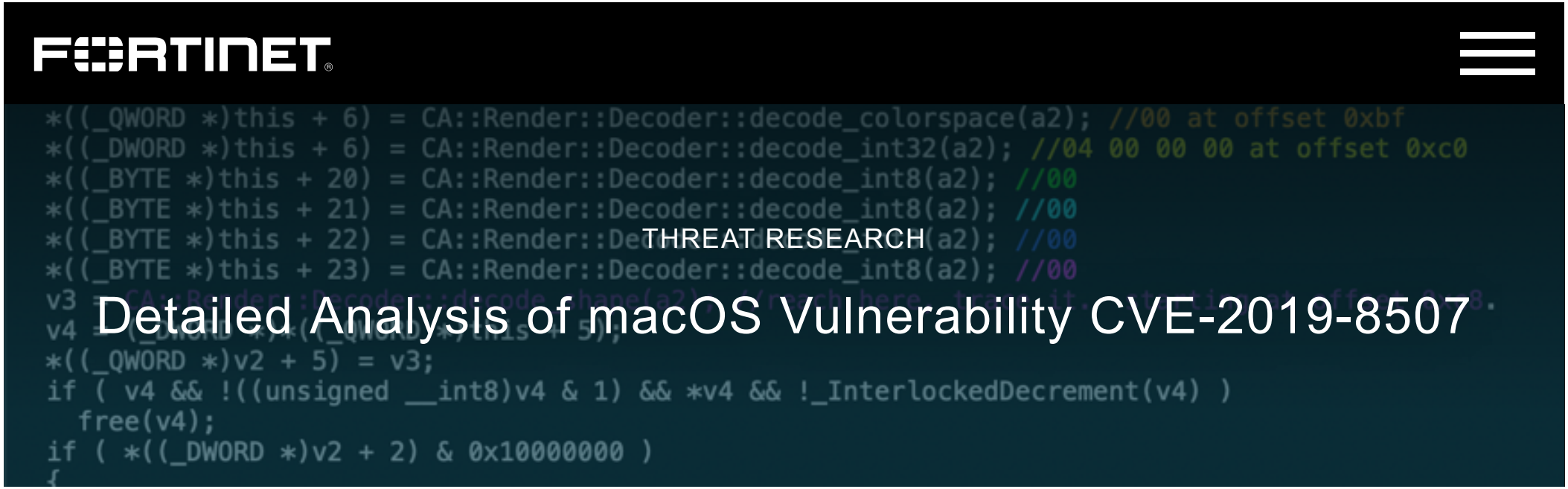


THREAT RESEARCH

Detailed Analysis of macOS Vulnerability CVE-2019-8507

By Kai Lu | April 23, 2019

FortiGuard Labs Threat Analysis Report on an Memory Corruption Vulnerability in QuartzCore while Handling Shape Object.

On March 25, 2019, Apple released macOS Mojave 10.14.4 and iOS 12.2. These two updates fixed a number of security vulnerabilities, including CVE-2019-8507 in QuartzCore (aka CoreAnimation), which I reported to Apple on January 3, 2019 using our FortiGuard Labs responsible disclosure process, read more. For more details on the Apple updates, please refer to https://support.apple.com/en-us/HT209600. In this blog I will provide a detailed analysis of this issue on macOS. Some of the analysis techniques used can be found in my previous blog, "Detailed Analysis of macOS/iOS Vulnerability CVE-2019-6231".

0x01 A Quick Look

QuartzCore, also known as CoreAnimation, is a framework used by macOS and iOS to create animatable scene graphics. CoreAnimation uses a unique rendering model where the graphics operations are run in a separate process. On macOS, the process is WindowServer. On iOS, the process is backboard.

The service named com.apple.CARenderServer in QuartzCore is usually referenced as CARenderServer. This service exists in both macOS and iOS, and can be accessed from the Safari Sandbox. A memory corruption vulnerability exists when QuartzCore handles a shape object in the function CA::Render::Decoder::decode_shape() on macOS. This may lead to unexpected application termination.

The following is the crash log of the WindowServer process when this issue is triggered.

0x02 Proof of Concept

In this section I will demonstrate a PoC (Proof of Concept) used to trigger this issue. The PoC is shown below.

A comparison between the original Mach message and the crafted Mach message is shown below.

Figure 1. The diff between the crafted Mach message and the original Mach message

Through binary diff, we only need to modify one byte at offset 0xB6 from 0x06 to 0x86 in order to trigger this issue.

As shown in the PoC's code, in order to send a crafted Mach message to trigger this issue, we first need to send a Mach message with msgh_id 40202 (the corresponding handler in the server is _XRegisterClient) to retrieve the connection ID for every newly-connected client.

Once we get the value of the connection ID, we set this value at the corresponding offset (0x2C) in the crafted Mach message. Finally, we just send this Mach message to reproduce this vulnerability.

0x03 Analysis and Root of Cause

In this section, I will dynamically debug this vulnerability with LLDB to determine the root cause. **Note that you need to debug the WindowServer process via SSH mode.**

Based on the stack backtrace of the crashed thread from the crash log, we could set a conditional breakpoint at the function CA::Render::Server::ReceivedMessage::run_command_stream using the following commands.

The value of conn_id can be obtained by setting a breakpoint at line 86 in the PoC's C code.

After this breakpoint is hit, we can read the buffer data of the crafted Mach message I sent. The register r13 points to the crafted Mach message.

Figure 2. The crafted Mach message CARenderServer received

The function CA::Render::Decoder::decode_object(CA::Render::Decoder *this, CA::Render::Decoder *a2) is used to decode all kinds of object data. The buffer data starting at offset 0x70000907dd52 is an Image object (marked in green).

Figure 3. The crafted Mach message with an abnormal Image object

The following code branch is used to parse the Image object data in the function CA::Render::Decoder::decode_object.

Figure 4. The code branch to handle the Image object data

Next, let's take a closer look at how the Image object is handled.

The following is the function CA::Render::Image::decode(). I add some comments that explain what each field in the Image object means.

Figure 5. The function CA::Render::Image::decode()

We can see that one byte at offset 0x70000907dd52 was mutated from 0x06 to 0x86. So the variable v4 is now equal to 0x86. The program could then jump to LABEL_31 to execute other branch codes because the variable v4 is larger than 0x20. At the end of LABEL_31, the program continues to handle the subsequent data that represents a Texture object by calling the function CA::Render::Texture::decode(CA::Render::Texture *this, CA::Render::Decoder *a2).

Figure 6. The function CA::Render::Texture::decode

We can see that it could invoke the function CA::Render::Decoder::decode_shape to handle the Shape object data.

Let’s continue to trace how the next set of data is handled.

Figure 7. The function CA::Render::Decoder::decode_shape

We can see that the variable v2 is equal to 0x02. It could then allocate a buffer whose size is 8 bytes. Finally, it could invoke the function CA::Render::Decoder::decode_bytes to decode several bytes of data. And this function takes three parameters: The 2nd one points to the previous buffer allocated by the function malloc_zone_malloc. The 3rd one is a size_t type, and could be calculated by the expression “4LL * v2 – 12”, which obviously causes an integer overflow where the result is equal to 0xfffffffffffffffc. So when it calls the function bzero(), its first parameter points to a smaller buffer, but its second parameter is a super large unsigned 64-bits integer, which could lead to memory corruption.

Figure 8. The function CA::Render::Decoder::decode_bytes

The root cause of this issue is that it lacked a restricted bounds check in the function CA::Render::Decoder::decode_shape.

Now that we have now finished the detailed analysis of this vulnerability, let’s look at how Apple fixed it.

Figure 9. The comparison between before patch and after patch

0x04 Conclusion

This vulnerability only affects macOS based on Apple's security update. This issue exists in QuartzCore when handling shape object in the function CA::Render::Decoder::decode_shape() due to the lack of restricted input validation. Through a comparison between code before and after the patch, we can see that this issue was addressed with improved input validation.

0x05 Affected Versions

macOS Mojave 10.14.2

macOS Mojave 10.14.3

0x06 Analysis Environment

macOS 10.14.2 (18C54) MacBook Pro

0x07 Timeline

Discovery date: January 1, 2019

Notification date: January 3, 2019

Confirmation date: March 20, 2019

Release date: March 25, 2019

0x08 Reference

https://support.apple.com/en-us/HT209600
https://www.fortinet.com/blog/threat-research/detailed-analysis-of-macos-ios-vulnerability-cve-2019-6231.html

Learn more about FortiGuard Labs and the FortiGuard Security Services portfolio. Sign up for the weekly FortiGuard Threat Intelligence Briefs.

Learn more about the FortiGuard Security Rating Service, which provides security audits and best practices.

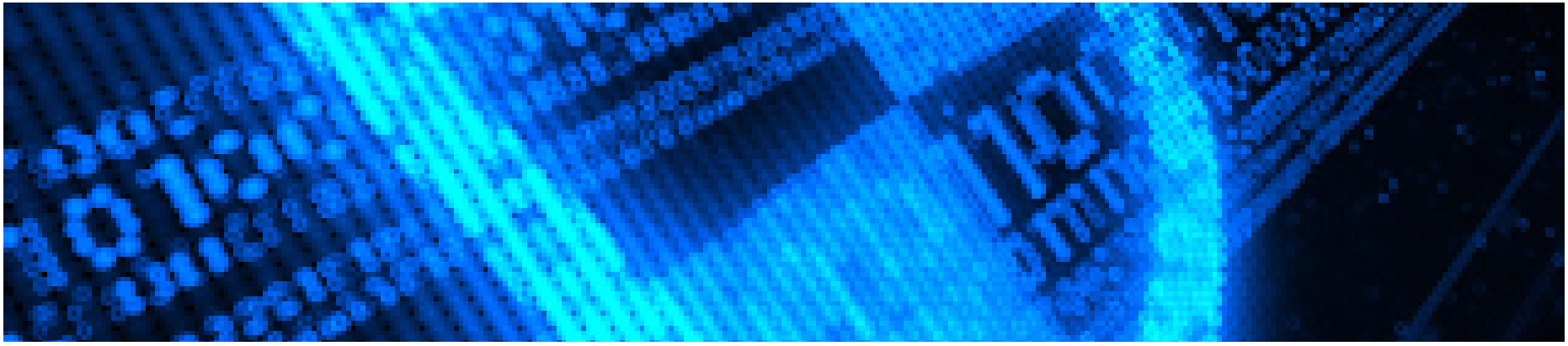
Read more about our Network Security Expert program, Network Security Academy program or our FortiVets program.

Tags: threat research, ios, fortiguard labs, vulnerability, Cybersecurity Architect

Related Posts

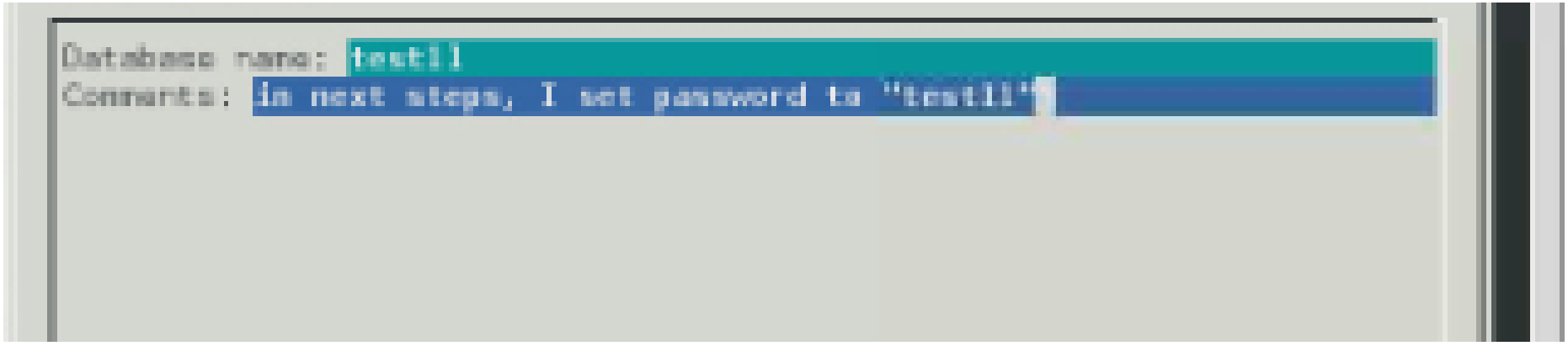
THREAT RESEARCH

Looking Back at Fortinet's Security Research and Vulnerability Discoveries



THREAT RESEARCH

A root cause analysis of CVE-2018-0797 - Rich Text Format Stylesheet Use-After-Free vulnerability



THREAT RESEARCH

Remote Password Change Vulnerability in HPE Vertica Analytic Database

Company

About Us
Why Fortinet
Security Fabric
Exec Mgmt
Careers
Certifications
Events
Industry Awards

Contact Us

(866) 868-3678