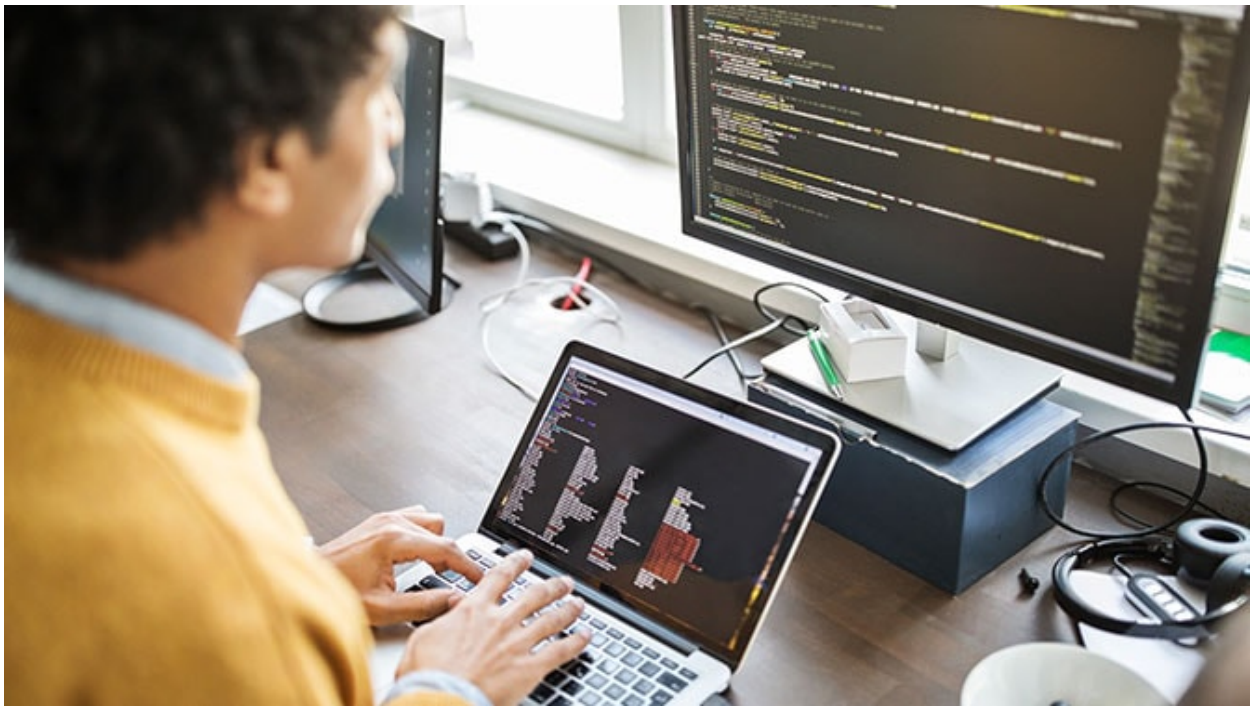
WellMess malware: analysis of its Command and Control (C2) server

pwc.co.uk/issues/cyber-security-services/insights/wellmess-analysis-command-control.html



Copy link

17/08/20

In July, the National Cyber Security Centre reported that threat actors had been using malware known as WellMess to target organisations involved in COVID-19 vaccine development.

We previously published analysis of the WellMess malware, including details of its capability and functionality. During our investigations, we identified another binary from shared Program Database (PDB) paths, which is highly likely a first stage Command and Control (C2) server used to pass data to a second stage C2. This analysis details the capabilities and limitations of the first stage WellMess controller, with a more in-depth report available from our private intelligence reporting services.

WellMess C2 analysis

While investigating the WellMess malware family, we identified that most samples contained PDB paths beginning with:

```
C:/Server/BotUI/App_Data/Temp/
```

Pivoting on this string returned an additional file that was not in our previous reporting or the July 2020 NCSC advisory on WellMess.

Filename	mwdg.sh
Filetype	ELF 64-bit
SHA-256	b10fa150e9f022838347115d39fa672440c740a014913947b79464b68dcc2d55
First seen	2020-05-04 03:20:12
Go version	1.8

This file contains functionality to communicate with the WellMess malware via HTTP and HTTPS communication methods, receiving the initial beacons and then handling and responding to the rc command detailed in our previous article. This file also contains code to handle additional commands that have previously not been seen in the WellMess malware and are likely commands sent from infrastructure controlled by the threat actor.

We assess this file is likely to be used as an intermediate C2 server that the WellMess malware communicates with before having data passed on to a second stage C2. It is likely that there are different variants of this first stage C2 software that also support the DNS protocol of WellMess.

WellMess server

The WellMess server is written in Go and imports two packages, lumberjack and garhttp, that are not part of the standard Go packages. The lumberjack package is an open source logging module, which is available on github. The garhttp package is not available in open source and contains the functionality used to communicate between the server and WellMess binaries.

The WellMess server uses the lumberjack package to create a development.log file in the current directory. This file is used to store information about all the connections made to the C2 and what ports and actions were taken. In addition, if the server encounters any errors then the log file will contain the Go error type and related call history that led to it. An example log file with errors is shown in Figure 1.

Figure 1 - Error log contents

The server uses folders in the current directory to store information sent and received from WellMess backdoors and the folder layout is shown in Figure 2. Additionally, the server uses a private key and certificate located in the current working directory during mutual TLS connections.

Figure 2 - Server directory contents

Based on strings in the binary, the folder names likely stand for Answers, Garbage, Questions and Secure. When the server receives communications from a WellMess backdoor in HTTP mode it will use the A folder to store the data received from the malware. It uses the Q folder to store questions

or queries that it is ready to send to the WellMess malware and optionally moves the files from the A and Q folders to the G folder once the data has been sent.

The S folder contains another set of the A, G and Q folders that are used when the server receives and sends data via the HTTPS communication protocol.

When receiving HTTP commands, the WellMess server is setup to receive POST requests that contain RC6 encrypted cookies. The server decrypts the cookies using a hardcoded RC6 key and expects the decrypted data to contain no more than four tags. As the RC6 key is hardcoded and the server does not have any functionality to modify it at runtime, it is unlikely that the threat actor behind WellMess would change the WellMess malware's RC6 key after initial access as that would also require changing the server software.

The four tags correspond with the format previously reported on in the WellMess backdoor with possible tag names being:

- head;
- title;
- service; and,
- body.

If there are three or four received cookie tags and they all match the regular expression below, then the title tag is checked to see what type of communication is being received with the options detailed in Table 1.

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

Table 1 - Title tag details

Tag regular expression	Description
a:\d{1,}_\d{1,}	Saves the received data into the A folder as a .tmp file
q:\d{1,}_\d{1,}	Saves the received data into the Q folder as a .tmp file
rc	Checks the Q folder for .tmp files to respond with
rcc	Checks a folder specified in the service tag for .tmp files to respond with
rs	Checks the A folder for .tmp files to respond with
rsc	Checks a folder specified in the service tag for .tmp files to respond with

del	Deletes a file specified by a relative or absolute filepath in the service tag

The WellMess backdoor supports the a:\d{1,}_\d{1,} and rc commands, as the initial beacons use the title tags a:1_0 and a:1_1 and the request for a command from the server is sent using the rc command. The other values do not appear to be used by the WellMess backdoor and, based on the functionality, are likely used to manage the operation of this intermediate C2 server.

When the server receives a a:\d{1,}_\d{1,} tag, it creates a folder with the path (relative or absolute) in the head tag. It stores the received information in a file called tmp1_0.tmp with the numbers in the filename being taken from the numbers in the title tag.

As an example, if the initial beacon had a cookie of:

```
MD5_hash/pa:1_0p
```

Then the title tag would be used to name the file as tmp1_0.tmp and the head tag of MD5_hash/p would be used to create the directories to store the received data.

The information stored in this file contains newline separated information of the RC6 encrypted and base64 encoded cookie, the IP address and port combination that the POST request came from, the MD5 hash of the received encrypted POST data and the received encrypted POST data. An example is shown in Figure 3.

Figure 3 – Stored initial beacon

On receipt of a message with rc in the title tag it will check the Q folder to see if there is a folder with the same name as the hash that is in the head tag of the rc message. If there is, then it expects there to be a subfolder which contains an arbitrary amount of *.tmp files.

The *.tmp files are expected to have contents with newline separated data of:

- A base64 encrypted cookie; and,
- An obfuscated string to be sent as the content of the 200 OK response.

The obfuscated string has the same format of character replacement and splitting into space separated chunks as described in our previous reporting[5] and is either random data or AES encrypted data depending on the command specified in the cookie.

If there is only one *.tmp file in the Q subfolder, then the server will send it with the command specified in the encrypted first line as the Set-Cookie content and the data in the second line to be sent as the body of the POST request. If there are multiple files in the subfolder, then the server will instead send the files as part of a chunked C command.

The rs command checks the A folder to see if there are any directories containing .tmp files and if it finds any then sends all the .tmp files in the first folder found. It uses the first line of the found file as the Set-Cookie data as in the rc command but also RC6 encrypts and obfuscates the file path of the

file that is being sent and puts it at the start of the content of the POST request, along with the remaining lines of the file to be sent.

The rsc and rcc commands do the same operations as the rs and rc commands respectively, but they also allow the service tag to specify the file path that is to be used to read data from.

The del command takes a file path in the service tag as the location on the server of a file to delete. If after deleting the file, the folder it was in is empty, then the folder is also deleted.

In HTTPS mode the server has a hardcoded Certificate Authority (CA) certificate that it uses to verify any certificates that connect to the server. In practice, the certificates of both the server and anyone communicating with it will need to be signed by the same CA. This is due to the fact that the embedded CA certificate in this sample has details, described in Table 2, that match the CA certificate details used in the WellMess backdoors.

Table 2 - CA details

SHA256 Fingerprint	163C405D87B632EA4E3D79210EAB2417C042CC3B2D8989A1D8ADF2B9A0F39CAF
Issuer	*, IT, Tunis
Subject	*, IT, Tunis
Thumbprint	E80641E4B2661873CBEEDDE46CCDF04A720529C9

When receiving HTTPS commands, the server has a subset of commands available to it. The rcc and rsc commands are not supported and the commands for sending files to be stored in the Q and A folders have tags of just q and a rather than the regular expression as in the HTTP commands.

When data is saved into the A and Q folders in HTTPS mode, the server creates two files – fileBody.tmp and filemd.tmp – rather than one. The fileBody.tmp file has the contents of the POST request placed into it verbatim and the filemd.tmp file contains the encoded cookie and IP/port combination of the backdoor the message came from.

The rs command also has slightly different functionality, as it moves the first pair of files in the A folder into the G folder and then sends the data back to the requestor. The response from the server contains two Set-Cookie headers with the first having a name of Cookie1 and the second having a name of Cookie2. The first cookie contains a base64 encoded string of the command received in the POST request for this file and the second cookie contains the metadata of where the file is located in the server, along with an MD5 hash of the POST request contents and the IP and port that the request came from.

Multiple rs commands will continuously return the data in the G folder if present and only return new files if the G folder has had its contents removed.

Security considerations

The threat actor behind this tool has made choices during development that increase the security of this server software.

The inclusion of mutual TLS in HTTPS communications with a self-signed CA means that there is no opportunity to carry out person-in-the-middle attacks on traffic between the server and the WellMess backdoor, as the server and WellMess backdoor would drop any connections that did not provide a correctly signed certificate.

In HTTP mode, the server software has no capability to decrypt or encrypt any of the RSA and AES encrypted data that is sent from the WellMess backdoor and so anyone that recovers files from this server without the corresponding encryption keys would not be able to decrypt the contents of the communications between the WellMess backdoor and this server.

Attribution

The NCSC has publicly attributed WellMess to the threat actor we track as Blue Kitsune (a.k.a. APT29). Although we cannot definitively tie the WellMess malware to a particular threat actor based on current information, the WellMess backdoor does share some design similarities with a previous Blue Kitsune tool called Seaduke.[1][2]

The operation of this WellMess server as an intermediate C2 fits the Seaduke model of using compromised third-party websites or infrastructure to host C2 servers that act as a staging environment to store commands and responses between the threat actor and the backdoor. The implementation details of Seaduke also have some similarities to WellMess, as both use encrypted cookies to transfer metadata about the data being sent and use obfuscated base64 data in HTTP requests as the contents of communications. These techniques are not unique to Blue Kitsune but provide an interesting correlation between the WellMess backdoor and Blue Kitsune tools used since 2015.

Conclusion

The WellMess backdoor, along with this C2 software, enables the threat actor to interact with victims in an offline manner and with a level of abstraction between the threat actor and the malicious software. The staging C2 detailed in this report can continuously collect information about any victims and store the results ready for the threat actor to review at a later date. Similarly, should any victims go offline for a period of time, then this C2 software can store the commands that it wishes to run until the victim comes back online.

The C2 has limited functionality to relay information between itself, the WellMess backdoor and presumably a further threat actor-controlled machine. It lacks the ability to inspect any of the content that is passed through it and relies on either some other local tool or remote commands to manage the transfer of files.