How WellMess malware has been used to target COVID-19 vaccines

pwc.co.uk/issues/cyber-security-services/insights/cleaning-up-after-wellmess.html

Copy link

16 Jul 2020

The UK's National Cyber Security Centre has reported that malware known as WellMess has been used in attempts to steal information and intellectual property from multiple organisations involved in COVID-19 vaccine development, amongst targeting of several other sectors.

The report, published with Canada's Communications Security Establishment and the US National Security Agency, attributed WellMess to the threat actor we track as Blue Kitsune (known in open source as APT29, Cozy Bear, and the Dukes).

WellMess malware was first reported in open source by LAC[1] and JPCERT[2] in mid-2018. We have analysed newer variants of the malware from 2020, which have a wide range of added functionality from the original samples including several robust network communication methods and the ability to run PowerShell scripts post-infection.

The following analysis is taken from our private intelligence reporting services, and describes the WellMess malware's capability and functionality.

WellMess analysis

The WellMess sample we examine in this insight is written in Go, though .NET WellMess samples also exist. The Go malware comes in 32-bit and 64-bit variants as both ELF and PE files, allowing the threat actor to develop the malware once and deploy it to many different architectures.

SHA-256	0322c4c2d511f73ab55bf3f43b1b0f152188d7146cc67ff497ad275d9dd1c20f
First seen	2020-04-06 01:09:29
Go version	1.8

The most recent samples of WellMess differ from the 2018 samples as they now support communicating with the C2 server via three separate communication methods. The sample analysed in this report is typical of the latest generation of WellMess.

The malware builds a pipe-separated userParameters string based on the victim machine's computer name, user domain, user name and several hardcoded values in the malware, and stores it for use throughout the malware's execution. An example string would be:

`computername|`**`console`**`|userdomain|username|`**`NTSUUpIIaFEU|104`**

The bold values are taken from data hardcoded in the binary and likely represent campaign or platform information about the running malware. In this sample a copy of the string then has the NTSUUpIIaFEU string appended for a second time and then MD5 hashed which is stored for later use.

As mentioned, the malware can communicate with the C2 server via three communication methods with each being enabled or disabled on a per sample basis.The three methods are:

- HTTP
- HTTPS
- DNS

In this sample the DNS communication method is not enabled; however, the method has been documented for completeness as the functionality is present in the malware.

For each communication method the malware follows a similar process; it establishes a connection with the C2 and then enters an infinite loop to exchange data. The details of the initial connection differ for each method but the main loop that exchanges data uses the same functions to carry out malicious functionality.

HTTP communications

The HTTP mode is the same communication method used in variants of the malware from 2018. Although it uses the non-encrypted HTTP protocol to communicate with the C2 it manually encrypts the contents of the requests to hide data from packet inspection. The malware creates an AES session key and initial value (as detailed in Appendix C) which are base64 encoded, appended to each other with a \n separator. Once appended, it is further encrypted with a hardcoded RSA public key and base64 encoded again and obfuscated before being sent to the C2 as the body of a POST request.

Figure 1. Initial message format

The obfuscation routine used on the final base64 data removes and replaces characters in the base64 string, according to Table 1:

Table 1. Character replacements

Plain text symbol	Obfuscated data
+	,
/	:
=	Three spaces

The string is then chopped into randomly chosen 3-8 character chunks and each chunk is separated with a single space. The final string is then set as the content of a HTTP POST request to the hardcoded IP and port present in the malware. The POST request also creates an obfuscated cookie to include in the POST request. The cookie is a string consisting of tags that contain information about the malware which is then RC6 encrypted with a hardcoded key, base64 encoded and split into random sized chunks. The first cookie sent to the C2 includes the previously generated MD5 session hash and has the following format before encryption:

```
md5_hash/pa:1_0p
```

Encrypted cookies generated by the malware can be decrypted with a script[3] provided by JPCERT, so long as the RC6 key has been obtained for the sample that generated the cookie.

Once the AES key and IV have been sent to the C2 the malware sends the userParameters string and the saved MD5 hash separated by a newline character in the same way. The only difference from the previous send is that the <;title;> tag contains the string a:1_1 this time.

After the initial connection to the C2 has been established, the malware will repeatedly attempt to receive a command from the C2. The communications follow the pattern shown in Figure 2.

Figure 2. Communication pattern

The 2398 random bytes of data are base64 encoded and then obfuscated as detailed previously in Table 1. This data is then sent to the C2 with an RC6 encrypted cookie with the <;head;> tag containing the MD5 session hash value, the <;title;> tag containing the string rc and the <;service;> tag being present but empty. The malware expects the server to respond with a HTTP 200 OK response that contains an RC6 encrypted cookie in a similar format to the sent cookie. This is checked in the malware by parsing the received cookie data according to the following regex which also matches the format of the sent cookie data.

```
[^;]*?);>(?P[^
```

The cookie is used by the malware to determine what type of command has been received from the C2; Table 2 summarises the possible options.

Table 2. Cookie tag details

Tag	Value	Description
head	G	Send a beacon to the C2 to maintain the connection
head	C	Start downloading multiple chunks of data from the C2
title	integer	The number of chunks to be downloaded from the C2 when in C mode
service	p	Calculate a new AES key and send it to the C2
service	f	Tells the malware the received data is to be decoded as base64 rather than UTF8
service	u	Updates the user agent used in communications to be a string sent from the C2
service	m	Updates the maximum size of data to be sent in one POST request to the C2
service	hi	Updates the wait time used by the G tag
service	pr	Changes the enabled protocol list
service	fu	Writes a file to the victim
service	fd	Sends a file from the victim to the C2
service	No Value	The contents from the C2 are executed as either command line or PowerShell script commands

If the response from the C2 contains a value of p in the service tag received, then the malware will generate a new AES key and send it to the C2 in the same way as during the initial connection to the C2.

A value of G in the head tag will make the malware send the string “Missed me? Interval=” with the currently configured value that is used by the malware to calculate wait intervals when receiving this command.

A value of C in the head tag will check a <;title;> tag for the number of chunks the C2 wants to send to the malware and will receive data and store it for later use until the title tag value in a received message matches the desired number of chunks from the C2. This method will be used when the C2 wishes to send a large amount of data to the malware but does not want to alert network monitoring tools that a large data transfer is occurring.

The service tag values, except for p, are used when interpreting the content of the data sent to the malware. The content of the data is stored in the Content segment of the 200 OK responses from the C2. The data in the content section is converted from the obfuscated format detailed in Table 1, base64 decoded and then decrypted using the AES key that was previously sent to the C2.

Once all the data has been received and decoded from the C2, the malware will call a function called botlib.Work which is responsible for the malicious behaviour of this malware. This function handles the u, m, hi, fu, fd and pr service tags as well as having a default case where the service tag does not contain any data. The fu commands take the contents of the response from the C2 as the file to write and parses the service tag for the file name and path to use. The service tag is expected to be in the following format:

```
fu|filepath|filename
```

The tag is split using the ‘|’ character and if the **filepath** parameter is present, then the data is written to a file using that parameter as the file name and path. If the path is not present, then the malware instead writes the file to the current directory the malware is in with a file name of upload.**filename**. The fd command uses the same format to find the file to send to the C2; however, if the **filepath** parameter is not present the malware will look for a file in the current folder called download.**filename**.

The pr command also utilises the ‘|’ as a separator with each protocol separated by the symbol. An example service tag that would enable all of the communication methods in the sample analysed would be:

```
pr|1|2|3
```

The default case when the service tag is empty allows the malware to treat the contents of the response from the C2 as a command to execute via the Go library functions os.exec.Command or os.exec.Start. The format of the received command is checked against the below regex pattern for validity before executing and the command is read from the body of the message received from the C2.

```
fileName:(?P.*?)\sargs:(?P.*?)\snotwait:(?P.*)
```

This method for running commands is slightly different from the older .NET variants of the backdoor which additionally supported running scripts via PowerShell.

HTTPS communications

When the backdoor is configured to use HTTPS to communicate with the C2, the functionality is largely the same as when in HTTP mode. The differences are that it lacks the options to update a session key due to encryption being handled by the TLS layer and it also does not have the option to send data to and from the C2 in the chunking mode previously described. In addition, only one transmission is made to the C2 when the malware is establishing a connection as there is no exchange of an AES session key. The hello message that is sent contains the same plaintext data as the HTTP mode.

```
Computername|console|userdomain|username|NTSUUpIIaFEU|104\nMD5_hash_string
```

However, the title tag has just the character a rather than a:1_1 that is used for the HTTP beacon.

The communication pattern between the victim and the C2 also follows the same pattern as described in Figure 2 but the outer layers of obfuscating the data and RSA encryption are removed due to the TLS protocol handling the encryption. The malware secures the communications between itself and the C2 by implementing a feature called mutual TLS.

Usually TLS operates in simple mode, which only requires a server to carry out authentication. Mutual TLS means the malware has a trusted SSL certificate hardcoded in its binary that it uses to check the certificate provided by the C2; the malware also forwards its own hardcoded certificate to the C2 that the C2 checks. So long as both the C2 and the malware accept the provided certificates then communications will be encrypted using TLS, otherwise the connection will be rejected and the malware will not be able to communicate with the C2.

When looking at the certificates embedded in the samples, it appears that the threat actor uses a self-signed certificate authority to sign both the C2 and the malware's certificates. The WellMess malware stores the C2 IP addresses it uses in the binary as a plaintext url. The format of the C2 url string for the example in Figure 3 would be one of:

- hxxp[:]//45.152.84[.]57:port_number; or,
- hxxps[:]//45.152.84[.]57:port_number.

This pattern can be used to identify numerous samples of the WellMess backdoor.

DNS communications

The DNS communication protocol is significantly different to the previous two modes. Data is sent to the C2 by DNS tunnelling. This involves prepending encoded data as a sub domain to a threat actor-controlled domain and then using DNS requests to transmit this data to the C2 with other DNS resolution services forwarding on the message.

The implementation of DNS tunnelling in this sample uses A records to send data to the C2 and then uses TXT records to receive commands during its main loop.The initial message sent to the C2 consists of the current time printed in DD/MM/YYYY HH:MM:SS AM/PM format. This is then encoded using a non-standard base32 encoding with an alphabet of ybndrfg8ejkmcpqxot1uwisza345h769. This time value is then used throughout the execution of the malware as part of the DNS beacon message to the C2.

The DNS method builds a URL string by splitting the encoded string into four parts of random length, separating each with a period. It then gzips and RC4 encrypts the previously generated MD5 hash string using a hardcoded key. As this sample did not have DNS functionality enabled, the key was not configured in this binary. The result of the RC4 operation is then encoded using the base32 encoding and then that is split into a period-separated string with each chunk a random length.

If the result of the manipulation of the MD5 hash gave the string abcdefg12345, the resulting period separated string could end up being something like .ab.cde.fg12.245. Each chunk can have a maximum length of 12 characters.

The previously encoded four-part message then has the command of this operation appended to it, in the initial beacon's case the command is id. That then has the MD5 string appended to it followed by the malicious domain. The whole message is then prepended with a sequence identifier in case the malware needs more than one DNS request to fit the information in. A couple of example strings with sections bolded to show the boundaries are included below to demonstrate the encoding:

a.0.0.gy516c.jufh3dycto.ryaukqtwga.7dgp3ykbgo.**id**.sr.nn.6r.xyf.j61.mi.it.nhi.fif.pp.9k.

a.0.0.gy516cjwfh.3dyctor.yaurqto.go7dnp3ykbgo.**id**.srnn6rxyfj.61miitnhif.ifpp9kgpf9p.wz3

As can be seen in the above examples, even when the contents of the message change, the part of the URL relating to the MD5 hash remains consistent albeit with different length chunks for each send.

When sending beacons during the main loop in DNS mode the malware will generate two base64 encoded strings from seven random bytes each and join them with a period to the command rc. It will then append the previously generated time string after separating it into four separate chunks. Finally, it will append another 10 base64 encoded strings, each generated from 7 random bytes. Finally the C2 domain is appended to the string and this is sent to the C2 as a DNS TXT request.

```
rc.IIiq2TtU8Q.njp7XG1_7A.gy516c.jwfh3d.yctoryau.rqtu8r7drqjykbgo.XaXwo1pH0w.owxdUchyYw
```

The response is expected to contain three TXT records. The first specifies the command that is being sent from the C2 and can be either of C or G. These commands operate in the same way described previously. If the first command does not match either of these values then the malware reads the data from the third TXT record, decodes it using the custom base32 algorithm, RC4 decrypts the result, then gzip decompresses the data to give the final data to be used.

The commands expected in this data are either the pr command, as previously described, or a com command. The com option calls the same functions as the default case in the HTTP and HTTPS modes which allows the malware to run a cmd shell but then exfiltrates the output via DNS A records.

Conclusion

In contrast to how it has abandoned some previous tooling, Blue Kitsune doubled down on its WellMess backdoor by improving its functionality even after it was exposed by open source reporting. The added functionality to support three separate communication protocols likely improves the malware's ability to avoid detection from network monitoring when the original HTTP protocol is not desired.

References

[1] Cyber Emergency Center, https://www.lac.co.jp/lacwatch/pdf/20180614_cecreport_vol3.pdf, 2018 (Japanese language source)

[2] 'Malware "WellMess" Targeting Linux and Windows', JPCERT, https://blogs.jpcert.or.jp/en/2018/07/malware-wellmes-9b78.html (6th July 2018)

[3] 'JPCERTCC/aa-tools', GitHub, https://github.com/JPCERTCC/aa-tools/blob/master/wellmess_cookie_decode.py

[4] 'ISO 3166 Country Codes', ISO, https://www.iso.org/iso-3166-country-codes.html

- TTPs
- Hashes and IP

TTPs

Application Layer Protocol: DNS - https://attack.mitre.org/techniques/T1071/004/

Application Layer Protocol: Web Protocols - https://attack.mitre.org/techniques/T1071/001/

Data Encoding: Standard Encoding - https://attack.mitre.org/techniques/T1132/001/

Data Encoding: Non-Standard Encoding - https://attack.mitre.org/techniques/T1132/002/

Data Obfuscation: Junk Data - https://attack.mitre.org/techniques/T1001/001/

Data Obfuscation: Protocol Impersonation - https://attack.mitre.org/techniques/T1001/003/

Exfiltration Over C2 Channel - https://attack.mitre.org/techniques/T1041/

Encrypted Channel: Symmetric Cryptography
- https://attack.mitre.org/techniques/T1573/001/

Encrypted Channel: Asymmetric Cryptography
- https://attack.mitre.org/techniques/T1573/002/

Native API - https://attack.mitre.org/techniques/T1106/

Non-Standard Port - https://attack.mitre.org/techniques/T1571/

More detailed information on each of the techniques used in this report, along with mitigations, can be found on the linked MITRE pages.

Hashes and IP

45.120.156[.]69	IP
188.241.68[.]137	IP
178.211.39[.]6	IP
220.158.216[.]130	IP
119.160.234[.]194	IP
31.170.107[.]186	IP
193.182.144[.]105	IP
191.101.180[.]78	IP
209.58.186[.]240	IP
103.205.8[.]72	IP
185.217.92[.]171	IP
93.113.45[.]101	IP

5.199.174[.]164	IP
103.103.128[.]221	IP
27.102.130[.]115	IP
119.81.184[.]11	IP
103.13.240[.]46	IP
101.201.53[.]27	IP
45.123.190[.]168	IP
103.216.221[.]19	IP
103.253.41[.]82	IP
209.58.186[.]197	IP
111.90.150[.]176	IP
141.255.164[.]29	IP
146.0.76[.]37	IP
45.152.84[.]57	IP
169.239.128[.]110	IP
185.145.128[.]35	IP
149.202.12[.]210	IP
103.253.41[.]102	IP
122.114.226[.]172	IP
85.93.2[.]116	IP
103.253.41[.]90	IP
119.160.234[.]163	IP
103.253.41[.]68	IP
81.17.17[.]213	IP
185.99.133[.]112	IP
66.70.247[.]215	IP

119.81.173[.]130	IP
120.53.12[.]132	IP
185.120.77[.]166	IP
176.119.29[.]37	IP
209.58.186[.]196	IP
145.249.107[.]73	IP
122.114.197[.]185	IP
79.141.168[.]109	IP
202.59.9[.]59	IP
46.19.143[.]69	IP
31.7.63[.]141	IP
119.81.178[.]105	IP
111.90.146[.]143	IP
2daba469f50cd1b77481e605aeae0f28bf14cedfcd8e4369193e5e04c523bc38	SHA256
2285a264ffab59ab5a1eb4e2b9bcab9baf26750b6c551ee3094af56a4442ac41	SHA256
b75a5be703d9ba3721d046db80f62886e10009b455fa5cdfd73ce78f9f53ec5a	SHA256
f3af394d9c3f68dff50b467340ca59a11a14a3d56361e6cffd1cf2312a7028ad	SHA256
8749c1495af4fd73ccfc84b32f56f5e78549d81feefb0c1d1c3475a74345f6a8	SHA256
00654dd07721e7551641f90cba832e98c0acb030e2848e5efc0e1752c067ec07	SHA256
0322c4c2d511f73ab55bf3f43b1b0f152188d7146cc67ff497ad275d9dd1c20f	SHA256
0b8e6a11adaa3df120ec15846bb966d674724b6b92eae34d63b665e0698e0193	SHA256
bec1981e422c1e01c14511d384a33c9bcc66456c1274bbbac073da825a3f537d	SHA256
93e9383ae8ad2371d457fc4c1035157d887a84bbfe66fbbb3769c5637de59c75	SHA256
0c5ad1e8fe43583e279201cdb1046aea742bae59685e6da24e963a41df987494	SHA256
4c8671411da91eb5967f408c2a6ff6baf25ff7c40c65ff45ee33b352a711bf9c	SHA256
5ca4a9f6553fea64ad2c724bf71d0fac2b372f9e7ce2200814c98aac647172fb	SHA256

1fed2e1b077af08e73fb5ecffd2e5169d5289a825dcaf2d8742bb8030e487641	SHA256
04169cc11e4d21fc63eefc120fe815b05bd08abf	SHA1
123f62a04a007c1ad81b9686ff27445b51054d4b	SHA1
ecde28e1b879e5a80630d2450b489dfa09c23ea7	SHA1
8830e9d90c508adf9053e9803c64375bc9b5161a	SHA1
51379e74f85ede610cdc5aaf250fee4cdac5e3b0	SHA1
553a38610bb554aac55aa6d00d926470d8c82698	SHA1
6ce0a07fdd4a6a774a7e3eae6f97f49868921fe3	SHA1
4807990b68d873c78d00d2be605c1b0ac24d09ee	SHA1
d10ca5474f723d83bbf3b3307d58c545d2be5dfc	SHA1
e212fa4384420c18beec83c3f1c8259481a63efa	SHA1
a2f9959767b6696e85f0aabae87632f539717884	SHA1
a8e60df51c30106a7d1b0170cbb0a9ca7e167ca7	SHA1
e45f89c923d0361ce8f9c64a63031860a76b2d10	SHA1
1e784e2f800ba32edee3159c03616c70fc68dc5b	SHA1
efda5178286678794b40987e66e686ce	MD5
3a9cdd8a5cbc3ab10ad64c4bb641b41f	MD5
969310a9775070c314377a9a4a665686	MD5
6fd56f2df05a77bdfd3265a4d1f2abac	MD5
98fe909510c79b21e740fec32fb6b1a0	MD5
01d322dcac438d2bb6bce2bae8d613cb	MD5
30247645638ff6d314c83044c831cdc4	MD5
e58b8de07372b9913ca2fbd3b103bb8f	MD5
429be60f0e444f4d9ba1255e88093721	MD5
a9485f3ecf7f35ba16a680a03d17c9ee	MD5
11796e9e5567954ffe6eb9049f29acb2	MD5

dc146f77caaaea3deae053d9dc5a82d2	MD5
18427cdcb5729a194954f0a6b5c0835a	MD5
ee6420f6bccd3eb9510211c020129c0c	MD5

Contact us

Form

Hide