Major Vulnerabilities Discovered in Qualcomm QCMAP

By Ori Hollander and Asaf Karas | October 14, 2020
14 min read

SHARE:

In a recent supply chain security assessment, we analyzed multiple networking devices for security vulnerabilities and exposures. During the analysis we discovered and have responsibly disclosed four major vulnerabilities in Qualcomm's QCMAP (Qualcomm Mobile Access Point) architecture that these devices were based on. An attacker that exploits the discovered vulnerabilities can gain remote root access to any of the affected devices.

The QCMAP architecture is in use on most modern Qualcomm-based modem SoCs, such as the MDM9xxx series. This leads us to estimate that these issues can potentially affect millions of devices of many different types in the wild.

In this blog post, we discuss the technical details of the vulnerabilities and how we discovered them, and offer guidance on how to detect and resolve them. Finally, we describe our disclosure process with the Qualcomm team who were aware of the issues and have provided patches that can be applied to fix the issues on vulnerable devices. The first and second vulnerabilities detailed below have been assigned the same CVE number (CVE-2020-3657) by Qualcomm and were listed in the Android Security Bulletin with critical impact (Android bug id A-153344684).

Qualcomm QCMAP – a technical background

Qualcomm manufactures the MDM (Mobile Data Modem) family of SoCs which provides various mobile connectivity features in a single package.

Global Multimode LTE IoT Modem
MDM9206
Qualcomm MDM 9206
Integrated GNSS
Integrated Cortex A7 processor
Hardened Security
One hardware design
One software image
One RF

Source: Qualcomm website

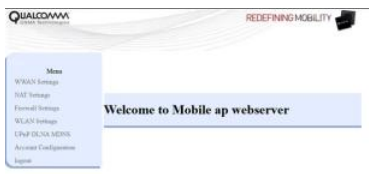
One of the software suites that run on many MDM-based devices is the QCMAP suite.

QCMAP is a software suite in charge of running many services in the mobile access point, which include the following key components:

1. A lighttpd-based web interface (complete with proprietary CGI scripts and authentication mechanisms)
2. A MiniDLNA-based media server (with support for UPnP and mDNS auto-configuration)
3. An iptables-based firewall interface

Modem chipsets using QCMAP are prevalent across a broad range of product types and industries, including networking devices such as mobile hotspots and LTE routers, as well as automotive infotainment/TCU units, smart adapters for industrial equipment, smart metering devices, smart medical equipment and IoT gateways.

QCMAP services are usually configured either via web:



Source: QCMAP web configuration homepage (index.html)

Or, via a dedicated command-line interface, implemented by the QCMAP_CLI binary:

```
5. Set Roaming 53. Set DLNA Media Directory
6. Get Roaming 54. Get DLNA Media Directory
7. Delete DMZ IP 55. Set MobileAP/WLAN Bootup Config
8. Add DMZ IP 56. Get MobileAP/WLAN Bootup Config
9. Get DMZ IP 57. Enable/Disable IPV4
10. Set IPSEC VPN Passthrough 58. Get IPv4 State
11. Get IPSEC VPN Passthrough 59. Get Data Bitrate
12. Set PPTP VPN Passthrough 60. Set UPnP Notify Interval
13. Get PPTP VPN Passthrough 61. Get UPnP Notify Interval
14. Set L2TP VPN Passthrough 62. Set DLNA Notify Interval
15. Get L2TP VPN Passthrough 63. Get DLNA Notify Interval
16. Set Autoconnect Config 64. Add DHCP Reservation Record
17. Get Autoconnect Config 65. Get DHCP Reservation Records
18. Get WAN status 66. Edit DHCP Reservation Record
19. Add Firewall Entry 67. Delete DHCP Reservation Record
20. Enable/Disable M-DNS 68. Activate Hostapd Config
21. Enable/Disable UPnP 69. Activate Supplicant Config
22. Enable/Disable DLNA 70. Get Webserver WWAN access flag
23. Display Firewalls 71. Set Webserver WWAN access flag
24. Delete Firewall Entry 72. Enable/Disable ALG
25. Get WWAN Statistics 73. Set SIP server info
26. Reset WWAN Statistics 74. Get SIP server info
27. Get Network Configuration 75. Restore Factory Default Settings(** Will
Reboot Device )
28. Get NAT Type 76. Get Connected Device info
29. Set NAT Type 77. Get Cradle Mode
30. Enable/Disable Mobile AP 78. Set Cradle Mode
31. Enable/Disable WLAN 79. Get Prefix Delegation Config
32. Connect/Disconnect Backhaul 80. Set Prefix Delegation Config
33. Get Mobile AP status 81. Get Prefix Delegation Status
34. Set NAT Timeout 82. Set/Get Gateway URL
35. Get NAT Timeout 83. Enable/Disable DDNS
36. Set WLAN Config 84. Set DDNS Config
37. Get WLAN Config 85. Get DDNS Config
38. Activate WLAN 86. Enable/Disable TinyProxy
39. Set LAN Config 87. Get TinyProxy Status
40. Get LAN Config 88. Set DLNAWhitelisting
41. Activate LAN 89. Get DLNAWhitelisting
42. Get WLAN Status 90. Add DLNAWhitelistingIP
43. Enable/Disable IPV6 91. Delete DLNAWhitelistingIP
44. Set Firewall Config 92. Set UPNPPinhole State
45. Get Firewall Config 93. Get UPNPPinhole State
46. Get IPv6 State 94. Configure Active Backhaul Priority
47. Get WWAN Profile 95. Get Backhaul Priority
48. Set WWAN Profile 96. Teardown/Disable and Exit
```

We have seen implementations of QCMAP in many device firmware images, including the following models that we have analyzed:

- ZTE MF920V
- TP-Link M7350 (in this device the QCMAP binaries are modified and do not have the found vulnerabilities)
- Netgear AC785

Discovery through software supply-chain security assessment

These issues highlight the importance of identifying and assessing the software supply chain for security. The issues were discovered as the JFrog security research team (formerly Vdoo) analyzed firmware images that are deployed in the field as part of an operational network. These devices were in turn developed by vendors who based their devices on the third-party Qualcomm-based modem. The operator of the network had zero visibility into the usage of that third-party component or to its potential vulnerabilities – known and unknown.

This discovery demonstrates the power and value of automated deep binary analysis in identifying new zero-day vulnerabilities in closed-source components. Using SCA Tools for automated security analysis capabilities delivered by Vdoo (now part of JFrog), the operator analyzed the firmware images received from the vendor. All of the device software components, including Qualcomm's QCMAP binaries, were then scanned and analyzed for potential zero-day vulnerabilities. These potential vulnerabilities were verified by our own research team, and then disclosed to Qualcomm for mitigation.

Vulnerabilities details

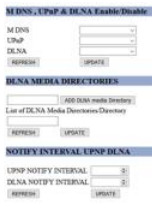
In this section, we provide a deep-dive description of the vulnerabilities in the QCMAP binaries we analyzed. Specifically, we relate to vulnerabilities in the web service of the architecture and in the main binary (QCMAP_ConnectionManager) that connects several architecture services. Code snippets in this post came from publicly available information. The vulnerabilities are a stack-based buffer overflow (CWE-121) and NULL dereference (CWE-476) in the QCMAP webserver and two command injections (CWE-78) in the CLI and connection manager components. The existence and severity of the vulnerabilities depend on the specific implementation of each manufacturer using this architecture. For example, in addition to devices that were fully vulnerable, some of the devices we researched did not run the vulnerable service at all, while in others some of the vulnerable binaries were modified or missing.

VD-1873 / CVE-2020-3657 – Command injection vulnerability (CVSSv3 8.8)

This is the most severe issue we spotted, as it can easily lead to remote code execution (albeit authenticated). The issue resides in the QCMAP_ConnectionManager binary.

Note that use of HTTPS in the web server depends on the device vendor (we have seen both HTTPS and HTTP used in different devices). Without the use of HTTPS, web authentication can be easily bypassed by a local-network attacker.

Part of the basic functionality of the media server is to allow the user to set media directories to publish from. This can be done, for example, via the web interface:



At the implementation level, the CGI handler at cgi-bin/qcmap_web_cgi passes data from the web form to the QCMAP_Web_CLIENT binary which parses the request. The sent data is expected to be in the format "var1=val1&var2=val2& var3=val3…". The first variable is expected to be the "page" variable. If it is set to "SetMediaDir" the code parses the next variables to set the DLNA media directory. It then sends the variables to the QCMAP_ConnectionManager binary, which takes care of the request in the function qmi_qcmap_msgr_set_dlna_media_dir and passes it to QCMAP_MediaService::SetDLNAMediaDir. In this function, the code splits the sent directory by the ',' character, and for each portion, it calls snprintf to create a command, which is then sent as an argument to the system function. There is no check on the user input to make sure that it doesn't include malicious characters, thus it is possible to pass a string with shell metacharacters (such as ';') and run arbitrary commands.

```
{
++str_start;
}
if (str_start != token) /*if leading space is present*/
memmove(token, str_start, strlen (str_start)+1);
}
/*end of removing leading space*/
snprintf(tmp, sizeof(tmp), "%s%s%s", bef, token, aft);
ret = system(tmp);
token = strtok_r(NULL, ",", &ptr);
}
```

For proof of concept, invoke the web interface with the following URL –

http://x.x.x.x/cgi-bin/qcmap_web_cgi?page=SetMediaDir&dir=fakedir;sleep%2010

VD-1871 / CVE-2020-3657 – Stack-based buffer overflow vulnerability (CVSSv3 7.6)

This issue resides in the QCMAP_Web_CLIENT binary and have been numbered as CVE-2020-3657 by Qualcomm together with VD-1873, as it resides in the same function, though it is a different vulnerability altogether with a different CVSSv3 score of 7.6.

QCMAP contains several CGI binaries that can be invoked by a corresponding page served by a web server. One of the pages of the web server is cgi-bin/qcmap_web_cgi. This page checks if the user has a valid session and then sends the information that was read to the CMAP_Web_CLIENT binary.

QCMAP_Web_CLIENT parses the data and performs the chosen operation. The sent data is expected to be in the format "var1=val1&var2=val2& var3=val3…". The Tokenizer function is the function in charge of parsing this format. The function takes the variable names and values and copies them into output buffers sent to it by the main function. The code does not check how many variables were actually passed. Instead it runs until it finishes reading variables, and copies them into the buffer. The buffer is an array of strings (a two-dimensional character array) coming from the "main" function. Since the array has a limited amount of space for variables, after a certain amount of variables the data will overflow out of the buffer. Since the call to the Tokenizer function is done within an infinite loop in the "main" function, classic return address overwriting is usually not possible, however – depending on the exact architecture – there might be multiple ways to exploit the issue to obtain code execution.

Note that the string copying is done by calling the strncpy function with the length of the source buffer instead of the destination buffer, which is equivalent to a simple strcpy call and has no security benefits.

```
ch = strtok_r(buf, "&",&saveptr);
while (ch != NULL)
{
strncpy(columns,ch,strlen(ch)+1);
strncpy(fields[i],ch,strlen(ch)+1);
ch2=strstr(columns,"=");
strncpy(values[i],&ch2[1],strlen(&ch2[1]) +1);
//strtok_r reentrant so that context switch doesn't cause loss of data.
ch = strtok_r(NULL, "&",&saveptr);
i++;
}
```

As is the case with many embedded devices, all of our tested devices did not have critical compiler security mitigations enabled (Stack canary support, Position independent executables, FORTIFY-SOURCE, etc.). This makes this vulnerability (and the next one, CVE-2020-25858) much easier to exploit for code execution, giving attackers full control over the devices and the ability to run arbitrary malicious code (rather than just crashing the vulnerable process).

For proof of concept, invoke the web interface with the following URL –

`http://x.x.x.x/cgi-bin/qcmap_web_cgi?`

`a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=`

`b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&`

`a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=`

`b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b&a=b`

VD-1872 / CVE-2020-25858 – NULL pointer dereference vulnerability (CVSSv3 6.5)

This issue resides in the QCMAP_Web_CLIENT binary.

In the Tokenizer function mentioned before, the strstr function is invoked to search for a '=' character, and its return value is used without checking. In several implementations we've seen that the call to strstr was replaced by a call to strchr. In both cases, if there is no '=' character, the function will return NULL, causing a NULL pointer dereference. This crashes the process.

```
ch = strtok_r(buf, "&",&saveptr);
while (ch != NULL)
{
strncpy(columns,ch,strlen(ch)+1);
strncpy(fields[i],ch,strlen(ch)+1);
ch2=strstr(columns,"=");
strncpy(values[i],&ch2[1],strlen(&ch2[1]) +1);
//strtok_r reentrant so that context switch doesn't cause loss of data.
ch = strtok_r(NULL, "&",&saveptr);
i++;
}
```

For proof of concept, invoke the web interface with the following URL –

http://x.x.x.x/cgi-bin/qcmap_web_cgi?a

VD-1874 / CVE-2020-25859 – Command injection vulnerability (CVSS v3 7.9)

This issue resides in the QCMAP_CLI binary.

QCMAP contains a Command Line Interface (CLI) utility called QCMAP_CLI. From within this CLI the user can change different settings of the device.

One of the possible options is to set the gateway URL. When setting the URL, the set request is sent to the QCMAP_ConnectionManager binary to the function qmi_qcmap_msgr_set_gateway_url, which, at some point, calls the QCMAP_LAN::EnableGatewayUrl function. In this function, the code calls snprintf to create a string including the URL and then calls the system function to create a new process. There is no check on the user input to make sure that it doesn't include malicious characters, thus it is possible to pass a string with shell metacharacters (such as ';') and run arbitrary commands.

This issue appears twice in some of the implementations of the function we saw.

```
addr.s_addr = htonl(this->lan_cfg.apps_ip_addr);
strlcpy(a5_ip, inet_ntoa(addr),QCMAP_LAN_MAX_IPV4_ADDR_SIZE);
snprintf(command,
MAX_COMMAND_STR_LEN,
"echo %s %s >> %s",a5_ip,
this->lan_cfg.gateway_url,QCMAP_HOST_PATH);
ds_system_call(command, strlen(command));
```

For proof of concept perform the next steps –

1. Run the QCMAP_CLI binary
2. Input: 82 <enter>
3. Input: 1 <enter>
4. Input: www;sleep 10 <enter>

devices. However, as the proof of concept snippets above show, exploitation is trivial given the technical knowledge. We will continue to closely monitor any abuse of this vulnerability and will update this blog post accordingly.

Identifying vulnerable devices

Since the vulnerable files have no identifying version information to the best of our knowledge, the issue is not straightforward to detect. If you have access to a live device follow these steps to check if your device is vulnerable:

- Verify that the device is running a web server by trying to connect directly or performing a port scan for relevant ports.
- After successfully logging in to the web server (with cookies allowed) your browser will have a valid session on the device. Now try executing the PoC for VD-1871. If the QCMAP_Web_CLIENT process seems to crash (for example, the web server times out), your device is vulnerable.
- Repeat the same steps as above for VD-1872, invoking the PoC and checking for a crash, and VD-1873, invoking the PoC and checking for a delay in the response.
- Find the QCMAP_Web_CLIENT binary and check if its SHA-256 is one of the following:

  ```
  71311beee4c761f85d46eaadab475541455adbd135f3c868c0800b1703378755
  5f19143efa90161bde6eb129f7b43bdf0a25e86ae7a749dc13b7ea645aa590f5
  e6d505c80de7ccce0cf297715f67e0efbbc30e7427a846ea04d64af1a9e77dae
  0079e76c4c9ca3668789fd4c58c24e66519365c86479f0d7477980d0b6422eed
  0a51f755716a688225573ca4cae469acdf6c6350d83d19098580e8e295692668
  ```

  If it is, your device might be vulnerable.
- If you are running a Qualcomm MDM-based Android device, make sure that your security patch level is updated to October 2020 (or later).

Mitigating vulnerabilities on impacted devices

If your device has been found vulnerable and updating or patching the firmware is not an option, the following mitigation techniques may be applied:

1. If your device does not have to be connected to the network, consider disconnecting it.
2. If the device must be connected to the network, make sure to block access to the web ports by using a firewall and by making sure that they are not forwarded to outer networks.
3. If you can deploy a device behind perimeter protection that contains a WAF, configure it to check for URLs with more than 40 query parameters.

Disclosure process

After disclosing these vulnerabilities to the Qualcomm PSIRT in June 2020, they notified us that the vulnerabilities were already known to them and were patched in 2019 and that they plan to publicly disclose them in October 2020. The two vulnerabilities that have been assigned CVE-2020-3657 were listed in Google's Android security bulletin for October 2020.

Questions? Thoughts? Contact us at research@jfrog.com for any inquiries related to security vulnerabilities.

In addition to discovering and responsibly disclosing vulnerabilities as part of our day-to-day activities, the JFrog security research team works to enhance software security by empowering organizations to discover vulnerabilities through automated security analysis. For more information and updates on JFrog DevOps Platform security features – click here.

Tags: security-research vulnerability disclosure Security Vulnerability

START A TRIAL >

SHARE: