Weeks of March 28 and April 11

Goals for this Week

1. Work on JS fuzzer
    a. Connect the data structure with an actual method to send the JSActor messages
    b. Write a loop to constantly send messages with JSActors
2. Try to understand how to print the data being sent in the JSActor message by altering Firefox
3. Start working on presentation slides
4. Work on a website to make it more professional
5. Polish the timeline to make it look good
6. Keep poking around Firefox source code and try and get the data of JS Actors printing
7. Read some research about the security standing of Firefox and Chrome (https://madaidans-insecurities.github.io/firefox-chromium.html)
8. Work on and finish Journal 5
    a. Add the knowledge I learned from the different security standings of Firefox and Chrome
    b. Include my process of getting my fuzzer working
9. Work on the board
    a. Plan out what I'm going to put on board
    b. Get dimensions for every picture and get foam core cut
    c. Glue on the different printed pictures and text
10. Work on and practice the presentation

My Research and What I Learned

The biggest part of my research for the past couple weeks was to find some more information about the low-level implementation of JSActors by perusing the C++ code. In addition, I found a resource comparing the security standing of Firefox and Chrome that I spent a lot of time understanding.

Internal JSActors Implementation and Other Firefox Things

Last time, I had understood enough of the JSActor Implementation to be able to modify it to print everytime `SendQuery` and `SendAsyncMessage` was called. While this is useful for troubleshooting, it would also be incredibly useful to view the exact data that is being sent with the messages. For example, I tried to reverse how the `PictureInPicture` IPC messages

work, however, I couldn't figure out what data to send in the `videoData` variable. Refer to the code below for how a PictureInPicture could work.

```
class PictureInPictureLauncherParent extends JSWindowActorParent {
  receiveMessage(aMessage) {
    switch (aMessage.name) {
      case "PictureInPicture:Request": {
        let videoData = aMessage.data;
        PictureInPicture.handlePictureInPictureRequest(this.manager,
videoData);
        break;
      }
    }
  }
}
```

The issue was that I tried to go to different files to find where the code would send the above message, but I couldn't figure out what the exact data sent looked like. If I could configure Firefox to display all the data that an IPC message contains, then it would be easier to reverse engineer how the message works. If Firefox were configured that way, then I could use the browser like usual and enter `PictureInPicture` mode for a video and just see what the browser calls internally with what data. As a reminder, I will post the code for the SendQuery method below for reference later on in the journal.

```
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;

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

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

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

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

From the above, there is a variable called `data` which conceivably contains the data that is being sent. The issue with printing that in C++ is that the `data` variable is a complex object which `std::cout` can't print out. The `std::cout` function expects its input to be of a certain type. So, with the advice of my advisor, I tried to figure out how the data structure of that variable is programmed. Looking at the line where the variable is declared, we can see that the type of the variable is an `Maybe<ipc::StructuredCloneData>`. Although I'm not entirely sure what the `Maybe` data structure is, I assume that it's just a way of checking whether or not there truly is data in the C++ code. This checks out because there are a couple IPC messages that don't send any data or take any data as required input. The more important data structure that actually stores the data is `ipc::StructuredCloneData`.

```
  Maybe<ipc::StructuredCloneData> data{std::in_place};
```

So, I tried to find all the references to `StructuredCloneData` in the entire project in order to see if I could print individual parts of the data structure. Searching the project helped me find a file called `StructuredCloneData.h`. The file is called a header file and defines how certain things should be structured. This was perfect because it would help me understand the different functions and variables the `StructuredCloneData` structure contained. I could then use that to specify what to print to the terminal screen. The biggest parts of the file that I found interesting was the following.

```
JSStructuredCloneData& Data() {
    return mSharedData ? mSharedData->Data() : mExternalData;
}

size_t DataLength() const {
    return mSharedData ? mSharedData->DataLength() : mExternalData.Size();
}

// For IPC serialization
void WriteIPCParams(IPC::Message* aMessage) const;
bool ReadIPCParams(const IPC::Message* aMessage, PickleIterator* aIter);
```

I'll detail the specific changes that I made in the code in the accomplishments section but this was another great start to understanding how JSActors are also implemented. The first function, `Data()`, appears to just return another data structure depending on what the `StructuredCloneData` contains. The `DataLength()` appears to just return how big the data is. The last two functions appear to be specific for IPC. The term serialization just means converting from one type of data to another so the other application can understand it. The only reason I was interested in those two functions is because it could potentially do something with IPC serialization. However, I'm not sure exactly how it works from just looking at the function definitions, so I would have to do some further code review to understand how those methods are used.

The other research I did with with regards to Firefox internals and such was trying to understand the bugs behind the two recent CVEs: CVE-2022-26485 and CVE-2022-26486. The way CVEs and bugfixes generally work is that everytime a CVE is found, the people working on Firefox create a new bug and assign it an identifier. Everytime the codebase is updated through something called commits, the updater usually includes the bug identifier the commit is aiming to fix (unless the commit is attempting to add a new feature). So, according to Mozilla's Security Advisory website for the two new CVEs, each one had the bug identifiers of 1758062 and 1758070 respectively. Luckily, Mozilla has a way of searching through previous commits online at https://hg.mozilla.org/mozilla-central/pushloghtml. After setting the time frame from 2 months ago to now, I searched for the above two bug identifiers. I couldn't find the latter bug which was the sandbox escape. However, I found the former bug which had to do with the XSLT processing bug. While that may not pertain to the scope of this project, it could be very useful

for my future plans and the fact that I was able to find the bug fix and the code that was changed as a result means that I can use that to my advantage and try to understand how other, more complicated sandbox escape bugs work. Here is the link to a page for a specific commit: https://hg.mozilla.org/mozilla-central/rev/49b96467418f45df768e5570bb59c22fe5c0bc10. The page includes the old code plus the new code that fixed the vulnerability.

Firefox Security Standing

The next big research topic I did was read an article about the security standing between Firefox and Chrome. The article was by Madaidans and was on his blogpost. The article was incredibly detailed but for the most part it highlighted the differences in sandboxing procedures between Chrome and Firefox. For starters, Madaidans states that the “current state [of Fission] is not as mature as Chromium’s site isolation.” Fission is Mozilla’s name for the project that is attempting to implement site isolation in their Firefox browser. Madaidan also states that there are multiple cross-site leaks within their implementation of Fission and allows “a compromised content process to access the data of another and bypass site isolation.” He also goes specifically into the implementations on both Windows and Linux specifically with Firefox’s sandbox implementation on Windows being the most comparable to the Chromium sandbox. The Linux side of sandbox escapes seem interesting because Madaidan states that Fission is “susceptible to various trivial sandbox escape vulnerabilities that span back years.” He even provides a couple examples such as using X11 to escape the sandbox. He also talks about seccomp-bpf which is a sandboxing technology on Linux but that isn’t implemented in a restrictive manner by Firefox. In addition, Madaidan describes how “Firefox has no separate GPU process” which means it cannot be independently sandboxed. This is very interesting because the recent CVE related to sandbox escape had to do with the WebGPU IPC messages. In addition, on Windows, there is a separate GPU process but sandboxing has not been enabled for it yet. The article also delves into exploit mitigations later on such as Firefox not having Arbitrary Code Guard (ACG) enabled. ACG is a way of ensuring that unexpected code doesn’t run, that is code potentially inserted by a hacker via a buffer overflow or another attack method. At the same time, Chromium hasn’t implemented ACG for every single one of their processes either; it’s only implemented it for the Audio process, while Firefox hasn’t implemented/enabled it for anything. Overall, Chromium has a more mature security standing than Firefox, and there are multiple implications for me finding this. For one, I can use this to find new paths to explore to potentially find a bug. It also has a more immediate benefit that I can present this to my audience and provide them with some interesting information they can take away. Other than that, I didn’t have much research done and most of my time was devoted to writing and finishing my fuzzer and a custom build of Firefox.

Accomplishments

- My biggest accomplishment by far was having completed my fuzzer.

- Here is the finished code

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

//
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random
function generate_random_input(type) {
    var randomInput;
    if (type === null) {
        return null;
    } else if (!type) {
        return false;
    }
    switch (type) {
        case "string":
            var chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuxyz0123456789";

            randomInput = '';
            for (var i = 0; i < Math.floor(Math.random() * 100000); i++) {
                var index = Math.floor(Math.random() * chars.length);
                randomInput += chars.charAt(index);
            }

            break;
        case "integer":
            randomInput = Math.floor(Math.random() *
Math.floor(Math.random() * 100000));

            break;
        case "float":
            randomInput = Math.random() * Math.floor(Math.random() *
100000);

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

var ipcJSActors = {
```

```
    "ClickHandler": {
        "Content:Click": null,
    },
    // seems to need some other kind of privilege "ContentSearch": "",
    //"FormValidation": {
    //    "FormValidation:ShowPopup": null
    //}, // 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": {
        // Maybe replace with the PDFJS:Save message (the below function
seems to only work by opening the GUI element)
        //"PDFJS:Parent:saveURL": {
        //    blobURL: "string",
        //    filename: "string"
        //}
    },
    //"Plugin": "RequestPlugins",
    "Prompt": {
        "Prompt:Open": {
            promptType: "string",
            title: "string",
            modalType: "integer",
            promptPrincipal: null,
            inPermutUnload: false,
            _remoteID: "string"
        },
    },
    // "ScreenshotsComponent": "",
    // "WebRTC": "",
    //"LoginManager": "",
    // "PictureInPicture": "", will work on this later though
    "PopupBlocking": {
        "UpdateBlockedPopups": {
            count: "integer",
            shouldNotify: false,
        },
        "UnblockPopup": {
            index: "integer",
        },
        "GetBlockedPopupList": null
    },
```

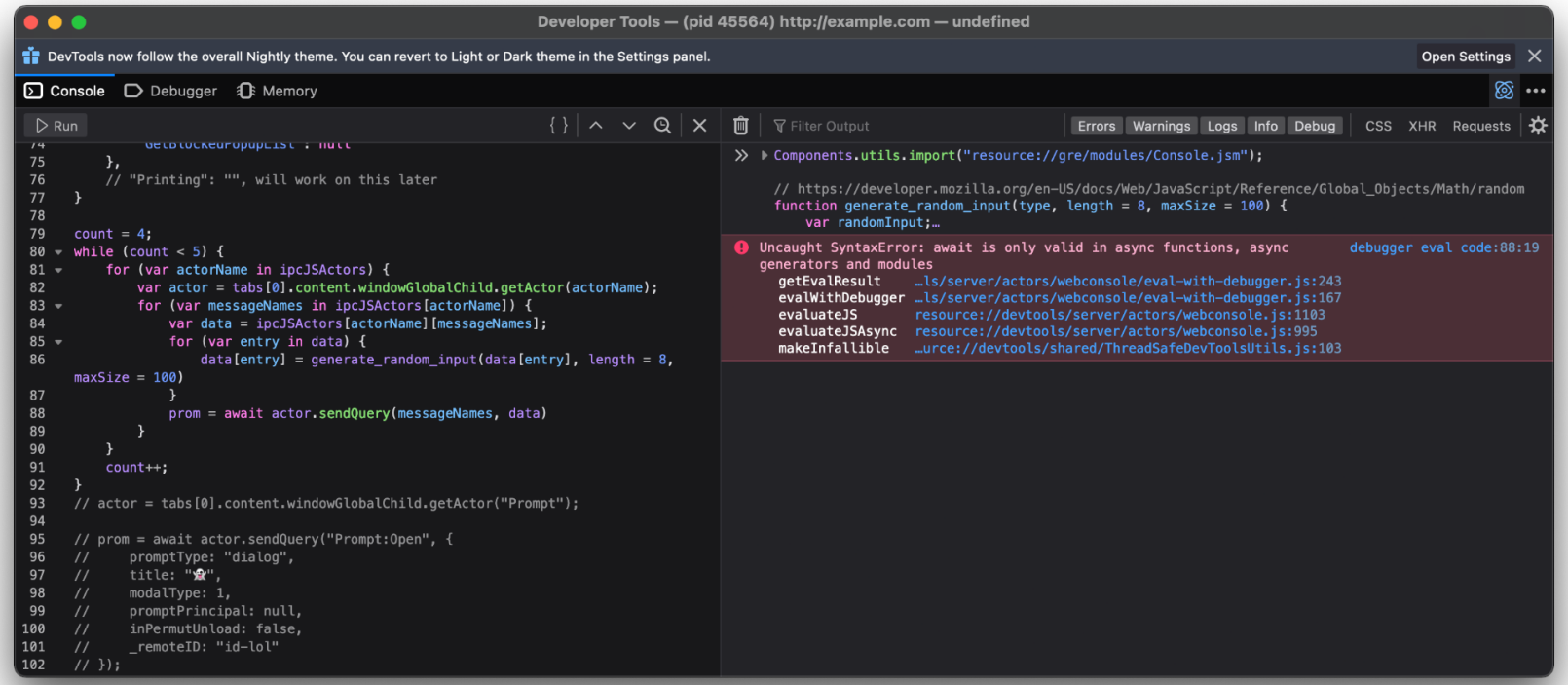
```
    // "Printing": "", will work on this later
}

var loop = async _ => {
    while (true) {
        var copy = JSON.parse(JSON.stringify(ipcJSActors))
        for (var actorName in copy) {
            var actor =
tabs[0].content.windowGlobalChild.getActor(actorName);
            for (var messageNames in copy[actorName]) {
                var data = ipcJSActors[actorName][messageNames];
                for (var entry in data) {
                    copy[actorName][messageNames][entry] =
generate_random_input(data[entry])
                }
                console.log("sending:" + copy[actorName][messageNames]);
                prom = await actor.sendQuery(messageNames,
copy[actorName][messageNames])
            }
        }
    }
}

await loop();
```

- As I was writing my fuzzer, there were multiple times where I would use console.log, which is just a way of printing output to the terminal, to ensure my program was working as intended.
- There were many issues that I faced along the way however.
- The first one was the way I wrote my code would overwrite the contents in the `ipcJSActors` dictionary. What that means is in the for loops where currently it says `copy[actorName][messageNames][entry]= generate_random_input(data[entry])`, instead of copy it had the original dictionary. This meant that on the next run (since it's an infinite loop), the dictionary would not contain the different data types my `generate_random_input` function requires (string, integer, and float).
    - To get around this, I had to create a brand new clone of the dictionary so that the original dictionary never changed during execution.
    - So I used a blog post by Samantha Ming called 3 Ways to Clone Objects. The solution I finally landed on, and that worked, was the following
    - `var copy = JSON.parse(JSON.stringify(ipcJSActors))`

- The second issue that I ran into was the fact that JSActors are asynchronous by default (at least `sendQuery` is).
    - Traditionally, on a computer, when code is executed it executes line by line. However, if a certain line of code or part of code takes a long time, it halts the execution of that program.
    - In order to speed up that process, we can send that one part of code to another process and have that execute in the background while the program continues executing. That's the idea behind asynchronous coding.
    - In this case, I was telling the code to wait for the actor to send the message and run that code with the `await` keyword in the following line:
    - `prom = await actor.sendQuery(messageNames, copy[actorName][messageNames])`
    - However, JavaScript returned with an error saying await was only available in async functions.



- To fix this, I found another blog post titled JavaScript Async and Await in Loops which used anonymous functions to create an async function
    - anonymous functions are just having functions stored in a variable rather than declaring an entirely new function.
    - In the code above I create an anonymous function with the following line of code:

        - `var loop = async _ => {`
    - Once I got the anonymous asynchronous function working, I faced another issue. It was still an anonymous function, I wasn't actually executing the function. So, I found another blog post titled JavaScript Anonymous Functions and that I could execute an anonymous function by just doing `variable();`
    - So to have my code run, I just had to do `loop();`
- Here is an output of one of the issues I ran into

```
Received Content:Click from the ClickHandler Actor via SendQuery

console.log: {"blobURL":"kgIsBfyVYR","filename":"IiY5tzDOLL"}

Received PDFJS:Parent:saveURL from the Pdfjs Actor via SendQuery

console.log: {"promptType":"4phUY87oMx","title":"MWQYQohAQu","modalType":6,"promptPrincipal":null,"inPermutUnload":false,"_remoteID":"CkKCCQei4J"}

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

console.log: {"count":5,"shouldNotify":false}

Received UpdateBlockedPopups from the PopupBlocking Actor via SendQuery

console.log: {"index":7}

Received UnblockPopup from the PopupBlocking Actor via SendQuery

console.log: null

Received GetBlockedPopupList from the PopupBlocking Actor via SendQuery

console.log: null

Received Content:Click from the ClickHandler Actor via SendQuery

console.log: {"blobURL":"Invalid type","filename":"Invalid type"}

Received PDFJS:Parent:saveURL from the Pdfjs Actor via SendQuery

console.log: {"promptType":"Invalid type","title":"Invalid type","modalType":"Invalid type","promptPrincipal":null,"inPermutUnload":false,"_remoteID":"Invalid type
"}

Received Prompt:Open from the Prompt Actor via SendQuery


Received EnterModalState from the BrowserElement Actor via SendAsyncMessage
```

- As you can tell, the input data is random at first, but then it says Invalid Type because my `generate_random_input` function outputs that if it gets a type that it doesn't understand.
- Here is what the code would output, minus the one `console.log` statement as that was added to the code later.

```
Received PDFJS:Parent:saveURL from the Pdfjs Actor via SendQuery

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

Received UpdateBlockedPopups from the PopupBlocking Actor via SendQuery

Received UnblockPopup from the PopupBlocking Actor via SendQuery

Received GetBlockedPopupList from the PopupBlocking Actor via SendQuery

Received Content:Click from the ClickHandler Actor via SendQuery

Received PDFJS:Parent:saveURL from the Pdfjs Actor via SendQuery

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

Received UpdateBlockedPopups from the PopupBlocking Actor via SendQuery

Received UnblockPopup from the PopupBlocking Actor via SendQuery
```

- The last thing I did was start to get some information about the data I was sending via `sendQuery` by editing the Firefox source code.
- In the JSActor.cpp file, where I added the other print code from the last journal, I also added this line:

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

- However, the above code had some issues, which when I tried to compile it, Firefox yelled at me for.

```
 0:15.00     Blocking waiting for file lock on package cache
 0:17.38 browser/app/firefox
 0:17.57 netwerk/test/http3server/force-cargo-program-build
 0:17.68     Blocking waiting for file lock on package cache
 0:21.03     Blocking waiting for file lock on package cache
 0:21.04     Blocking waiting for file lock on package cache
 0:21.23 In file included from Unified_cpp_dom_ipc_jsactor0.cpp:2:
 0:21.23 /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/dom/ipc/jsactor/JSActor.cpp:253:65:
 error: no member named 'DataLength' in 'mozilla::Maybe<mozilla::dom::ipc::StructuredCloneData>'; did you mean to use '->' instead
of '.'?
 0:21.23    std::cout << "\t Sent with the following data: \n\t" << *data.DataLength << "\n\n";
 0:21.23                                                          ^
 0:21.23                                                          ->
 0:21.23 /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/dom/ipc/jsactor/JSActor.cpp:253:66:
 error: reference to non-static member function must be called; did you mean to call it with no arguments?
 0:21.23    std::cout << "\t Sent with the following data: \n\t" << *data.DataLength << "\n\n";
 0:21.23                                                           ~~~~~^~~~~~~~~~
 0:21.23                                                                ()
 0:21.23 /Users/abhinavvemulapalli/Documents/coding/firefox_bug_hunting/browsers/firefox-source/dom/ipc/jsactor/JSActor.cpp:253:60:
 error: indirection requires pointer operand ('size_t' (aka 'unsigned long') invalid)
 0:21.23    std::cout << "\t Sent with the following data: \n\t" << *data.DataLength << "\n\n";
 0:21.23                                                         ^~~~~~~~~~~~~~~~
 0:25.71     Blocking waiting for file lock on package cache
 0:25.92     Finished release [optimized] target(s) in 8.33s
 0:25.97     Finished release [optimized] target(s) in 20.14s
 0:26.66 warning: using `procedural-masquerade` crate
 0:26.66   --> toolkit/components/cascade_bloom_filter/src/lib.rs:24:1
 0:26.66    |
 0:26.66 24 | / rental! {
 0:26.66 25 | |     mod rentals {
 0:26.66 26 | |         use super::Cascade;
 0:26.67 27 | |
 0:26.67 ...  |
 0:26.67 33 | |     }
 0:26.67 34 | | }
 0:26.67    | |_^
 0:26.67    |
 0:26.67    = note: `#[warn(proc_macro_back_compat)]` on by default
 0:26.67    = warning: this was previously accepted by the compiler but is being phased out; it will become a hard error in a futur
e release!
 0:26.67    = note: for more information, see issue #83125 <https://github.com/rust-lang/rust/issues/83125>
 0:26.67    = note: The `procedural-masquerade` crate has been unnecessary since Rust 1.30.0. Versions of this crate below 0.1.7 wi
```

- The first error there was caused by having a “*” in front of everything. From what I remember about C, the “*” operator usually refers to pointers, which is not applicable in this case, so I had to remove it (I believe it was present as an artifact of my prior testing the code).
- The second error was the fact that `DataLength` was a function and not a variable, so I had to add parentheses to call it as a function.
- The last error was an error specific to C++. Usually, calling a function of an object is done with a period, but for some reason C++ uses “->” to call a function inside of an object.

- Once I fixed those errors and re-compiled and ran my usual Prompt:Open test code I got the following output:

```
         Sent with the following data:
        1376


Received Browser:Thumbnail:CheckState from the Thumbnails Actor via SendQuery

         Sent with the following data:
        16


Received Browser:FirstPaint from the BrowserTab Actor via SendAsyncMessage

2022-04-15 14:44:25.884 firefox[49019:1838664] Warning: Expected min height of view: (<NSButton: 0x105f08400>) to be less than or e
qual to 30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-04-15 14:44:25.885 firefox[49019:1838664] Warning: Expected min height of view: (<NSButton: 0x122cbb000>) to be less than or e
qual to 30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-04-15 14:44:25.885 firefox[49019:1838664] Warning: Expected min height of view: (<NSButton: 0x122cb8800>) to be less than or e
qual to 30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-04-15 14:44:25.886 firefox[49019:1838664] Warning: Expected min height of view: (<NSButton: 0x122cbbc00>) to be less than or e
qual to 30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-04-15 14:44:25.887 firefox[49019:1838664] Warning: Expected min height of view: (<NSButton: 0x122cba400>) to be less than or e
qual to 30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-04-15 14:44:25.888 firefox[49019:1838664] Warning: Expected min height of view: (<NSPopoverTouchBarItemButton: 0x122cbc000>) t
o be less than or equal to 30 but got a height of 32.000000. This error will be logged once per view in violation.
2022-04-15 14:44:25.888 firefox[49019:1838664] Warning: Expected min height of view: (<NSPopoverTouchBarItemButton: 0x1393dec00>) t
o be less 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/bas
e-loader.js:289' would run

Received Prompt:Open from the Prompt Actor via SendQuery

         Sent with the following data:
        88


Received EnterModalState from the BrowserElement Actor via SendAsyncMessage


Received LeaveModalState from the BrowserElement Actor via SendAsyncMessage
```

- As you can see in the above photo, there is a line with “Sent with the following data” followed by the number 88 which is the size of the payload. I’m not sure what it’s measuring in and what other information I can get but it is a good first starting step.
- That’s about all I was able to get done, but it has set me up for success in the future when I continue to work on this project. It has also gotten me to the completion of my goal of writing a fuzzer.

Reflection on Goals and Timeline

Overall, I'm incredibly proud of how far I've come. I've learned so much about browser internals and how Firefox works and also how to read C++. I've hit my secondary goal of writing a fuzzer that could be used to find bugs. Now, it hasn't found any yet, however, if I keep working on it, there may be some potential for it to work in the future. There is a lot to be done for the fuzzer, but the project in general. Some of my future plans include trying to get a coverage-guided fuzzer working (see one of the previous journals for the definition of coverage-guided fuzzing). In addition, I want to explore more of the Firefox source code and see if I can find any lower level bugs in the IPC implementation. A potential future idea could include writing a lower level fuzzer or even writing a proof of concept exploit code for some of the newer sandbox escape bugs. With regards to my timeline, I believe it was fairly accurate all the way to the end. Some of the things on there were conditional and only hinged on me finding an actual vulnerability. My actual timeline for understanding Firefox and writing a fuzzer were fairly accurate.