4 DNS rebinding in --inspect (insufficient fix of CVE-2018-7160)

Share:

TIMELINE

v6ak submitted a report to Node.js. Dec 31st (2 years ago)

Summary: While the debugger (i.e., the --inspect option) tries to prevent DNS rebinding, the whitelist is excessive.

Description: The whitelist includes “localhost6”, which is not that widespread. When “localhost6” is not present in /etc/hosts, it is just an ordinary domain that is resolved via DNS, i.e., over network. If the attacker controls victim's DNS server or can spoof its responses, the DNS rebinding protection can be bypassed by using the “localhost6” domain. As long as the attacker uses the “localhost6” domain, they can still apply the attack described in CVE-2018-7160.

Reasoning why localhost6 is not so common and Node.js should not rely on its presence in the hosts file:

- It is not even present in the node:latest Docker image (sha256:aa1930b56896a43dedb227526d5d40f4a6e9157f9d8703f9584650cde510438a)
- I haven't seen it in Windows 10.
- Unlike RFC 6761 for localhost, I have found no RFC that mentions localhost6 (see https://www.google.com/search?q=localhost6+site%3Atools.ietf.org).

Steps To Reproduce:

Preconditions: Victim has no entry for localhost6 in hosts and attacker controls DNS responses. (It does not matter if the attacker control the DNS server or the network communication between the DNS server and the victim.)

1. Victim runs node with --inspect option
2. Victim visits attacker's webpage
3. The attacker's webpage opens http://localhost6:9229
4. Victim finds no “localhost6” entry in hosts file, so it asks the DNS server and gets <attacker's-IP>. (Maybe the response will have a short TTL. There are multiple tricks to make DNS rebinding successful in a short time, but I am not going to be exhaustive.)
5. Victim loads webpage http://localhost6:9229 from <attacker's-IP>.
6. The webpage http://localhost6:9229 tries to load http://localhost6:9229/json from attacker's server. (If the IP address of “localhost6” is still cached, attacker needs to retry. There are techniques that can speed it up, like using RST packet.)
7. Due to a short TTL, the DNS server will be soon asked again about an entry for “localhost6”. This time, the DNS server responds “127.0.0.1”.
8. The http://localhost6:9229 website (i.e., the one hosted on <attacker's IP>) will retrieve http://localhost6:9229/json from 127.0.0.1, including webSocketDebuggerUrl.
9. Now, the attacker knows the webSocketDebuggerUrl and can connect to is using WebSocket. Note that WebSocket is not restricted by same-origin-policy. By doing so, they can gain the privileges of the Node.js instance.

Vulnerable code: https://github.com/nodejs/node/blob/fdf0a84e826d3a9ec0ce6f5a3f5adc967fe99408/src/inspector_socket.cc#L584

Impact:

Attacker can gain access to the Node.js debugger, which can result in remote code execution.

Supporting Material/References:

- Original vulnerability: https://nvd.nist.gov/vuln/detail/CVE-2018-7160
- Vulnerable code: https://github.com/nodejs/node/blob/fdf0a84e826d3a9ec0ce6f5a3f5adc967fe99408/src/inspector_socket.cc#L584
- Documentation that mentions the vulnerable behavior: https://nodejs.org/en/docs/guides/debugging-getting-started/

Impact

Attacker can gain access to the Node.js debugger, which can result in remote code execution.

mcollina Node.js staff posted a comment. Jan 1st (2 years ago)

Thanks for the report. We'll analyze this report as soon as possible as most of the team is taking some vacation time right now.

One quick question: as far as I understand DNS, for this to work it would require access to the same network or be anyway be able to control a DNS server that the developer machine is already using. Can you include some step-by-step documentation that would allow us to reproduce this issue? I think most of us are not familiar on how to make a DNS rebind for localhost6.

v6ak posted a comment. Jan 1st (2 years ago)

On DNS access: You are right. This is doable for local network adversary that can intercept all your traffic or for your DNS provider.

On reproduction: There are multiple approachs:

A. Simulation: This approach does not rely much on environment. It skips DNS server, web browser and /etc/hosts, so it does not actually perform the DNS rebinding, it just simulates its outcome from Node.js perspective: A client from trusted network (i.e., loopback in this case) tries to access a security-sensitive resource, but the Host HTTP header actually corresponds with an attacker's domain (i.e., “localhost6” in this case). It can be easily done with curl oneliner:

curl 127.0.0.1:9229/json -H "Host: localhost6"

Advantages: Easy to do, does not depend much on environment.

Disadvantages: The simulation does not explain the principle of the attack. It also does not test doability under a specific environment.

B. Basic attack scenario: Just getting http://127.0.0.1:9229/json by an actual DNS rebinding attack. I can dockerize some DNS rebinding script that demonstrates actual attack. Maybe I cannot dockerize DNS spoofing, but if you voluntarily use the malicious DNS server, you can reproduce the attack.

C. Doing complete attack scenario. This requires to perform B first. Then, you can connect to the webSocketDebuggerUrl and debug the Node.js application. With enough understanding of the protocol (or with a WebSocket reverse proxy and manual work), the attacker can obtain debug console and execute some demo command, e.g., child_process.exec(“xterm") or child_process.exec("calc.exe"). For doing so, it seems that we need to send something like this:
{"id": `<sequential id>` ,"method":"Runtime.evaluate","params":

Maybe we will need to do perform some handshake first or create some context, not sure. I haven't tried that.

Should I proceed with doing B or even C?

mcollina Node.js staff posted a comment. Jan 2nd (2 years ago)

I think that's enough details. We'll confirm this asap.

mcollina Node.js staff posted a comment. Jan 2nd (2 years ago)

@chalker @mylesborins you were involved in the original discussion in https://github.com/nodejs-private/security/issues/181 and https://github.com/nodejs-private/node-private/pull/102.

I've found this text in https://github.com/nodejs-private/node-private/pull/102#issuecomment-371453420:

> Domains localhost, localhost6 are fine. DNS rebinding not possible.

@chalker Can you confirm this? Even if `localhost6` is not defined?

> Safeguard to make sure when they resolve to a loopback ipv4/ipv6 address seems reasonable, even a simple check for 127.0.0.1 / ::1 / :ffff:127.0.0.1 should be ok, I guess.

This was present in https://github.com/nodejs-private/node-private/pull/102/commits/becd88ad2686e020d2f40dcd1d2bfdd51d215662, but then it was removed from the final commit.

Should we add it back?

v6ak posted a comment. Jan 2nd (2 years ago)

While verifying that localhost/localhost6 corresponds to the right IP address might look like a good idea, I find this mechanism quite fragile. There are two requirements:

1. It needs to be permanent.
2. You need to check the point of view of all browsers rather than Node.js's PoV.

In real world, there are multiple issues (both theoretical and practical) with ensuring any of those requirement:

- Obviously, you shouldn't ask DNS, as it would break (1) – doing so would make the attack still possible, just a little harder. Maybe you would need to just check the hosts file.
- Format ambiguities (rather theoretical) could break (2) – when having multiple parsers of hosts file, there can be some edge cases that can lead to Node.js thinking that there is a proper record for localhost6, while some browser would ignore that.
- Updates (rather theoretical) – a system update can break (1) by removing localhost6 from
- Sandboxing can break (2). Docker images can have a different /etc/hosts, so it can break. And while Flatpak/Snap look safe, there are potential more sandboxing mechanisms that I haven't explored or that will be created in future.
- Trusted local subnetwork can break (2). (The trusted subnetwork might be virtual, i.e., inside a single virtual computer, which is a special case of the previous point.) When you allow other (maybe virtual) computers within this network to access the host, they might be also abused for this attack, as they might have a different hosts file.

In my opinion, adding further measures that would allow keeping "localhost6" entry in the whitelist is hard (or even impossible) to do it right. On the other side, I see little benefit (or maybe no benefit at all) of keeping "localhost6" in the whitelist.

mcollina Node.js staff changed the status to Triaged. Jan 13th (2 years ago)

> In my opinion, adding further measures that would allow keeping "localhost6" entry in the whitelist is hard (or even impossible) to do it right. On the other side, I see little benefit (or maybe no benefit at all) of keeping "localhost6" in the whitelist.

Thanks! I think it's pretty clear what will should do. We'll try to include this in the next security release.

danbev updated CVE reference to CVE-2021-22884. Feb 18th (2 years ago)

danbev closed the report and changed the status to Resolved. Feb 23rd (2 years ago)

@v6ak Thank you for the report! This has now been included in this security release.

danbev requested to disclose this report. Feb 23rd (2 years ago)

v6ak agreed to disclose this report. Feb 23rd (2 years ago)

This report has been disclosed. Feb 23rd (2 years ago)

v6ak posted a comment. Feb 24th (2 years ago)

I have originally just skimmed the text, so I had missed few things in the release notes:

- The attack is described as "denial of service". Why?
- The description also contains "when the whitelist includes \"localhost6\"". This is technically true, but the problem is in the hardcoded whitelist. So, this condition is always true on the vulnerable versions.

danbev posted a comment. Feb 24th (2 years ago)

> `The attack is described as "denial of service". Why?`

Sorry about that, that was a mistake on my part when creating the blog. I've created a pull request to fix this and it should be updated soon.

The Internet Bug Bounty rewarded v6ak with a $500 bounty. Feb 25th (2 years ago)