JSON Web Tokens and Bash Injection

Network Security HS16 - Group 6
Lukas Bischofberger, Silvan Egli, Jonas Passerini, Lukas Widmer

November 14, 2016

Abstract

A chat services suffers from two vulnerabilities, which allow an arbitrary user to gain administrative access rights and to extract the secret key of the web application by performing a bash injection. The first part of the challenge is a JSON Web Token (JWT) vulnerability where the client is able to select a trivial 'null' signing algorithm. Using a fake authentication token, it is possible to circumvent the authentication mechanism and send chat messages as an administrator. This enables a normal user to send privileged admin commands. The second part of the challenge exploits a bash injection, which is possible due to a wrongly configured subprocess call with unsanitized user input and therefore allows arbitrary code execution. A malicious admin command can be crafted to extract the secret key from the settings file.

1 Challenge Description

1.1 Type of Challenge

The first part of the challenge requires to analyze and craft authentication tokens, which can be done offline. To test the crafted tokens, the web application needs to be online.

For the second part of the challenge, the web application needs to be running in order to extract the secret key, and is therefore an online attack.

1.2 Category

The challenge belongs to the category Web Security and also requires some basic Linux and Networking understanding.

1.3 Mission

A simple chat service is provided by a company in the form of a web application. There is only one chat room, and all authenticated users might read messages and send new ones to this chat room. There is an additional administrative role, which allows users belonging to this role to invoke several admin commands, e.g. to change the background color of the chat window. Admin commands sent by basic users have no effect.

The secret key used by the chat service is highly sensitive, since it is also used by other services of the company. The goal of the challenge is to expose the secret key. To achieve the goal, two vulnerabilities need to be combined in a clever way. In a first step, an attacker has access to a basic user account and the goal is to gain administrative user rights by circumventing the token based authentication mechanism. In a second step, the attacker uses the administrative rights gained in the previous step to craft a malicious admin command, which exposes the secret key.

1.4 Learning Goal

The first part of the challenge teaches the student about how to collect and analyze token based authentication mechanisms, in particular the JSON Web Token library. The vulnerability should increase the awareness of the problem, if the client is allowed to choose the signing algorithm.

The second part of the challenge demonstrates the danger of executing unsanitized input from an untrusted source and how to exploit such a vulnerability with a limited interface.

2 Implementation

2.1 Requirements

The web application is implemented in Python using the web framework Django and SQLite is used as the database backend. Token authentication is provided by the JSON Web Token library, which is either outdated or patched to enable the first vulnerability. The frontend of the chat application is provided by Angular2. The admin commands are parsed by the web framework and then executed with a bash command in a subprocess, without sanitizing the input first. The bash commands have direct access to the local file system and allow an attacker to perform a bash injection.

2.2 Deployment

The web application is deployed behind a web server and listens on port 80.

3 Solution

3.1 Hints

1. Analyze the tokens used to authenticate chat messages. Can you decode the token to reveal further information about the library used to verify the tokens?

2. The admin commands do not require any database access. Try to find out what happens, e.g. if an administrator changes the background color of the chat window.

3. The secret key of a Django applications is usually stored in a file called settings.py

3.2 Step-by-Step Instructions

1. Exploit the JWT web token vulnerability to bypass authentication

   (a) Collect and analyze the authentication tokens used by the application to authenticate chat messages. To do so, you can use Wireshark or the Firefox Add-on Tamper Data.

   (b) Decode the base64 encoded tokens and find out that the chat application utilizes an outdated JSON Web Token library.

   (c) A short Google search should reveal several open vulnerabilities, such as the "null Algorithm" described on the auth0.com blog [1].

   (d) Learn about the algorithm and create your own token, which authenticates you as the administrator. To find the ID of an administrator you might need to look at the source of the chat history.

   (e) You should be able to send chat messages as an administrator and invoke admin commands.

2. Perform a bash injection by crafting a malicious admin command

[1] `https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/`

(a) Try to find out how admin commands work. Observe the CSS style sheet and observe that the color values are directly written into the CSS file.

(b) Find a bash command to read and print the contents of a file, which could be used as a color value. e.g. cat settings.py

(c) Embed the bash command in an admin command, such that instead of the color value the output of your bash command is written into the CSS file. Note that in bash injections, it is often not possible to inject commands with white space characters and you might need to modify your bash command accordingly. An example for an admin command could be: \\background $(cat,settings.py)

(d) Take a look again at the CSS file, which should now contain the secret key. It is also possible to perform different attacks, e.g. to open a reverse shell and then manually searching the secret key.

4 Mitigation

The JWT vulnerability can be solved by disallowing clients to choose an algorithm, especially if it is a 'null' algorithm. Additionally, all dependencies should be kept up to date to prevent other vulnerabilities.

The bash injection vulnerability can be prevented by properly sanitizing the user input and by disabling the shell access. A nicer solution would be to prevent the use of subprocesses completely and instead manage the color configuration within Django, e.g. by writing the values to the database and dynamically parse the website using a templating system.

The attacker had full access to the system by executing arbitrary commands or opening a reverse shell, therefore the whole system is compromised and all keys should be renewed, including all the systems which used the same secret key.

5 Conclusion

The presented challenge consists of two vulnerabilities and demonstrates how they can be combined to extract the secret key of a web application. The student learns how the JWT library works and how the token authentication can be bypassed, if the client is able to select a weak or even trivial signing algorithm. The student also learns how to perform a bash injection by exploiting a subprocess call suffering from unsanitized user input. Both

vulnerabilities reflect common problems, which could mostly be tackled in a simple way.