 19`
 0:19.00 security/manager/ssl/ipcclientcerts/force-cargo-library-build
 0:20.91 In file included from Unified_cpp_dom_ipc_jsactor0.cpp:2:
 0:20.91 In file included from /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/dom/ipc/jsactor/JSAc
tor.cpp:8:
 0:20.91 In file included from /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/obj-x86_64-apple-dar
win21.3.0/dist/include/mozilla/dom/JSActor.h:11:
 0:20.91 In file included from /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/obj-x86_64-apple-dar
win21.3.0/dist/include/ipc/EnumSerializer.h:11:
 0:20.91 In file included from /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/ipc/chromium/src/chr
ome/common/ipc_message_utils.h:19:
 0:20.91 In file included from /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/ipc/chromium/src/bas
e/pickle.h:17:
 0:20.91 In file included from /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/obj-x86_64-apple-dar
win21.3.0/dist/include/mozilla/BufferList.h:17:
 0:20.91 /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/obj-x86_64-apple-darwin21.3.0/dist/include
/mozilla/Maybe.h:664:15: error: invalid operands to binary expression ('std::ostream' (aka 'basic_ostream<char>') and 'const mozilla::dom:
:ipc::StructuredCloneData')
 0:20.91       aStream << aMaybe.ref();
 0:20.91       ~~~~~~~ ^  ~~~~~~~~~~~~~
 0:20.91 /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/dom/ipc/jsactor/JSActor.cpp:249:56: note:
in instantiation of member function 'mozilla::operator<<' requested here
 0:20.91   std::cout << "\t Sent with the following data: \n\t" << data << "\n\n";
 0:20.91                                                        ^
 0:20.91 /Users/abhinavvemulapalli/.mozbuild/clang/bin/../include/c++/v1/cstddef:141:3: note: candidate function template not viable: no k
nown conversion from 'std::ostream' (aka 'basic_ostream<char>') to 'std::byte' for 1st argument
 0:20.91   operator<< (byte  __lhs, _Integer __shift) noexcept
 0:20.91   ^
 0:20.91 /Users/abhinavvemulapalli/.mozbuild/clang/bin/../include/c++/v1/ostream:748:1: note: candidate function template not viable: no k
nown conversion from 'const mozilla::dom::ipc::StructuredCloneData' to 'char' for 2nd argument
 0:20.91 operator<<(basic_ostream<_CharT, _Traits>& __os, char __cn)
 0:20.91 ^
 0:20.91 /Users/abhinavvemulapalli/.mozbuild/clang/bin/../include/c++/v1/ostream:781:1: note: candidate function template not viable: no k
nown conversion from 'const mozilla::dom::ipc::StructuredCloneData' to 'char' for 2nd argument
 0:20.91 operator<<(basic_ostream<char, _Traits>& __os, char __c)
 0:20.91 ^
```

Screenshot 4

- While the error may seem hard to decipher, it's a fairly simple error that I've encountered in other languages. The `std::cout` function is expecting the arguments I’m passing to it to be of a certain type, be it a string or number or whatever. However, the function doesn’t know what to do with the data type that the variable `data` is.

- This was one of the biggest roadblocks I faced in the past couple of weeks and am hoping that my advisor will be able to potentially help me with it, especially considering that he has much more experience in C++ than me.

- My second biggest accomplishment was getting the prompt to not show up in the GUI but still happen in the backend (that is, the application doesn’t freeze up).
    - To do so, I altered the content of the PromptParent.jsm file.
    - More particularly I commented out the entire `try-catch` block in the `openContentPrompt` method.
    - In addition, I commented out the following sections in the `openPromptWithTabDialogBox` method:

```
// Tab or content level prompt
      let dialogBox = win.gBrowser.getTabDialogBox(browser);

      if (dialogBox._allowTabFocusByPromptPrincipal) {
        this.addTabSwitchCheckboxToArgs(dialogBox, args);
      }

      bag = PromptUtils.objectToPropBag(args);
      await dialogBox.open(
        uri,
        {
          features: "resizable=no",
          modalType: args.modalType,
          allowFocusCheckbox: args.allowFocusCheckbox,
        },
        bag
      ).closedPromise;
```

- Particularly, I commented out the code from the `dialogBox.open` to the `closedPromise` section

```
// Ensure we set the correct modal type at this point.
      // If we use window prompts as a fallback it may not be set.
      args.modalType = Services.prompt.MODAL_TYPE_WINDOW;
      // Window prompt
      bag = PromptUtils.objectToPropBag(args);
      Services.ww.openWindow(
        win,
        uri,
        "_blank",
        "centerscreen,chrome,modal,titlebar",
        bag
      );
    }

    PromptUtils.propBagToObject(bag, args);
```

    - In the above section, I commented out the `Services.ww.openWindow` till its close.
    - With the three sections commented, anytime I send a “Prompt:Open” message, I get the output of Screenshot 1, but I don’t get a physical prompt to interact with.
- The last accomplishment of mine was finishing the majority/the most important part of my fuzzer: adding the function to randomly generate input and adding most of the messages
    - To being, I created a function called generate_random_input which takes in one required argument and two optional arguments.
    - The `type` argument determines whether or not I return a random string, integer, or float
    - If I specify string for the type, I can specify how long of a string I want. If I don’t specify a length, it will default to 8
    - If I specify integer or float for the type, I can specify the highest number it can generate. If I don’t specify a number, it defaults to 100. I’ll put the entire function after and a screenshot of the function working.

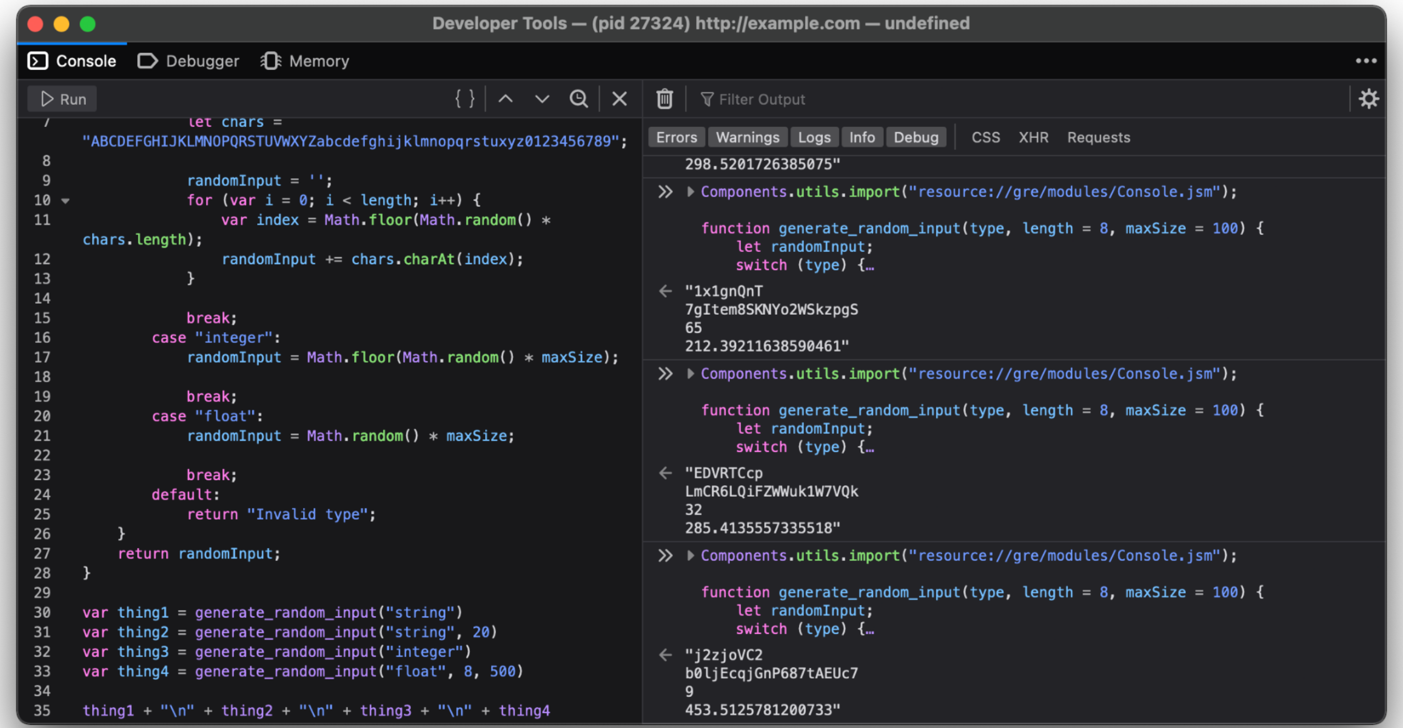
```
function generate_random_input(type, length = 8, maxSize = 100) {
    let randomInput;
    switch (type) {
        case "string":
            let chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuxyz0123456789";

            randomInput = '';
            for (var i = 0; i < length; i++) {
                var index = Math.floor(Math.random() * chars.length);
                randomInput += chars.charAt(index);
            }

            break;
        case "integer":
            randomInput = Math.floor(Math.random() * maxSize);

            break;
        case "float":
            randomInput = Math.random() * maxSize;

            break;
        default:
            return "Invalid type";
    }
    return randomInput;
}
```


Developer Tools — (pid 27324) http://example.com — undefined

```
            let chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuxyz0123456789";

            randomInput = '';
            for (var i = 0; i < length; i++) {
                var index = Math.floor(Math.random() *
chars.length);
                randomInput += chars.charAt(index);
            }

            break;
        case "integer":
            randomInput = Math.floor(Math.random() * maxSize);

            break;
        case "float":
            randomInput = Math.random() * maxSize;

            break;
        default:
            return "Invalid type";
    }
    return randomInput;
}

var thing1 = generate_random_input("string")
var thing2 = generate_random_input("string", 20)
var thing3 = generate_random_input("integer")
var thing4 = generate_random_input("float", 8, 500)

thing1 + "\n" + thing2 + "\n" + thing3 + "\n" + thing4
```

```
298.5201726385075"
>> ▸ Components.utils.import("resource://gre/modules/Console.jsm");

     function generate_random_input(type, length = 8, maxSize = 100) {
         let randomInput;
         switch (type) {…
← "1x1gnQnT
7gItem8SKNYo2WSkzpgS
65
212.39211638590461"
>> ▸ Components.utils.import("resource://gre/modules/Console.jsm");

     function generate_random_input(type, length = 8, maxSize = 100) {
         let randomInput;
         switch (type) {…
← "EDVRTCcp
LmCR6LQiFZWWuk1W7VQk
32
285.4135557335518"
>> ▸ Components.utils.import("resource://gre/modules/Console.jsm");

     function generate_random_input(type, length = 8, maxSize = 100) {
         let randomInput;
         switch (type) {…
← "j2zjoVC2
b0ljEcqjGnP687tAEUc7
9
453.5125781200733"
```

Screenshot 5

- Lastly, I added code for all the other messages I could find. There were two in particular that were fairly complicated, so I still want to add the messages and such for them, but it will take longer. I've attached all of the code for my fuzzer below.

```
Components.utils.import("resource://gre/modules/Console.jsm");
```

```
//
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Ob
jects/Math/random
function generate_random_input(type, length = 8, maxSize = 100) {
    let randomInput;
    switch (type) {
        case "string":
            let chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuxyz0123456789";

            randomInput = '';
            for (var i = 0; i < length; i++) {
                var index = Math.floor(Math.random() * chars.length);
                randomInput += chars.charAt(index);
            }

            break;
        case "integer":
            randomInput = Math.floor(Math.random() * maxSize);

            break;
        case "float":
            randomInput = Math.random() * maxSize;

            break;
        default:
            return "Invalid type";
    }
    return randomInput;
}

let ipcJSActors = {
    "ClickHandler": "Content:Click",
    // seems to need some other kind of privilege "ContentSearch": "",
    "FormValidation": "FormValidation:ShowPopup", // SOME OF THE
PARAMETERS; it doesn't fully work though
    //actor.sendQuery("FormValidation:ShowPopup", {position: "after_start",
screenRect: {width: 100, left: 150, top: 250, height: 100}, message:
"Testing"})
    "Pdfjs": {
        "PDFJS:Parent:saveURL": {
            blobURL: "string",
            filename: "string"
        }
    },
    //"Plugin": "RequestPlugins",
    "Prompt": {
        "Prompt:Open": {
            promptType: "string",
            title: "👻",
            modalType: 1,
            promptPrincipal: null,
            inPermutUnload: false,
            _remoteID: "id-lol"
        },
    },
    // "ScreenshotsComponent": "",
    // "WebRTC": "",
    //"LoginManager": "",
    // "PictureInPicture": "", will work on this later though
    "PopupBlocking": {
        "UpdateBlockedPopups": {
            count: "integer",
            shouldNotify: false,
        },
        "UnblockPopup": {
            index: "integer",
        },
        "GetBlockedPopupList"
    },
    // "Printing": "", will work on this later
}

// actor = tabs[0].content.windowGlobalChild.getActor("Prompt");

// prom = await actor.sendQuery("Prompt:Open", {
//     promptType: "dialog",
//     title: "👻",
//     modalType: 1,
//     promptPrincipal: null,
//     inPermutUnload: false,
//     _remoteID: "id-lol"
// });
```

- So, in the above code, I've declared a dictionary called ipcJSActors which have the actor followed by another dictionary containing all the messages in that actor I want to fuzz. The value for a message could be another dictionary, if that message requires any data to be sent. In that "data" dictionary, I just have the property the message requires and the type of data it is expected to send. If it's a boolean (true or false), I just arbitrarily chose a value.

Reflection on Goals and Timeline

Overall, even though I didn't have many goals for this journal, I got a lot done. I even got more done than I originally set out with my goals. I was able to understand more of the source code than I set for myself in my goals. I was also able to patch Firefox successfully even though I never explicitly had it in my goals. As for my timeline, I still think I'm on track, but I'm starting to feel a little overwhelmed primarily because I need to start thinking about my presentation and all the things that go with that. Otherwise, I'm on track to completing my fuzzer (and even start running my fuzzer and testing it) by the presentation.