



Vulnerability Research
Report: TinyCheck

Table of Contents

Table of Contents	1
Sayfer	2
Executive Summary	3
Product Details	4
Name	4
Version	4
Summary	4
Source Code	4
Vulnerability #1 - Use of Hard-coded Credentials	5
Definition	5
Description	5
Mitigation	6
Vulnerability #2 - Command Injection	7
Definition	7
Description	7
Mitigation	7
Vulnerability #3 - Server Side Request Forgery (SSRF)	8
Definition	8
Description	8
Mitigation	9
Exploit - Chaining the Vulnerabilities	10
Appendix A - Command Injections	11
Appendix B - More findings	12

Sayfer

Here at Sayfer, we specialize in vulnerability research and malware analysis in desktop, mobile, web, IoT, and hardware fields.

Over time we formed in-house tooling to identify and hunt cyber threats, these tools provide us with a thorough understanding of malware behavior, and allow us to perform better static and dynamic analysis in different testing environments.

Executive Summary

We found 3 different vulnerabilities in the TinyCheck¹ product. Each vulnerability has a high severity by itself, but once combined into a chain, a remote attacker could exploit them to get a RCE (remote code execution) on the remote server.

We added as much technical information for every vulnerability, explained why it is exploitable and proposed a mitigation strategy.

In a nutshell, we used the default credentials of TinyCheck to get a token which allowed us to edit a configuration file. We edited 2 sections in the configuration file:

1. A section that allows us to use malicious code that will be executed later
2. A URL that will be called, and trigger the malicious code from the first section

We wanted to make sure that everything is clear to your team, so we created a PoC script that will be attached to this document.

[Appendix B](#) describes more findings that might be relevant but we weren't using them in our chain of vulnerabilities.

¹ <https://github.com/KasperskyLab/TinyCheck/>

Product Details

Name

TinyCheck

Version

1.0 (sourced by the install script, there is no public version)²

Tested on commit: 9d13030aba08cc4075b44c6cce75610e2d1391b4

Summary

TinyCheck allows you to easily capture network communications from a smartphone or any device, which can be associated to a Wi-Fi access point, in order to quickly analyze them.

Source Code

Available on GitHub under the organization “KasperskyLab” and the repository “TinyCheck”:
<https://github.com/KasperskyLab/TinyCheck/>

²

<https://github.com/KasperskyLab/TinyCheck/blob/9d13030aba08cc4075b44c6cce75610e2d1391b4/install.sh#L256>

Mitigation

Prompt the user for their custom strong credentials during the installation phase or before the first use.

Vulnerability #2 - Command Injection

Definition

The software constructs all or part of a command by using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralize special elements that could modify the intended command. For more information please read about CWE-77⁴

Description

The application code in the “frontend” service is vulnerable to command injections in several areas, the full list is in [appendix A](#).

For instance, by looking at the function `start_capture`⁵ we could see that the code concatenates a string variable to the shell command and uses `shell=True`⁶ rather than a list of arguments.⁷

```
try:
    sp.Popen("tshark -i {} -w {} -f \"tcp or udp\" " format(self.iface self.pcap), shell=True)
    return {"status": True,
            "message": "Capture started"}
```

If a malicious actor could control the `iface` variable, he could run arbitrary code on the server. A value with a payload that exploits this vulnerability would be:

```
`whoami>/tmp/exploit.out`
```

Since `start_capture` is called when the user captures network activities, the next time the user will do so, the malicious payload will be executed on the remote machine.

Mitigation

Convert the command into a list to better serialize each argument and prevent injection:

```
sp.Popen(["tshark", "-i", self.iface, "-w", self.pcap, "-f", "\"tcp or udp\""]) # pseudo code, untested
```

⁴ <https://cwe.mitre.org/data/definitions/77.html>

⁵ <https://github.com/KasperskyLab/TinyCheck/blob/9d13030aba08cc4075b44c6cce75610e2d1391b4/server/frontend/app/classes/capture.py#L22-L55>

⁶ https://security.openstack.org/guidelines/dg_avoid-shell-true.html

⁷ https://security.openstack.org/guidelines/dg_use-subprocess-securely.html

Vulnerability #3 - Server Side Request Forgery (SSRF)

Definition

By providing URLs to unexpected hosts or ports, attackers can make it appear to look as the server is sending the request, while possibly bypassing access controls such as firewalls that prevent the attackers from accessing the URLs directly. For more information please read about CWE-918⁸

Description

The “watchers” service reads the configuration file under the path of “watchers/iocs”. Which is a list of URLs that later the service uses to make an HTTP GET request.

A malicious actor, that could control that list, can create an HTTP request **from** the server, while bypassing firewalls or reaching local servers, specifically, the “frontend” server, that is configured by default to run on localhost and is not open to the world.

Since this list is available in the configuration file which can be changed remotely by using `api/config/edit` endpoint, this payload will trigger request the local “frontend” service:

```
https://raspberrypi/api/config/edit/watchers/iocs/http://localhost/api/network/ap/start|DummyPlaceHolder
```

After this request was made, the configuration file looks like this:

```
watchers:
  iocs:
    - http://localhost/api/network/ap/start
    - DummyPlaceHolder
  whitelists:
```

⁸ <https://cwe.mitre.org/data/definitions/918.html>

Once the watchers will be restarted the new IOC - “http://localhost/api/network/ap/start” will be called via HTTP get request, an endpoint the malicious actor didn’t have access to before.

Mitigation

To prevent SSRF there is a need to fully understand the architecture deeply, we can not give a one-size-fits-all solution, but we highly recommend reading the “Server-Side Request Forgery Prevention Cheat Sheet by OWASP” - Available Protections section⁹

Exploit - Chaining the Vulnerabilities

There are different ways to get RCE(remote code execution) on the host machine, the way we choose to chain all of these vulnerabilities is:

1. Grab the JWT token by using the default credentials (vulnerability #1)
2. Edit the configuration file and add command injection code to the network/in section, that code will be executed later (vulnerability #2)
3. Edit the configuration file and add our custom IOC URL to the IOCs list, with the local URL “frontend” server at `http://localhost/api/capture/start|DummyPlaceholder`

Once the user will capture network traffic or the “watchers” services will be restarted, our custom command injection will run on the remote machine.

Appendix A - Command Injections

Below is a list of all the sourcecode we could find on the application that is vulnerable to command injection:

1. analysis/classes/suricataengine.py:32
2. analysis/classes/zeekengine.py:352
3. analysis/classes/zeekengine.py:354
4. analysis/classes/zeekengine.py:356
5. server/frontend/app/classes/analysis.py:27
6. server/frontend/app/classes/capture.py:49
7. server/frontend/app/classes/network.py:145
8. server/frontend/app/classes/network.py:238
9. server/frontend/app/classes/network.py:240
10. server/frontend/app/classes/network.py:297
11. server/frontend/app/classes/network.py:308
12. server/frontend/app/classes/network.py:316

The application also has safe OS command calls, but these are still being called with `shell=True` and might be vulnerable in the future:

1. server/frontend/app/blueprints/misc.py:16
2. server/frontend/app/classes/network.py:277
3. server/frontend/app/classes/network.py:279
4. server/frontend/app/classes/network.py:280
5. server/frontend/app/classes/network.py:281
6. server/frontend/app/classes/network.py:295

Appendix B - More findings

While conducting the research we found a few findings that we believe can be improved:

1. The application runs as the root user, any exploit for the web application will give high privilege access to the malicious actor.
2. The application runs in debug mode, which isn't suitable for a production environment.
3. The endpoint `/api/config/list` lists all the configuration data and is not protected by any authentication system, while it is not showing the password field, in the future, more configurations might be added and will be automatically shown in this endpoint.



Please do contact us back regarding the next steps, we are available at security@sayfer.io

If you want to encrypt your message please use our public PGP key:

<https://sayfer.io/3D4A560F984C41BE.asc>

Key ID: 3D4A560F984C41BE

Website: <https://sayfer.io>