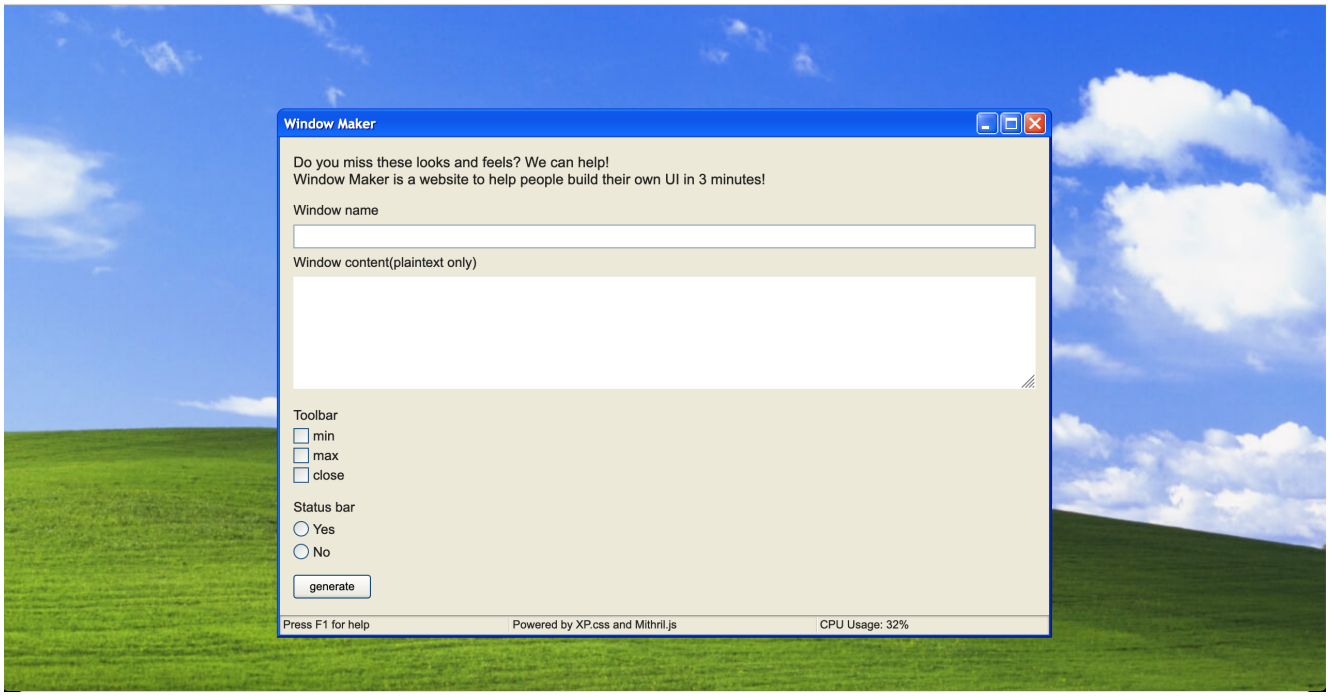
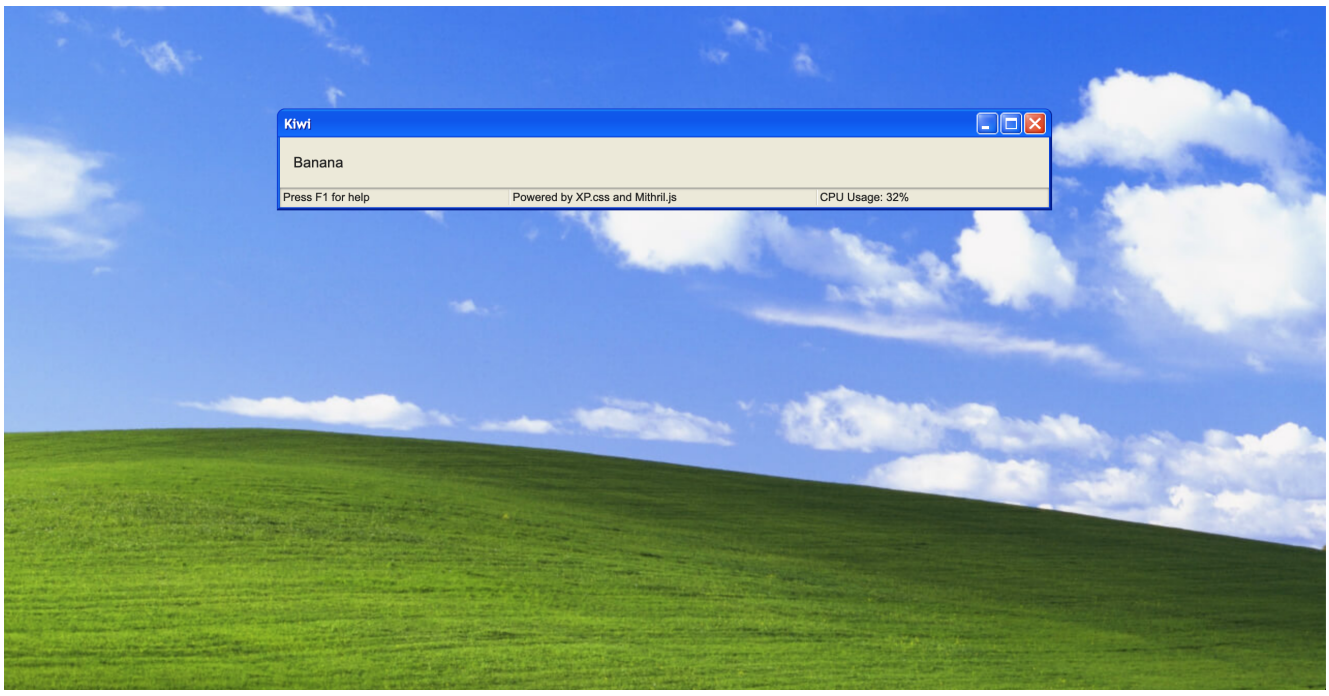
0422

Analysis



Nostalgic, isn't it? Well for a guy like me born in 2000 maybe this is less nostalgic than others, but that's another topic
So let's try to fill this form, submit it and see what's happening



Let's have a look around and see if we can find something interesting

```
function main() {
  const qs = m.parseQueryString(location.search)

  let appConfig = Object.create(null)
  appConfig["version"] = 1337
  appConfig["mode"] = "production"
  appConfig["window-name"] = "Window"
  appConfig["window-content"] = "default content"
  appConfig["window-toolbar"] = ["close"]
  appConfig["window-statusbar"] = false
  appConfig["customMode"] = false

  if (qs.config) {
    merge(appConfig, qs.config)
    appConfig["customMode"] = true
  }

  let devSettings = Object.create(null)
  devSettings["root"] = document.createElement('main')
  devSettings["isDebug"] = false
  devSettings["location"] = 'challenge-0422.intigriti.io'
  devSettings["isTestHostOrPort"] = false

  if (checkHost()) {
    devSettings["isTestHostOrPort"] = true
    merge(devSettings, qs.settings)
  }

  if (devSettings["isTestHostOrPort"] || devSettings["isDebug"]) {
    console.log('appConfig', appConfig)
    console.log('devSettings', devSettings)
  }

  if (!appConfig["customMode"]) {
    m.mount(devSettings.root, App)
  } else {
    m.mount(devSettings.root, {view: function() {
      return m(CustomizedApp, {
        name: appConfig["window-name"],
        content: appConfig["window-content"] ,
        options: appConfig["window-toolbar"],
        status: appConfig["window-statusbar"]
      })
    }})
  }

  document.body.appendChild(devSettings.root)
}
```

This is the main function of the website and it creates the entire page. Given a read at the code we understand that the location.search is parsed and used to create and object called qs

Then is created the object appConfig with Object.create(null) which creates an object but it provides null as prototype so it won't have any properties by default. After several properties are assigned to appConfig, the main function check if qs has the config properties and then it merges qs.config inside appConfig

Let's take a closer look to the merge function to see what happens

```
function merge(target, source) {
  let protectedKeys = ['__proto__', "mode", "version", "location", "src", "data", "m"]

  for(let key in source) {
    if (protectedKeys.includes(key)) continue

    if (isPrimitive(target[key])) {
      target[key] = sanitize(source[key])
    } else {
      merge(target[key], source[key])
    }
  }
}
```

Basically, it takes every key in the source object and checks if it's in a list of forbidden keys, and if that is true it don't merge them.
Otherwise it checks if the value for a given key is a primitive type (boolean, number, string, and even null or undefined) and if it is, the value is sanitized and then merged
If the value is not primitive then the merge function is called recursively with the new values for that given key

```
function sanitize(data) {
  if (typeof data !== 'string') return data
  return data.replace(/[<>%&\$\s\\]/g, '_').replace(/script/gi, '_')
}
```

The sanitize function is very simple. It checks if the given data is a string, and if it is, the function replace all the character that matches the regular expression along with every occurence of "script"

Let's turn back to the main function where another object, called devSettings is created with the same method used for appConfig, and after assigning some properties there is an if clause which has a call to a function as condition

```
function checkHost() {
  const temp = location.host.split(':')
  const hostname = temp[0]
  const port = Number(temp[1]) || 443
  return hostname === 'localhost' || port === 8080
}
```

Essentially the function split the host using the colon as delimiter and check if the hostname is equals to localhost or the port is 8080, which is weird since neither of them is true, and so the function will return false and the if clause is skipped

Next everything is mounted inside devSettings.root and is appended in the body. Something appended in the body seems like an injection point, so we can try to figure out how to add something to devSettings.root. Since an object is created with our input we can try something cool tricks. Maybe you've heard about prototype pollution?

A step back and five forward

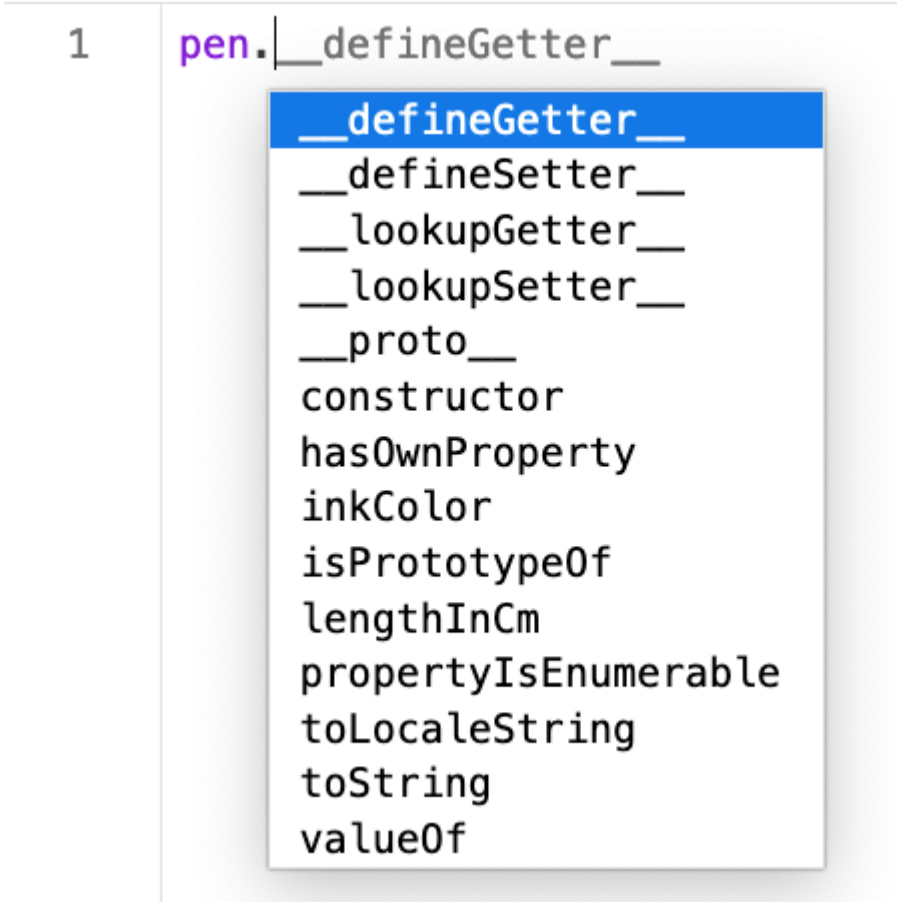
Before we understand what is prototype pollution, i think that it's better to understand how object works in JavaScript
An object is a collection of pairs that contain keys and values, and the can be declared in this way

```
var pen = { inkColor : "blue", lengthInCm : "10" }
```

We can access the values of the object in two different ways: with the dot (.) notation or with the square brackets
Let's try this in the console to have a better understand on what is happening

```
» console.log(pen.inkColor)
  console.log(pen["lengthInCm"])
  blue
  10
```

But wait, when we use the dot notation in the console, the autosuggest shows us some properties that we haven't definied



Where these comes from if we haven't defined?
Well here is where prototypes comes in the game, which are the mechanism used by objects to inherit features.
This means that objects can act as a template for other inheriting objects. This action is called prototype chain

Everytime we create an object it inherits features from a prototype called Object.prototype and the same things can be said for strings which will inherit from Strings.prototype and for arrays as well

To escalate from one object to his prototype we can use the `__proto__` as well as `constructor.prototype` , and there's a little difference since the first is the actual object used in the chain and the second shows the prototypes of the object it is used on

Here is where prototype pollution makes is appereance. Basically if an attacker can manipulate `__proto__` or `constructor.prototype` can change the application behaviour.
Since every object inherits the prototypes from their prototype, attackers could take advantage of that to pollute Object.prototype which will cause each object to be polluted

For example, we notice that every object has a toString() method. Now, what if an object uses this method and we can pollute it to change is functionality?
Let' go again in the console to demonstrate it

```
» var pen = { inkColor : "blue", lengthInCm : "10" };
  console.log("This is the normal toString: "+pen.toString());
  var evilObject = {};
  evilObject.__proto__.toString = function(){
    return "Prototype Pollution is fantastic"
  }
  console.log("This is not the normal toString: "+pen.toString());
  This is the normal toString: [object Object]        debugger eval code:2:9
  This is not the normal toString: Prototype          debugger eval code:7:9
  Pollution is fantastic
```

See? We have redefinied the toString() method with something else, and is behaviour is changed in every object
Cool isn't it?

Back on track

We have said that the goal is to inject something inside devSettings.root, but devSettings is created with null as prototype, so it won't inherit nothing at all. Given another read at the code, we realize that devSettings is merged with qs.settings, but there's a big problem

```
if (checkHost()) {
  devSettings["isTestHostOrPort"] = true
  merge(devSettings, qs.settings)
}
```

The merge happens inside the if clause above, and we know that the checkHost() function always return false, or at least in normal cases

Bypass all the things

The first thing that came to my mind was to inject a new behaviour inside the split function, but after a couple of tries i've realized that everything is treated a string and so the redefinition of the function could not work

After at lot of thinking, trying and error, and bad words i've realized that maybe, since the function uses array, we can change some behaviour in Array.prototype. First of all we need an array, and given another look at the url i've found it

`?config[window-name]=Kiwi&config[window-content]=Banana&config[window-toolbar][0]=min&config[window-toolbar][1]=max&config[window-toolbar][2]=close&config[window-statusbar]=true`

The config[window-toolbar] uses numerical index so it could be an array. This is confirmed due to the fact that in the source code appConfig[window-toolbar] is declared as an array

```
let appConfig = Object.create(null)
appConfig["version"] = 1337
appConfig["mode"] = "production"
appConfig["window-name"] = "Window"
appConfig["window-content"] = "default content"
appConfig["window-toolbar"] = ["close"]
appConfig["window-statusbar"] = false
appConfig["customMode"] = false
```

We want to use config since is merged before devSettings and in normal cases devSettings won't be merged at all

Now that we have an array, we could reach Array.prototype by using `config[window-toolbar][__proto__]` but the sanitize function has blacklisted `__proto__` so we must use the following code

`config[window-toolbar][constructor][prototype]`

Which is equals to Array.prototype . Now is where the tricky part begins, since the array used inside the checkHost() function doesn't using anything peculiar at first i wasn't able to understand what do i need to change to bypass the function

Then, like a solar beam, a question came to me

> "Is there any way of changing globally the value of a certain index in every single array that's in the code?"

And yes, there's a way. If we add at Array.prototype a value for a certain index every array in the code will have the same value if not overwrited

```
function checkHost() {
  const temp = location.host.split(':')
  const hostname = temp[0]
  const port = Number(temp[1]) || 443
  return hostname === 'localhost' || port === 8080
}
```

Since in the checkHost() function hostname gets overwrited, we cannot pollute the value for the index 0.
But we can for the index 1 since by default port is not explicitly delcared inside the url.
In the end the first part of our payload will be the following

`?config[window-toolbar][constructor][prototype][1]=8080`

Trick the sanitize

Now, we have bypassed the checkHost(), it's possible to merge devSettings with qs.settings
In particular since only devSettings.root is appended in the page, we want to add something malicious to it to reach the XSS

The first thing to do is to find inside devSettings.root a property that inject the code directly inside HTML.
After a couple of research is possible to find an innerHTML in this series of properties

`devSettings.root.ownerDocument.body.innerHTML`

Knowing this, we can leverage the merge function to insert our malicious code, but
as mentioned before the merge has a call to a sanitize function

```
function sanitize(data) {
  if (typeof data !== 'string') return data
  return data.replace(/[<>%&\$\s\\]/g, '_').replace(/script/gi, '_')
}
```

Here the only thing that we can do to bypass it, is to trick the function to think that our malicious code isn't a string but an object

After some tries it's possible understand that if we add and index at the end of the series of properties written above, the typeof check will result in an object (which contains a string, but still an object)

The tricky part here is the fact that an object in the merge function will be merged recuresively only unite together primitive values.
But in this case for that given key, which is innerHTML, in the target is a string while in the source is an object, so everything works fine

In conclusion

We have everything we need to inject a malicious html tag, but there are two last thing to point out

First of all there is CSP header in the wep page

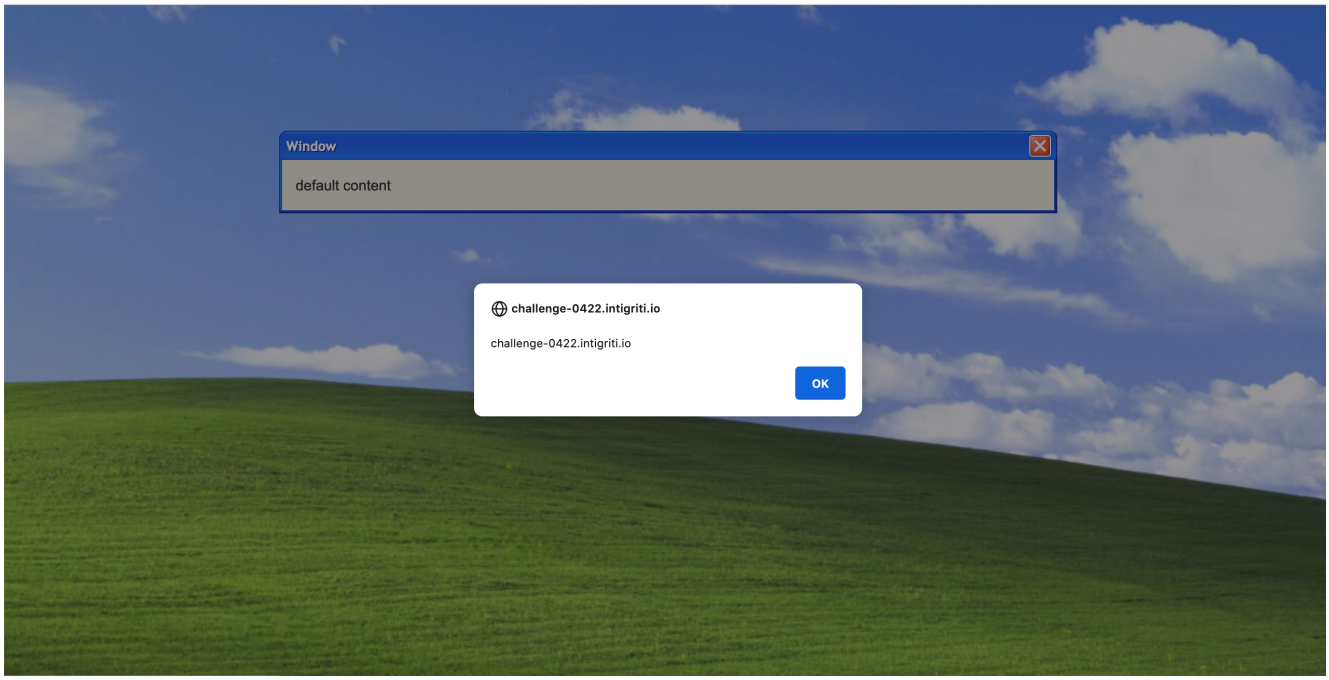
```
<meta http-equiv="Content-Security-Policy" content="default-src 'none';
style-src 'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self';
script-src 'self' 'unsafe-inline';">
```

This header allow us to use inline script and event handler

The second thing is that we cannout use directly the equal sign (=) inside the url beacuse the framework used to create the page (mithril.js) uses it as a delimiter to split the url
We can bypass this thing with a simple URL Encode

So, in the end our paylod will be the following

```
?config[window-toolbar][constructor][prototype][1]=8080&settings[root]
[ownerDocument][body][innerHTML][0]=
<img/src/onerror%3dalert(document.domain)>
```



Credits

Thanks to @aszx87410 for creating this awesome and super fun challenge, and also very big thanks to @TheRealBrenu for sharing thoughts and doing the challenge with me