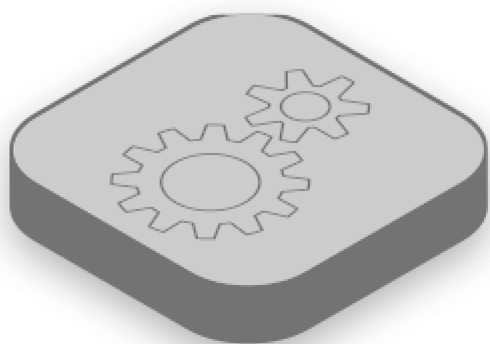
IcedID Banking Trojan Uses COVID-19 Pandemic to Lure New Victims

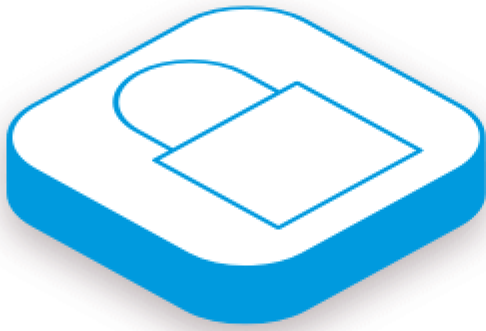
March 4, 2021

App Tiers Affected:

Client

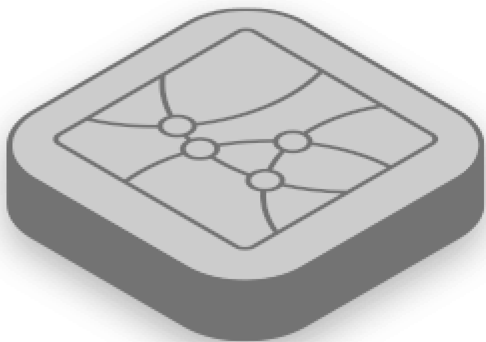


Services

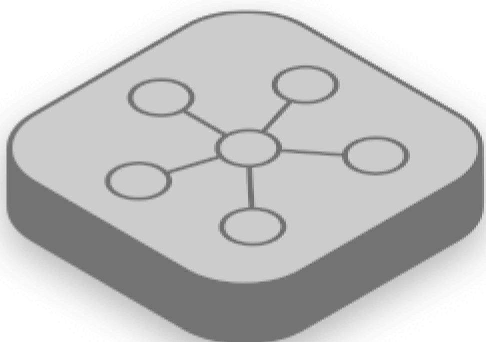


Access

TLS



DNS



Network

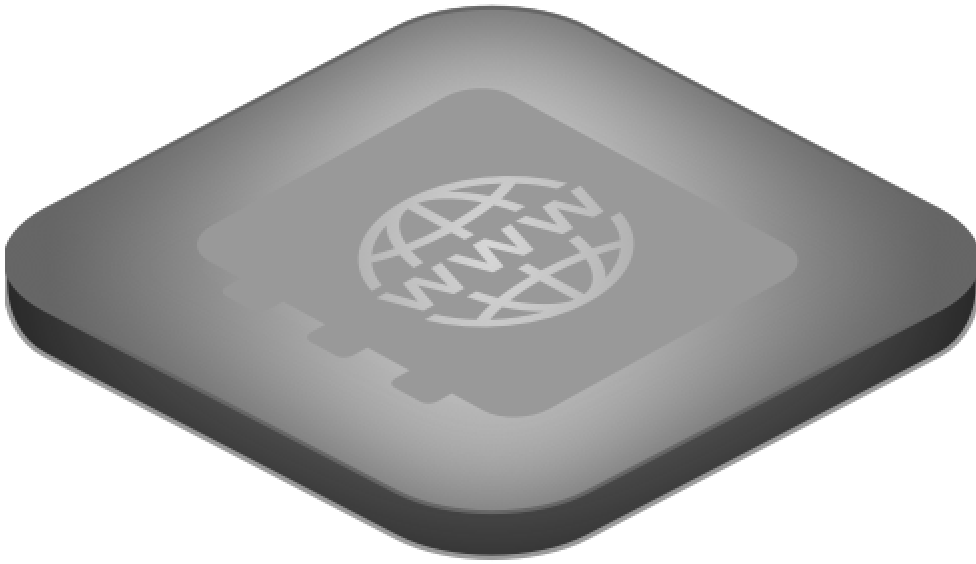
App Tiers Affected:

Client

Services

Access

TLS



DNS

Network

The IcedID malware, also known as Bokbot, is a banking trojan first discovered in 2017 that steals credentials by tricking browser functions into redirecting traffic. It is a stealthy, fileless malware with anti-sandbox capabilities. Previously, F5 Labs analyzed IcedID decompression methods for web injecting relevant files into a target list. This is a much deeper attack chain analysis of IcedID and its techniques.

Stage 1: IcedID is Distributed through Microsoft Word Document Email Attachments

In recent attacks, IcedID has been deployed as part of the TA551, or Shathak, email-based malware distribution campaign, often targeting English-speaking victims. The campaign uses lures tied to the COVID-19 pandemic to trick users into opening malicious attachments. Over the past year, F5 Labs has seen that the majority of phishing and fraud attacks have been cloaked in pandemic-related lures.

In the current campaign, IcedID rides in on Microsoft Word documents with a poisoned macro that inserts an installer to install the malware, which is designed to steal users' credentials, payment card data and other sensitive information from major financial institutions and retailers.

Stage 2: IcedID is Installed and Injected

The malicious Microsoft Word macro download and executes the installer which relocates itself to %APPDATA%\Local\{user}\ or %APPDATA%\Local\{GUID}\ and sets a scheduled task to run every hour or user logon for persistency. Later on the installer tries to download a PNG image from several command-and-control (C&C) domains.

Inside the PNG is an IcedID loader encrypted with RC4 hidden as a legitimate PNG file, a technique called steganography to hide itself from security solutions such as antivirus, malware-detecting sandboxes, and static analysis tools.

The installer transfers control to the RC4-decrypted shellcode that injects itself into other processes by creating a suspended process, writing the shellcode to the process's memory, setting an asynchronous procedure call (APC) thread to transfer control to the shellcode, and lastly calling NtResumeThread to start the injection.

The injected process is usually msiexec.exe or svchost.exe, as shown in Figure 1. Both are digitally signed by a well-known Microsoft certificate, making it harder to detect.

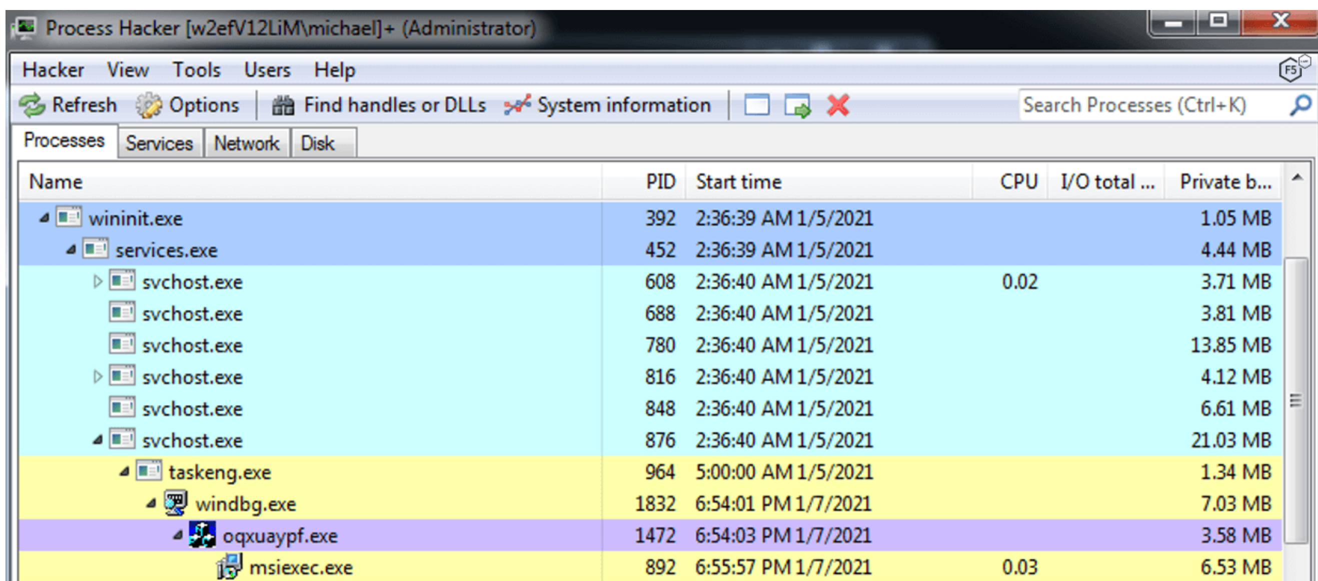


Figure 1. IcedID process tree in runtime.

IcedID APC Injection

IcedID uses part of the process hollowing technique to inject malicious code into suspended processes and to evade process-based defenses. Figure 2 shows this technique's specific code snippet for IcedID. The suspended legitimate process is either svchost.exe or msiexec.exe, which are known and signed Microsoft processes.

```
loc_70280:                              ; CODE XREF: sub_7009C+1D5↑j
                cmp     esi, 0AB753AC9h
                jnz     short loc_70292
                mov     eax, [edi+eax*4]
                add     eax, ecx
                mov     [ebp+ZwAllocateVirtualMemory], eax
                jmp     short loc_7030E
; ---------------------------------------------------------------------------

loc_70292:                              ; CODE XREF: sub_7009C+1EA↑j
                cmp     esi, 2AA75732h
                jnz     short loc_702A4
                mov     eax, [edi+eax*4]
                add     eax, ecx
                mov     [ebp+ZwQueueApcThread], eax
                jmp     short loc_7030E
; ---------------------------------------------------------------------------

loc_702A4:                              ; CODE XREF: sub_7009C+1FC↑j
                cmp     esi, 0BD5E7CB6h
                jnz     short loc_702B6
                mov     eax, [edi+eax*4]
                add     eax, ecx
                mov     [ebp+ZwWriteVirtualMemory], eax
                jmp     short loc_7030E
; ---------------------------------------------------------------------------

loc_702B6:                              ; CODE XREF: sub_7009C+20E↑j
                cmp     esi, 0BD04B6BCh
                jnz     short loc_702C8
                mov     eax, [edi+eax*4]
                add     eax, ecx
                mov     [ebp+NtResumeThread], eax
                jmp     short loc_7030E
```
Figure 2. Code injection technique and its functions.

To use this technique, the malware does the following:

1. Creates a process using *CreateProcessA* with a SUSPENDED flag,
2. Allocates memory using *NtAllocateVirtualMemory API*,
3. Writes the shellcode to memory using ZwWriteVirtualMemory API,
4. Changes the memory protections using *NtProtectVirtualMemory*,
5. Creates an APC thread routed to the shellcode
6. With the process still in suspended mode, the technique's last step is to call NtResumeThread API to resume the process.

Once this is done, the shellcode walks through the process environment block (PEB) structure, comparing a hash against function names retrieved in order to create dynamic resolving (Figure 3 and Figure 4).

```
current_ordinal = *(et_address_of_name_ordinals + counter);
pFunction_name = dll_base + *(addr_of_names_rva + 4 * counter);
sum_ror = 0;
function_name = *pFunction_name;
current_ordinal_rva = current_ordinal;
if ( function_name )
{
  pFunction_name_2 = pFunction_name;
  letter_of_function_name = function_name;
  do
  {
    sum_ror = letter_of_function_name + __ROR4__(sum_ror, 13);
    letter_of_function_name = *++pFunction_name_2;
  }
  while ( *pFunction_name_2 );
  current_ordinal_rva = current_ordinal;
  v2 = 0;
}
function_hash = sum_ror ^ 0x784EF074;
switch ( function_hash )
{
  case 0xC8D67F90:
    LdrLoadDll = (dll_base + *&et_address_of_functions[4 * current_ordinal_rva]);
    break;
  case 0x9D023473:
    LdrGetProcedureAddress = (dll_base + *&et_address_of_functions[4 * current_ordinal_rva]);
    break;
  case 0xAB753AC9:
    ZwAllocateVirtualMemory = (dll_base + *&et_address_of_functions[4 * current_ordinal_rva]);
    break;
  case 0x2AA75732:
    ZwQueueApcThread = (dll_base + *&et_address_of_functions[4 * current_ordinal_rva]);
    break;
  case 0xBD5E7CB6:
    ZwWriteVirtualMemory = (dll_base + *&et_address_of_functions[4 * current_ordinal_rva]);
    break;
```

Figure 3. Code snippet: IcedID scans the process environment block to find modules by encrypted hash.

```
if ( !a1 )
  return 0;
pPEB = __readfsdword(0x30u);
v2 = 0;
if ( !pPEB )
  return 0;
v3 = pPEB->Ldr->InInitializationOrderModuleList.Flink;
if ( !v3 )
  return 0;
ldr_data_entry = &v3[-2];
if ( !ldr_data_entry )
  return 0;
dll_name = ldr_data_entry->FullDllName.Buffer;
dll_name_first_char = dll_name ? LOBYTE(dll_name->Flink) : 'C';
dll_base = ldr_data_entry->DllBase;
if ( !dll_base )
  return 0;
if ( dll_base->e_magic != 'ZM' )
  return 0;
dll_nt_headers = (dll_base + dll_base->e_lfanew);
if ( dll_nt_headers < dll_base )
  return 0;
if ( dll_nt_headers >= &dll_base[64] )
  return 0;
if ( dll_nt_headers->Signature != 'EP' )
  return 0;
export_table_rva = dll_nt_headers->OptionalHeader.DataDirectory[0].VirtualAddress;
if ( !export_table_rva )
  return 0;
if ( (dll_base + export_table_rva) < dll_base )
  return 0;
```

Figure 4. Second code snippet: IcedID scans the process environment block to find modules by encrypted hash.

To make things harder for antimalware controls and security researchers, this technique dynamically creates an import table with no strings associated with function names in memory. It does so by making use of the PEB, a data structure within the process' memory to hold information about the current processes. IcedID scans through the PEB, enumerating all the module function names, find the needed functions and store them in variables for later use.

F5 Labs Newsletter

- One email per week, with newsletter exclusives
- Latest security research insights
- CISO-level expert analysis

F5 Labs Newsletter

The information you provide will be treated in accordance with the F5 Privacy Notice.

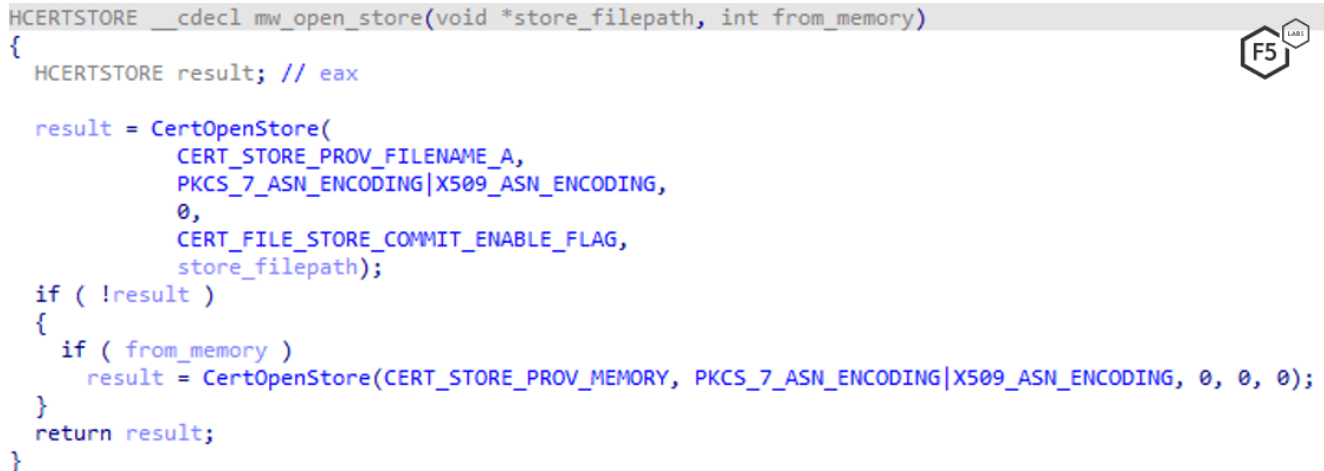
Great! You should receive your first email shortly.

Stage 3: IcedID Sneaks through the System

With the malware fully running in the infected system, it now seeks out targets within the system to ensure it's in a good place to steal credentials. However, it also needs to continue to resist analysis and evade detection by antivirus software.

IcedID Certificate Store

IcedID creates a *tmp* file inside the %TEMP% folder (Figure 5), which contains a certificate store used to save all self-signed certificates the malware has generated so it won't have to regenerate the certificate for every website.



```
HCERTSTORE __cdecl mw_open_store(void *store_filepath, int from_memory)
{
  HCERTSTORE result; // eax

  result = CertOpenStore(
             CERT_STORE_PROV_FILENAME_A,
             PKCS_7_ASN_ENCODING|X509_ASN_ENCODING,
             0,
             CERT_FILE_STORE_COMMIT_ENABLE_FLAG,
             store_filepath);
  if ( !result )
  {
    if ( from_memory )
      result = CertOpenStore(CERT_STORE_PROV_MEMORY, PKCS_7_ASN_ENCODING|X509_ASN_ENCODING, 0, 0, 0);
  }
  return result;
}
```

Figure 5. IcedID generates code to create a certificate store.

IcedID String Evasion Techniques

IcedID uses string encryption to avoid being detected. String obfuscation is an evasion technique that malware uses to hide malicious activity as well as to make static analysis harder for researchers and automations. A common way to hide strings is to create a hash and compare against a hard coded precalculated hash. IcedID uses this method to find the browser's file name without using the associated string, which could be detected within the malware, as shown in Figure 6.

```
file_hash = 0;
filename_lower = PathFindFileNameA(pszPath);
filename = filename_lower;
if ( !filename_lower )
  return 0;
CharLowerA(filename_lower);
v5 = *filename;
i = 0;
if ( *filename )
{
  current_chr = *filename;
  do
  {
    hash_transform_1 = __ROR4__(file_hash + i + current_chr, 3);
    ++i;
    file_hash = hash_transform_1;
    current_chr = filename[i];
  }
  while ( current_chr );
}
hash_transform_2 = file_hash ^ 0x784EF074;
if ( hash_transform_2 != 0x66A2381F )
{
  switch ( hash_transform_2 )
  {
    case 0x67C9783B:
      return 4;
    case 0x827AA2CF:
      return 1;
    case 0xAB14D0E5:
      return 3;
  }
```

Figure 6. IcedID uses string encryption to create a hash and avoid being detected.

Another evasion technique that IcedID uses is to encrypt the malicious strings beforehand and only decrypt them in memory, as shown in Figure 7.

```
BYTE *__cdecl DecodeString(BYTE *encoded, BYTE *out)
{
  unsigned int v3; // [esp+0h] [ebp-Ch]
  unsigned __int16 v4; // [esp+4h] [ebp-8h]
  unsigned __int16 i; // [esp+8h] [ebp-4h]
  BYTE *v6; // [esp+14h] [ebp+8h]

  v3 = *encoded;
  v4 = *encoded ^ *(encoded + 2);
  v6 = encoded + 6;
  for ( i = 0; i < v4; ++i )
  {
    v3 = i + ((v3 << 25) | (v3 >> 7));
    out[i] = v3 ^ v6[i];
  }
  return out;
}
```

Figure 7. IcedID technique that encrypts malicious strings and decrypts them in memory.

Finally, another common IcedID technique is to hide strings within the stacks. It breaks down the string one character at a time and saves it inside a variable, which is put onto the stack, as shown in Figure 8.

```
  if ( v5 == 'c' )
    return filename[1] == 'h' && filename[2] == 'r' && filename[3] == 'o' && filename[4] == 'm' && filename[5] == 'e';
  if ( v5 != 'f'
    || filename[1] != 'i'
    || filename[2] != 'r'
    || filename[3] != 'e'
    || filename[4] != 'f'
    || filename[5] != 'o'
    || filename[6] != 'x' )
  {
    return 0;
  }
```

Figure 8. IcedID hiding strings in the stacks.

IcedID Anti-debugging and Anti-Sandboxing Techniques

To hide itself from virtual machines, often used to debug or detect malware, IcedID needs to determine if it is running in a sandbox. It does this by first calculating the timing execution of the *cpuid* instruction by using the read time-stamp counter (RDTSC) to count the number of CPU cycles since reset. It uses *SwitchToThread* in this function to measure with RDTSC without the context switch fluctuations, as shown in Figure 9. Note that the call to *SwitchToThread* happens prior to the RDTSC instruction to ensure the measurement happens in the same time slice (without a context switch in the middle of measuring). For a more reliable measure, this is done in a loop 16 times.

```
unsigned __int64 mw_rdtsc_diff()
{
  void (*vSwitchToThread)(void); // edx
  unsigned int time_delta_low_dword; // ebp
  unsigned __int64 start_time; // kr08_8
  unsigned __int64 time_delta; // kr10_8
  bool loop_count_eq_one; // zf
  unsigned int time_delta_high_dword; // [esp+10h] [ebp-28h]
  int loop_count; // [esp+1Ch] [ebp-1Ch]
  void (*vSwitchToThread2)(void); // [esp+20h] [ebp-18h]

  vSwitchToThread = *SwitchToThread;
  time_delta_low_dword = 0;
  loop_count = 16;
  time_delta_high_dword = 0;
  vSwitchToThread2 = *SwitchToThread;
  do
  {
    vSwitchToThread();
    start_time = __rdtsc();
    _EAX = 1;
    __asm { cpuid }
    time_delta = __rdtsc() - start_time + __PAIR64__(time_delta_high_dword, time_delta_low_dword);
    time_delta_high_dword = HIDWORD(time_delta);
    time_delta_low_dword = time_delta;
    vSwitchToThread2();
    __rdtsc();
    loop_count_eq_one = loop_count-- == 1;
    vSwitchToThread = vSwitchToThread2;
  }
  while ( !loop_count_eq_one );
  return time_delta / 0;
}
```

Figure 9. IcedID using the SwitchToThread anti-sandboxing technique.

With this information, IcedID can check the hypervisor's brand using *cpuid* with EAX=0x40000000 and turn on a bit accordingly inside the *in_vm* variable, as shown in Figure 10.

```
int mw_anti_debug()
{
  int in_vm; // esi

  in_vm = (unsigned int)mw_rdtsc_diff() > 0x14;
  _EAX = 0x40000000;
  __asm { cpuid }
  switch ( _EBX )
  {
    case 'awMV':                                // VMwa
      return in_vm | 4;                         // turn 3rd bit
    case 'VneX':                                // XenV
      return in_vm | 8;                         // turn 4th bit
    case 'rciM':                                // Micr
      return in_vm | 0x10;                      // turn 5th bit
    case 'KMVK':                                // KVMK
      return in_vm | 0x20;                      // turn 6th bit
    case 'prl ':                                //  lpr
      return in_vm | 0x40;                      // turn 7th bit
    case 'xoBV':                                // VBox
      in_vm |= 0x80u;                           // turn 8th bit
      break;
  }
  return in_vm;
}
```

Figure 10. IcedID checking the hypervisor's brand using cpuid.

IcedID Maintains Persistence

To stay persistent on the infected machine, IcedID copies itself to directory %AppData%\Roaming\%username%\ or (in some variants) also copies itself to C:\Users\%username%\AppData\Roaming\[GUID]\. It then creates a task in the Task Scheduler to run the malware upon user logon or every hour, as shown in Figure 11. This ensures IcedID remains running after reboots.

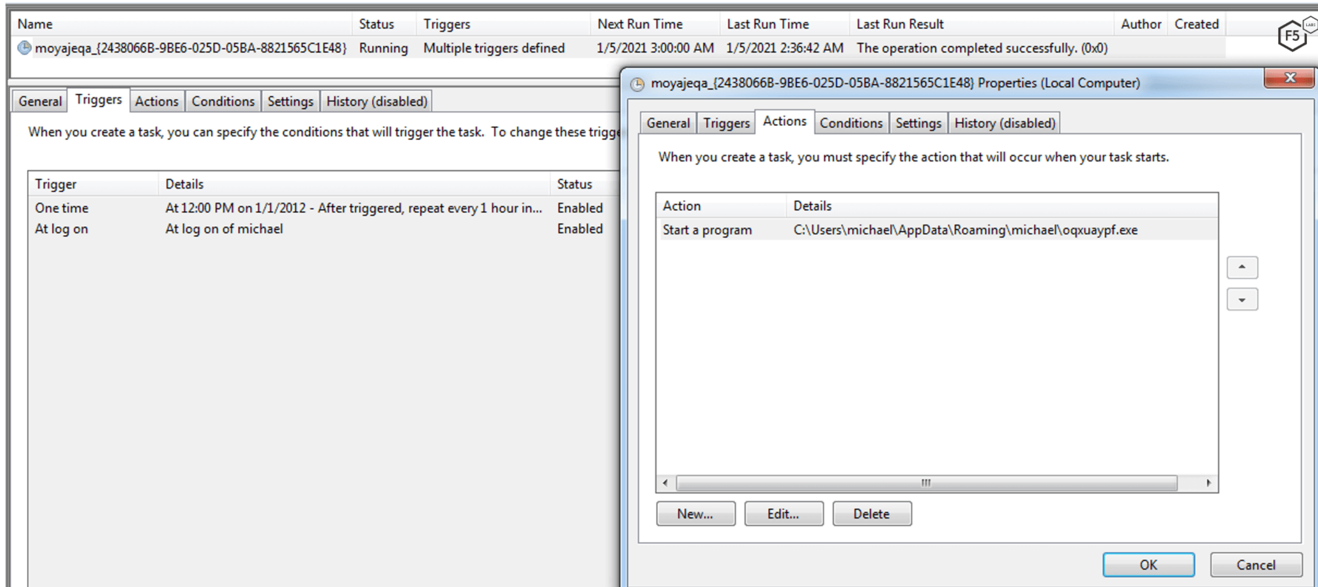


Figure 11. IcedID setting up a malicious task in the Task Scheduler.

Hashes

Installer/Runner

- 0547235018162552fcbbb67196017100
- D2275febf5f95a75a304ad2c13101f6d

PNG

cfb7a24b2f7d58d6d6dbcb91529b8020

Command-and-Control Domains

- blholove[.]co
- morganholes[.]cyou
- marmateria[.]cyou
- atombody[.]best

Installer/Runner

- MD5: 1705b9771134ce41e4d7e4d0f3b6d344
- MD5: 4fa2f9ed6756fc6e1efba2f6cf54e290

Command-and-Control Domains

- blholove.co
- morganholes.cyou
- marmateria.cyou

- atombody.best

Installer/Runner

MD5: d2275febf5f95a75a304ad2c13101f6d

Command-and-Control Domains

- fdelopoh.club
- zedebobo.top
- shmylvaro.pw
- resonanse.cyou