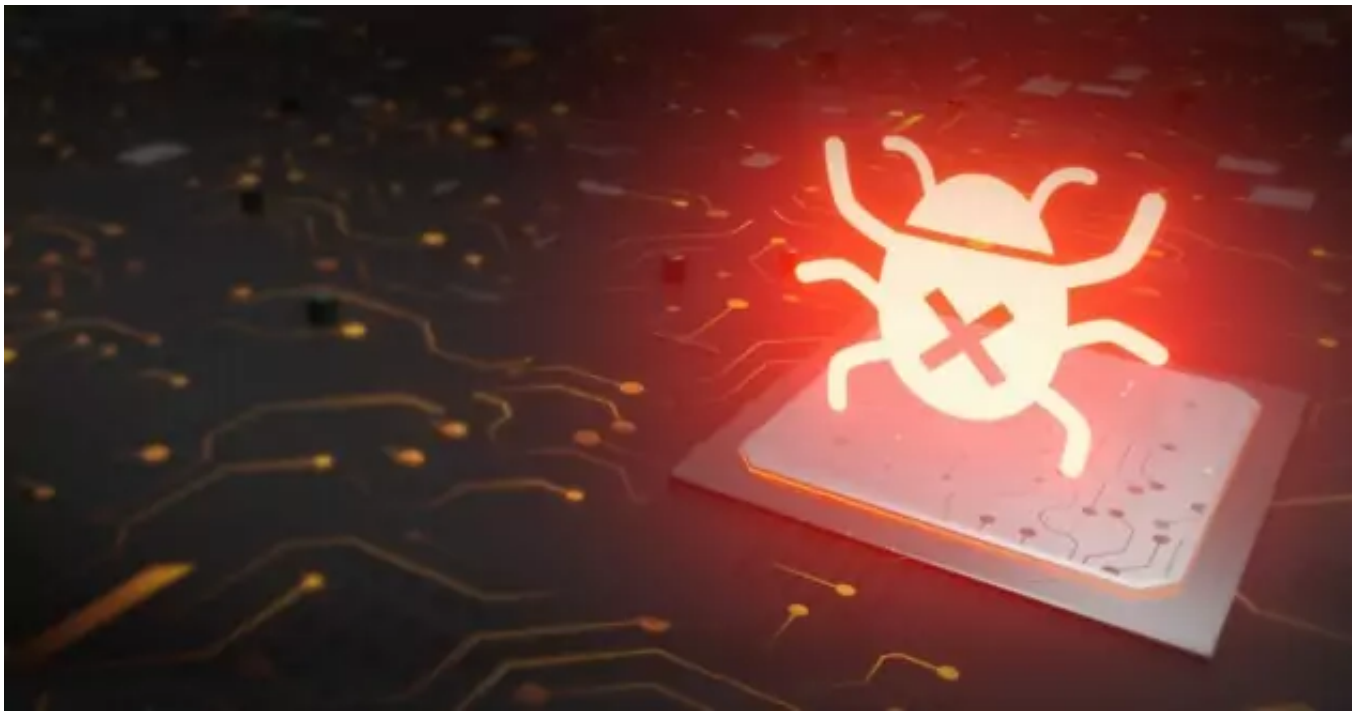
Stealthy WailingCrab Malware misuses MQTT Messaging Protocol

securityintelligence.com/x-force/wailingcrab-malware-misues-mqtt-messaging-protocol/



This article was made possible thanks to the hard work of writer Charlotte Hammond and contributions from Ole Villadsen and Kat Metrick.

IBM X-Force researchers have been tracking developments to the WailingCrab malware family, in particular, those relating to its C2 communication mechanisms, which include misusing the Internet-of-Things (IoT) messaging protocol MQTT.

WailingCrab, also known as WikiLoader, is a sophisticated, multi-component malware delivered almost exclusively by an initial access broker that X-Force tracks as Hive0133, which overlaps with TA544. WailingCrab was first observed in December 2022, and since then it has been used extensively in email campaigns to deliver the Gozi backdoor often against Italian targets. In recent months, Hive0133 has targeted organizations beyond Italy with email campaigns delivering WailingCrab, frequently using themes such as overdue delivery or shipping invoices.

The malware authors have focused on stealth and anti-analysis techniques in the continued development of the WailingCrab malware. The malware itself is split into multiple components, including a loader, injector, downloader and backdoor, and successful requests to C2-controlled servers are often necessary to retrieve the next stage. Legitimate, hacked websites are used for initial C2 communications to lower the chance of network detection, and payloads are often hosted on well-known platforms such as Discord. C2 servers are often taken down quickly or stop responding soon after a campaign which may prevent threat researchers from accessing them and retrieving the next stages of the malware. Additionally, WailingCrab makes use of code obfuscation, anti-analysis, and anti-sandbox techniques throughout its code.

WailingCrab's core component is its backdoor, which is installed on the system only if the malware's initial stages are completed successfully. Since mid-2023, WailingCrab's backdoor component has communicated with the C2 using the MQTT protocol which is a lightweight IoT messaging protocol. MQTT uses a publish/subscribe architecture, whereby messages are published to 'topics' and received by subscribers, with message distribution handled by a centralized broker. In this instance, WailingCrab uses the legitimate, third-party broker, **broker.emqx[.]io**, which allows it to hide the true address of the C2 server.

WailingCrab's use of the MQTT is notable, as this protocol is not commonly used by malware. There have only been a handful of instances reported, with the most recent being the MQsTTang backdoor attributed to the threat actor Mustang Panda. As a result of this, the protocol's use may not be monitored as closely by security teams, allowing the backdoor's C2 communications to fly under the radar.

This blog provides an overview of WailingCrab and its C2 communications, with a focus on its use of the MQTT protocol.

Delivery

Since its inception, WailingCrab has been distributed via email spam campaigns using Microsoft Excel attachments, Microsoft OneNote attachments or PDF attachments. In recent months, Hive0133 has favored the use of PDF attachments containing malicious URLs in their email campaigns delivering WailingCrab. When clicked, the links will download and execute JScript files, which in turn will download and

execute the WailingCrab loader, which is usually hosted as an attachment file on Discord. Below is an example of a Hive0133 email campaign delivering WailingCrab on 19 October.

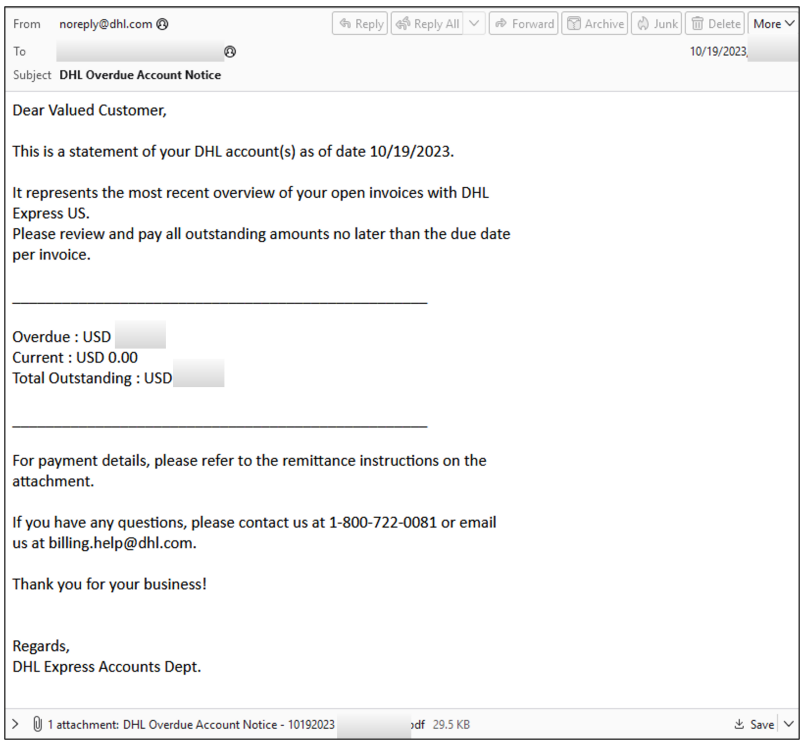


Figure 1: Hive0133 Email from 10/19/2023 delivering WailingCrab Loader.

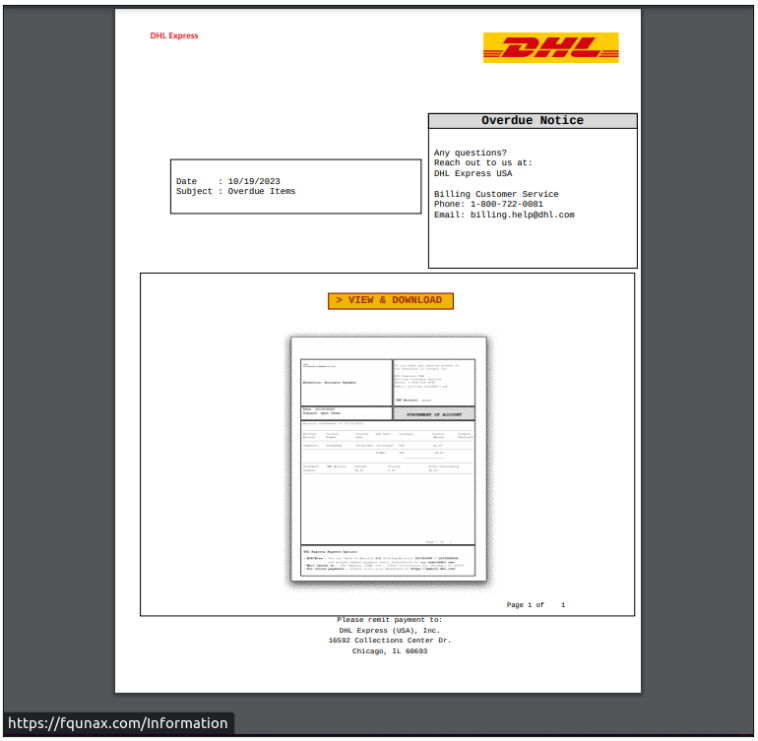


Figure 2: Hive0133 Email PDF Attachment with Malicious Link Leading to WailingCrab Loader.

WailingCrab components

Many of the technical details of WailingCrab's operation and early variants have already been discussed in other research, therefore in this blog, we will focus on new developments and those aspects which have not already been reported on.

The primary samples used for this research are **24c5f4868dc5af255edbb993d98de51a** and **f6ea7ec5d94bc65bf82a6b42b57a6c82**, which were from campaigns in September, and **f6d0b9617405f35bb846d671edda75d3** which was observed in July and is the reference for the earlier version of the MQTT protocol use. These samples are all first-stage WailingCrab Loaders, which were hosted on Discord and downloaded via spam campaigns like the one described above. The subsequent WailingCrab components discussed in this blog were all unpacked or downloaded by these loaders.

WailingCrab loader

The first component of WailingCrab is its loader, which commonly uses a legitimate DLL file as a template, with the malicious code patched over one of the DLL's exported functions. Its purpose is to load the second stage which is stored within the DLL as an encrypted shellcode.

This initial loader component of WailingCrab has received a few updates in more recent samples. In the previous version, the loader would overwrite its own data section in memory with the code for its second-stage component. In the new version, the malware first loads a legitimate Windows DLL, such as **BingMaps.dll**, and then overwrites the code for one of the DLL's exported functions with WailingCrab's second-stage shellcode. It also patches the code at the DLL's original entry point such that it returns immediately rather than running its original code, allowing execution to proceed unimpeded to the maliciously patched export function.

The WailingCrab loader then creates a new thread to run its second-stage shellcode within the context of the legitimate loaded DLL.

WailingCrab injector

The second stage is the WailingCrab injector, the functionality of which has not changed much from the previous version.

The Injector component starts by looping through the currently running processes on the host system and creates a hash of each process filename until it finds one that matches its target hash, which in the analyzed sample corresponds to **explorer.exe**. At this point, the malware will also compare the hash of each running process name to a list of hashes associated with sandbox or debugging applications, and will not continue if any of these are found.

Once it identifies its target process, WailingCrab opens it and allocates memory within the process space. It then decrypts its third-stage component using XOR and writes the decrypted payload contents to the allocated memory region, along with a string containing the file path of the initial loader.

Next, WailingCrab searches the DLLs loaded within the target process (i.e. explorer.exe) and looks for ntdll.dll. Within the ntdll.dll instance, it finds the address of a target API function, in this case, **RtlWow64GetCurrentMachine**. It then overwrites the contents of this function with 12 bytes of trampoline hook code, the purpose of which is to jump to the start of the copied payload.

The malware then creates a new thread within the target process. The start address of the thread is set to that of the hooked API function, e.g. **RtlWow64GetCurrentMachine**. Upon creation, the new thread executes the target API function, which now contains the hook code, and this then transfers execution to the payload, which is the next WailingCrab component.

WailingCrab downloader

The third WailingCrab component is a Downloader/Loader, which is responsible for loading the Backdoor component. The code for this stage is run within the context of the injected process; in this case, **explorer.exe**.

Much of the functionality of the downloader is the same as in previous versions, however, there have been some updates. In prior versions, this component would download the backdoor, which would be hosted as an attachment on the Discord CDN. However, the latest version of WailingCrab already contains the backdoor component encrypted with AES, and it instead reaches out to its C2 to download a decryption key to decrypt the backdoor.

The WailingCrab downloader starts in the same manner as prior versions, by sleeping for a set period, and then deleting the original loader file on disk. It also creates a mutex, where the mutex name is a hardcoded numeric string, for example, "**823264"**.

At this point in the previous version, WailingCrab would then perform connectivity and anti-sandbox checks by attempting to connect to https://www.wikipedia.org/ and also a non-existent domain and confirming that the results of both are what it expects. However, these checks have been removed from the new version, and the malware proceeds straight to C2 communication.

WailingCrab proceeds to register with the C2. It randomly generates bot ID values and also gathers basic system information including domain, hostname, username, language and system time. These values are then formatted into a pipe-delimited string, along with an eight-digit campaign ID which is hardcoded into the malware.

In the previous version, the randomly generated bot ID was a single eight-digit string, however in the new variant three 16-digit strings are generated instead, and these will be used later on by the backdoor component as MQTT topic names during its communication with the C2.

Previous Version:

<campaign_id>||<botid>||<domain>||<hostname>||<username>||<admin_status>||<language_id>||<system_time>||<os_version>||

e.g.

84234775||52270349||-||DESKTOP-JGLLJLD||admin||0||1033||2023.07.12+06:50||10.00.19044||

Scroll to view full table

New Version:

<campaign_id>||<16_digit_bot_value1><16_digit_bot_value2><16_digit_bot_value3>||domain||<hostname>||<username>||<admin_status>||<language_id>||<system_time>||<os_version>||

e.g.

63785091||258318294498701539673492921154037193237834753421||-||DESKTOP-JGLLJLD||admin||0||1033||2023.09.15+11:32||10.00.19045||

Scroll to view full table

This string is then base64 encoded and added to the Cookie field in the HTTP registration request sent to the C2. The C2 URL is chosen at random from a list, and the following URLs were present in the analyzed sample:

https[:]//epikurgroup[.]com/plugins/content/jw_allvideos/jw_allvideos/tmpl/Res

ponsive/oiyqnk182.php?id=1

https[:]//rgjllc[.]pro/wp-

content/themes/sydney/inc/notices/uiqbw123udibjk1d2.php?id=1

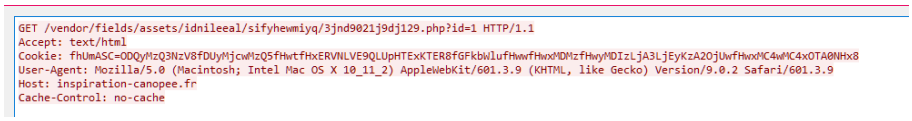
https[:]//advocates4consumerprotection[.]com/wp-

includes/js/tinymce/skins/iudjh9iwd182.php?id=1

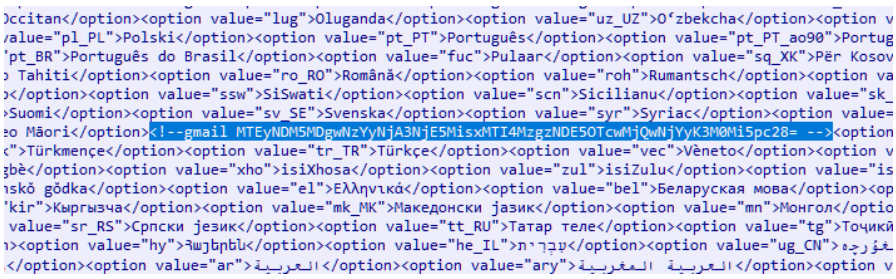
Scroll to view full table

The C2 registration domains are usually legitimate WordPress-based websites that have been compromised by the threat actor to include a malicious PHP file that processes the requests from the WailingCrab malware. When a request is made to one of these URLs with the correct cookie set, the malicious PHP code inserts a comment into the source code of the returned webpage which contains the response data for the downloader.

The below image shows a request being made to one of the C2 URLs, where the cookie field contains the base64 encoded registration string.



If the registration request is successful, the source code of the returned web page will contain a comment containing the word ‘gmail’ followed by a base64 string, similar to that seen in the below image.



In prior versions of WailingCrab, the base64 response would be decoded to reveal a Discord CDN URL path which the next stage of the malware could be downloaded from.

In the new version, the decoded base64 instead contains an encrypted AES key. In this case, the first 8 bytes of the decoded data are XOR’d together, and the final value is then used to XOR-decrypt the remaining 32 bytes of the data which is the AES key.

The AES key is then used by WailingCrab to decrypt the backdoor component using AES-256 in CBC mode with a null-byte IV. The backdoor code is then executed.

WailingCrab backdoor

The WailingCrab backdoor is a sophisticated piece of malware responsible for installing persistence and beaconing to the C2. The backdoor installs itself in a randomly named subdirectory of either the user's %AppData% folder or the %ProgramData% folder. WailingCrab copies several files to this directory and modifies some by overwriting them with chunks of its code:

C:\ProgramData\<malware_dir>\version.dll (copied from C:\WINDOWS\system32\version.dll and modified)
C:\ProgramData\<malware_dir>\<random_name>.exe (copied from C:\WINDOWS\system32\printfilterpipelinesvc.exe)
C:\ProgramData\<malware_dir>\xpspushlayer.dll (copied from C:\ProgramData\<malware_dir>\version.dll)
C:\ProgramData\<malware_dir>\thumbs.db (contains encrypted WailingCrab code)
C:\ProgramData\<malware_dir>\<random_name>.dll (copied from C:\ProgramData\<malware_dir>\version.dll and modified)

Scroll to view full table

WailingCrab installs persistence by creating a randomly named subkey under the registry Run key and adding the file path of the copied and randomly renamed **printfilterpipelinesvc.exe** file. When this file is executed it loads the modified version.dll file via DLL hijacking, and execution then jumps between the various WailingCrab code chunks loaded from the other files, eventually ending up within an injected **explorer.exe** instance. The full technical details of this process are beyond the scope of this blog but are available in our full malware report on X-Force Exchange.

MQTT communication

Communication between the WailingCrab backdoor component and the C2 is performed using the MQTT protocol which is a lightweight IoT messaging protocol. MQTT uses a publish/subscribe architecture, with message distribution handled by a centralized broker. WailingCrab uses a third-party broker, **broker.emqx[.]io**, which allows it to hide the true address of the C2 server.

The basics of the MQTT protocol are quite straightforward. The client starts by sending a connect request to the broker, specifying its client ID, which the broker then acknowledges. After that the client can either publish messages to specific 'topics', or it can request to subscribe to a topic. Any clients who are subscribed to a topic will then receive future messages published on that topic.

Previous version

We initially observed WailingCrab using the MQTT protocol in mid-2023, and this version of the backdoor communicated with the C2 using the following procedure.

1. The backdoor starts by sending a connect request to broker.emqx[.]io using a randomly generated client ID.

Backdoor -> broker.emqx.io
– MQTT Connect packet
– Protocol version 5
– Randomly generated Client ID, e.g. xMMRYbWc

Scroll to view full table

1. It then registers with the C2 by publishing a message to a topic named using the same campaign ID string found in the WailingCrab downloader. The registration message is 25 characters long and consists of the number '1', which is likely the message type, followed by a randomly generated 16-digit string, followed by the eight-digit bot ID generated by the downloader. For example:

Backdoor -> broker.emqx.io (Campaign Topic) -> C2

– MQTT Publish packet

– Topic: Campaign ID (e.g. 84234775)

– Message: Registration type message, e.g. ‘1852460395177254652270349’

1 digit message type = ‘1’ (registration message)

16 digit randomly generated string, e.g. ‘8524603951772546’

8 digit bot ID, e.g. ‘52270349’

Scroll to view full table

1. The purpose of the randomly generated 16-digit string (8524603951782546) is for it to be a client-specific topic name for the C2 and backdoor to communicate via, as opposed to the campaign name topic which is shared by all samples in the same campaign. The backdoor sends a subscribe request to the MQTT broker with the topic name set to this 16-digit string, which means that the broker will forward any future messages published to this topic to the backdoor. Once the subscription request is sent, the broker will then respond with a ‘subscribe ack’ packet to acknowledge the subscription request.

Backdoor -> broker.emqx.io

– MQTT Subscribe packet

– Subscribe Topic: Generated client-specific topic name e.g. 8524603951772546

Scroll to view full table

1. The backdoor then publishes a general ‘check-in’ type message to the campaign topic, which consists of the character ‘2’ followed by the eight-digit bot ID.

Backdoor -> broker.emqx.io (Campaign Topic) -> C2

– MQTT Publish packet

– Topic: Campaign ID (e.g. 84234775)

– Message: Check-in type message, e.g. ‘252270349’

1 digit message type = ‘2’ (check-in message)

8 digit bot ID, e.g. ‘52270349’

Scroll to view full table

1. At this point, the backdoor checks for any received messages. The C2 will publish any messages for the target to the client-specific topic, which the broker will then forward to the backdoor since it has subscribed to that topic. The messages from the C2 take the form of the character ‘0’ if the C2 does not have any further instructions for the client, or the character ‘2’ followed by a download path:

C2 -> broker.emqx.io (Client Topic) -> Backdoor

– MQTT Publish packet

– Topic: Client Specific (e.g. 8524603951772546)

– Message: Either ‘0’ or payload message, e.g. ‘2+113557834996881826 9+114315240013221068 8+djibh1ud21dORD1s.iso|1’

1 digit message type = ‘0’ or ‘2’

Download path with ‘/’ characters replaced by ‘+’, e.g. ‘+113557834996881826 9+114315240013221068 8+djibh1ud21dORD1s.iso’

Pipe delimiter, followed by single digit, usually ‘1’ or ‘2’

Scroll to view full table

1. If the backdoor receives a download path from the C2 it will append the received path string to the URL https[:]//cdn.discordapp[.]com/attachments/ to create the full download URL. Any '+' characters in the path are converted to '/'. The backdoor will then download a payload from the constructed URL, decode it using base64 and then decrypt it using the same XOR-based algorithm used throughout the malware. The malware will generate a random filename with the .log extension, write the payload to the user's Temp directory, and then execute the file by creating a new process with the following command:

```
rundll32.exe <payload_filepath>, DllRegisterServer
```

Scroll to view full table

1. The backdoor will then report the status back to the C2 by publishing a message to the campaign channel. If the payload download and execution operation was a success then it will send a message consisting of the character '3' followed by the bot ID. Otherwise, if there was an error, it will send the character '4' followed by the bot ID.

```
Backdoor -> broker.emqx.io (Campaign Topic) -> C2
  – MQTT Publish packet
  – Topic: Campaign ID (e.g. 84234775)
  – Message: Result type message, e.g. '352270349'
    1 digit message type = '3' or '4'
    8 digit bot ID, e.g. '52270349'
```

Scroll to view full table

1. The malware will then disconnect the MQTT connection and then sleep for a fixed period of time before checking in with the C2 again.

New version

Newer versions of WailingCrab, observed from September 2023 onwards, use an updated protocol when communicating with the C2.

As described above, in the previous version, all clients infected by a specific campaign would register with and send regular check-in messages to a single centralized MQTT topic named after the campaign ID. The initial registration message sent by each client would then contain a randomly generated 16-character numeric string which would then serve as the name of a client-specific topic where the C2 could send commands/payloads to the respective client.

In the new version, the use of the centralized topic has been removed, and infected clients and the C2 now communicate solely via client-specific topics, the names of which are taken from the three randomly generated 16-digit strings created by the downloader component and shared with the C2 as part of its initial request to the WordPress C2 URL.

For example, if the initial registration request sent by the downloader component contained the following:

```
63785091||2583182944987015396734929211540371932378347534 21||-||DESKTOP-
JGLLJLD||admin||0||1033||2023.09.15+11:32||10.00.19045||
```

Scroll to view full table

Then the three client-specific topic names would be:

1. 2583182944987015
2. 3967349292115403
3. 7193237834753421

Scroll to view full table

The use of Discord for hosting payloads has also been removed from this stage of the malware, and the backdoor now receives a shellcode-based payload directly from the C2 via MQTT rather than a Discord-based download path.

A full breakdown of the new C2 communication protocol is as follows:

1. As with the previous version, the backdoor starts by sending an MQTT connect packet to broker.emqx[.]io via TCP port 1883 using a randomly generated eight-character client ID.

Backdoor -> broker.emqx.io

– MQTT Connect packet

– Protocol version 5

– Randomly generated Client ID, e.g. xMMRYbWc

Scroll to view full table

1. It then retrieves the local time of the infected system and constructs a datetime structure which it encodes using base64. The backdoor then publishes a message containing the base64 string, with the topic set to the second of the randomly generated topic names.

Backdoor -> broker.emqx.io (Client Topic 2) -> C2

– MQTT Publish packet

– Topic: Client topic 2 (e.g. 3967349292115403)

– Message: Check-in message, e.g. 'AgAcABwAFAAKAOcH'

Base64 encoded datetime value, e.g. 02 00 1c 00 1c 00 14 00 0a 00 e7 07

Breaks down as six 2–byte values: Hour Min Sec Day Month Year

Scroll to view full table

1. The backdoor then sends a subscribe request to the third topic.

Backdoor -> broker.emqx.io

– MQTT Subscribe packet

– Subscribe Topic: Client topic 3 (e.g. 7193237834753421)

Scroll to view full table

1. The backdoor checks to see if it has received any messages via the subscribed topic, and then sends a disconnect packet. If no message is received, the malware proceeds to sleep and then restart the communications loop. If a message has been received, the backdoor checks that the length is greater than 128 bytes, and if so proceeds to decode it from base64. The first 8 bytes of the decoded payload contain a payload ID value, and the second set of 8 bytes is used to calculate the XOR key to decrypt the rest of the payload data. The backdoor expects the decrypted payload to be another shellcode component which it then executes in a new thread.

C2 -> broker.emqx.io (Client Topic 3) -> Backdoor

– MQTT Publish packet

– Topic: Client topic 3 (e.g. 7193237834753421)

– Message: Base64 encoded shellcode payload

Backdoor -> broker.emqx.io

– MQTT Disconnect packet

Scroll to view full table

1. If a payload is received then, the backdoor reconnects to the MQTT client using a new randomly generated client ID. It then publishes a message with the character '0' to the third client-specific topic acknowledging receipt of the payload, and publishes a second message to the first client-specific topic with the results of loading the payload. This message consists of the eight-byte payload ID followed by either the character '3' if the payload was executed successfully, or the character '4' if an error was encountered. The backdoor then sends a disconnect packet and proceeds to sleep before restarting the communications loop.

Backdoor -> broker.emqx.io

– MQTT Connect packet

– Protocol version 5

– Randomly generated Client ID

Backdoor -> broker.emqx.io (Client Topic 3) -> C2

– MQTT Publish packet

– Topic: Client topic 3 (e.g. 7193237834753421)

– Message: Acknowledgement of payload ‘0’

Backdoor -> broker.emqx.io (Client Topic 1) -> C2

– MQTT Publish packet

– Topic: Client topic 1 (e.g. 2583182944987015)

– Message: Result type message

1 digit message type = ‘3’ or ‘4’

8 digit payload ID

Backdoor -> broker.emqx.io

– MQTT Disconnect packet

Scroll to view full table

Conclusion

The move to using the MQTT protocol by WailingCrab represents a focused effort on stealth and detection evasion. The MQTT protocol is currently not commonly used by malware. It therefore is unlikely to come under much scrutiny by existing security solutions, especially in environments that use MQTT for legitimate IoT traffic. However, as MQTT is primarily used for IoT traffic, this may also make malicious use of it easier to detect in environments or systems that should not have IoT-related activity.

The newer variants of WailingCrab also remove the callouts to Discord for retrieving payloads, further increasing its stealthiness. Discord has become an increasingly common choice for threat actors looking to host malware, and as such it is likely that file downloads from the domain will start coming under higher levels of scrutiny. Therefore, it is not surprising that the developers of WailingCrab decided on an alternative approach.

The upgrades to the C2 communication protocol have also been an unfortunate blow to security researchers. In the initial version, the use of the communal campaign topic made it relatively straightforward to observe the malware’s activity. The fact that WailingCrab uses a public broker means that anyone could subscribe to the campaign topic and monitor the messages being sent to it. In the new version the developers have switched to communicating via client-specific topics only, and unfortunately removing this wider visibility of the malware’s activity.

Recommendations

- Ensure anti-virus software and associated files are up to date
- Search for existing signs of the indicated IOCs in your environment
- Consider blocking and or setting up detection for all URL and IP-based IOCs
- Consider blocking or monitoring the use of the MQTT protocol, especially in environments or systems that should not have IoT-related activity
- Keep applications and operating systems running at the current released patch level
- Exercise caution with attachments and links in emails.

IOCs

Indicator	Indica Type

24c5f4868dc5af255edbb993d98de51a	MD5
f6ea7ec5d94bc65bf82a6b42b57a6c82	MD5
f6d0b9617405f35bb846d671edda75d3	MD5
971dd6c48909adf98861fb8457125faa	MD5
f03e8c10c84f7ba65caa84e194831a34	MD5
9e20801f9f4dc6f871d67470583dde0c	MD5
https[:]//epikurgroup[.]com/plugins/content/jw_allvideos/jw_allvideos/tmpl/Responsive/oiyqnk182.php?id=1	URL
https[:]//rgjllc[.]pro/wp-content/themes/sydney/inc/notices/uiqbw123udibjk1d2.php?id=1	URL
https[:]//advocates4consumerprotection[.]com/wp-includes/js/tinymce/skins/iudjh9iwd182.php?id=1	URL
https[:]//inspiration-canopee[.]fr/vendor/fields/assets/idnileeal/sifyhewmiyq/3jnd9021j9dj129.php?id=1	URL
https[:]//www.p-e-c[.]nl/wp-content/themes/twentytwentyone/hudiiiwj1.php?id=1	URL
https[:]//vivalisme[.]fr/forms/forms/kiikxnmlogx/frrydjqb/vendor/9818hd218hd21.php?id=1	URL
https[:]//tournadre.dc1-mtp[.]fr/wp-content/plugins/kona-instagram-feed-for-gutenbargwfn/4dionaq9d0219d.php?id=1	URL
https[:]//studiolegalecarduccimacuzzi[.]it/Requests/tmetovcqhnisl/vendor/gyuonfuv/languages/vgwtdpera/Requests/5i8ndio12niod21.php?id=1	URL
https[:]//erbilmail[.]com/ind9010j29d0j2.php?id=1	URL
https[:]//www.flow[.]enterprises/wp-admin/css/piudbnui91nid2s1.php?id=1	URL
https[:]//luna-render[.]com/wp-admin/css/pinqidnmwm1192.php?id=1	URL
https[:]//loopmatrix[.]in/dbh9182hdj1o2mde.php?id=1	URL
https[:]//humandata[.]solutions/wp-admin/js/diub890hd91222.php?id=1	URL
broker.emqx[.]io	Doma
https[:]//cdn.discordapp[.]com/attachments/1128405963062378558/1128406314452799499/dw4qdkjbqwijhdhbwqjid.iso	URL
https[:]//cdn.discordapp[.]com/attachments/1148282858646016085/1148283708323938354/dj12iodd12iond.pic	URL
https[:]//cdn.discordapp[.]com/attachments/1151584859765080094/1151616346656886885/d17b28d9d2m9od92.avi	URL

https[:]//cdn.discordapp[.]com/attachments/1124390807626076192/1128383419970240662/s42.iso	URL

Scroll to view full table

Data Protection | Data Security | IBM X-Force Research | Security Intelligence | security intelligence & analytics | X-Force

Charlotte Hammond
Malware Reverse Engineer, IBM Security

Ole Villadsen
Cyber Threat Hunt Analyst, IBM Security

Kat Metrick
Security Consultant, IBM X-Force