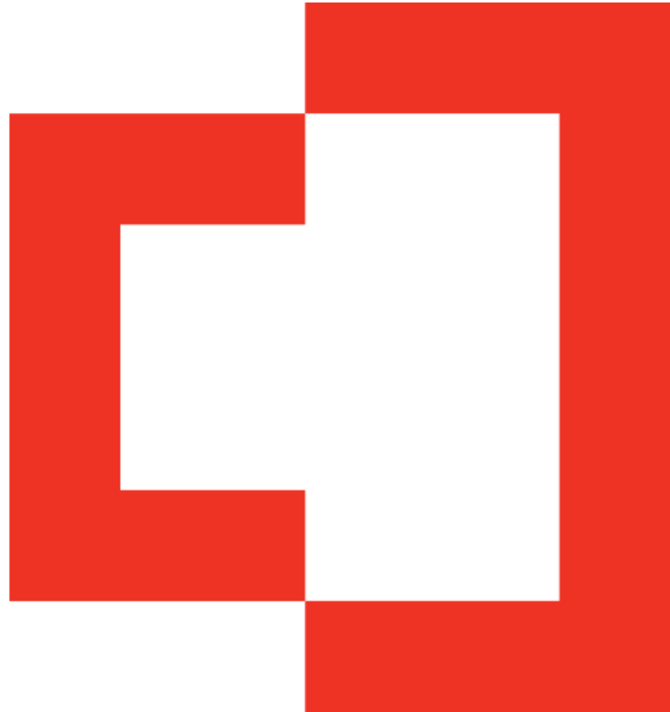
TrickBot Leverages Zoom Work from Home Interview Malspam, Heaven's Gate and… Spamhaus?

gosecure.net/blog/2021/12/03/trickbot-leverages-zoom-work-from-home-interview-malspam-heavens-gate-and-spamhaus/

GoSecure Titan Labs

December 3, 2021



The team of expert analysts at GoSecure Titan labs have reverse-engineered a new TrickBot cleverly hidden in a Zoom job interview email through a sample obtained from GoSecure Titan Inbox Detection and Response (IDR). The email message contained a shortcut (LNK) file entitled *Interview_details.lnk* and that LNK file downloads a loader which will be examined in this blog. GoSecure Titan Labs named the loader TrickGate because it uses the Heaven's Gate technique to load TrickBot, one of the world's most prevalent botnets.

Analysis

Infection Chain

The initial infection vector is via malspam. The email (906379938be59269713995cf29058f42), shown in *Figure 1*, is entitled *FINAL interview – September 3* and congratulates the user on passing an internal interview. It provides a link purporting to be zoom details for a final interview. The link downloads an LNK file (6e49d82395b641a449c85bfa37dbbbc2) from hxxps://workdrive[.]zohoexternal[.]com/file/6c8ha295582e90c3e4655b87b82bb100f011b.

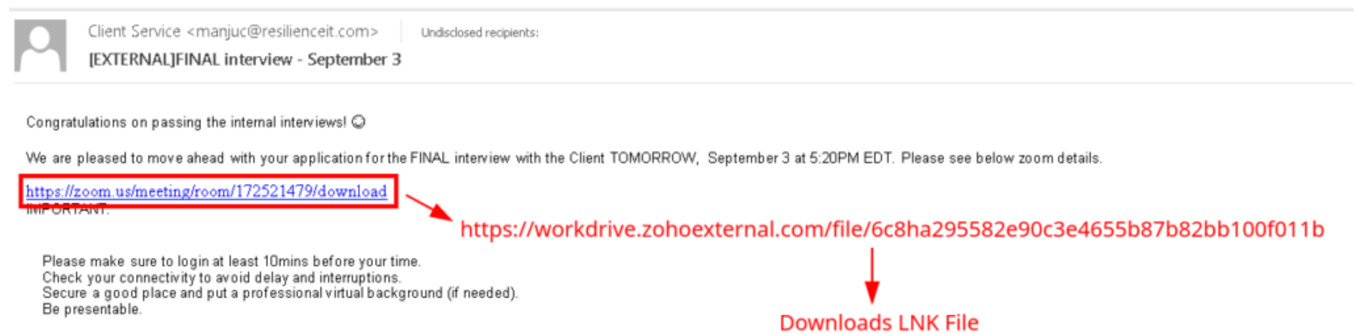


Figure 1: Zoom Interview Malspam

Once executed, the LNK file, displayed in *Figure 2*, opens Notepad as a decoy, then uses curl –silent to download TrickGate, a 32-bit C/C++ compiled portable executable (PE), from hxxp://185[.]14[.]31[.]112/images/moonfrontmars[.]png. The LNK file then saves TrickGate (442f1e3d2825d51810bf9929f46439d2) in the *%TEMP%* directory as *tmp.exe* and executes it using the start command.

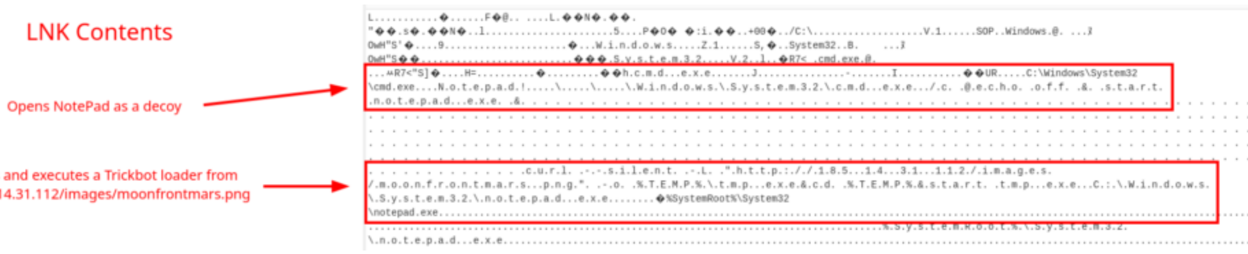


Figure 2: LNK Contents

TrickGate

In TrickGate's *.rsrc* section, the file HTML/DATA contains over 255 KB of encrypted shellcode. The shellcode is decrypted directly in the *.rsrc* section using the decryption key *planbetufernasoberpalade*. It should be noted that the decryption key varies from sample to sample. The decryption routine is depicted in *Figure 3*.

Figure 3: TrickGate's Initial Decryption Routine

Once decrypted, the shellcode (87dc309108bbf70e3e67efbf9d4c09da) is copied to memory and executed there. Besides executable code, the shellcode also contains an encrypted 64-bit portable executable. *Figure 4* shows the PE in the process of being decrypted. As can be observed from the decryption routine, the decryption simply involves XORing a byte from the decryption key with a byte from the encrypted PE. The decrypted PE (8da11d870336c1c32ba521fd62e6f55b) only contains headers and a *.text* section, which is later written to yet another section of memory.

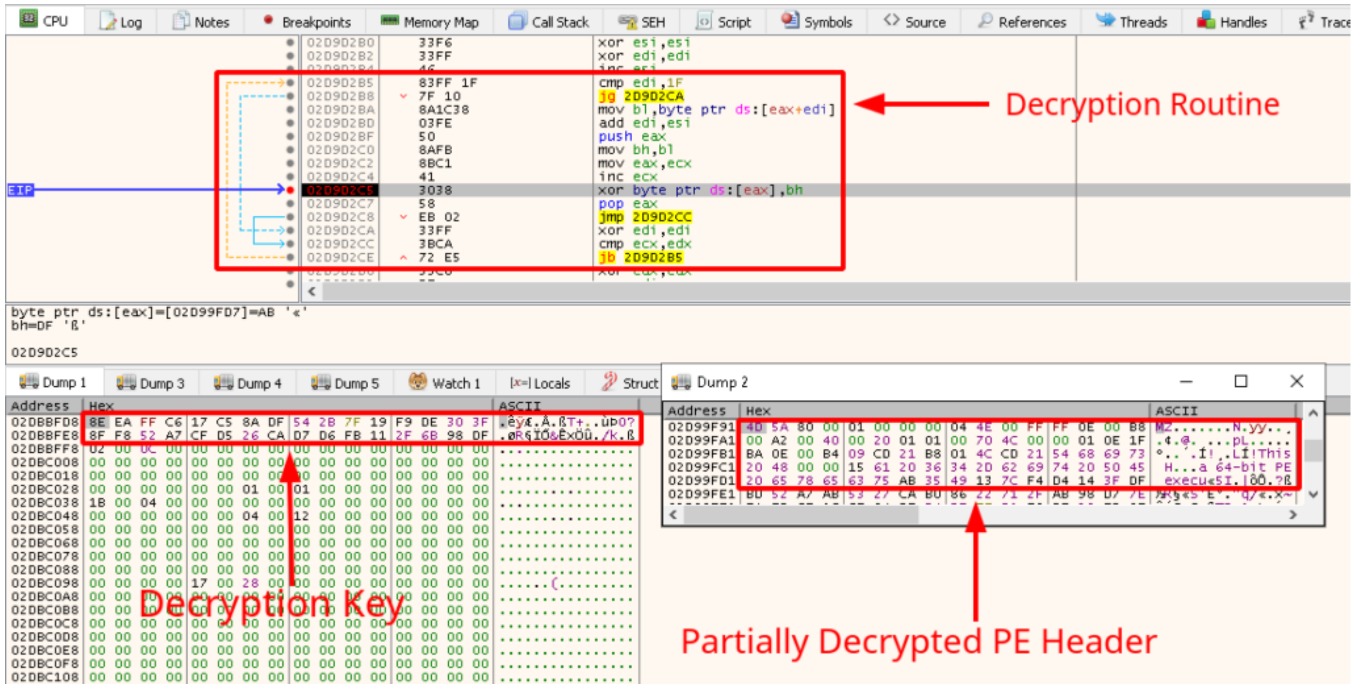


Figure 4: 64-bit PE Decryption Routine

Next, the shellcode calls *kernel32.CreateProcessInternalW*, as depicted in *Figure 5*. Since the second parameter, *lpApplicationName*, is null, the process to be created is specified by the third parameter, *lpCommandLine*, which contains a pointer to the path for Windows Error Reporting Manager (*wermgr.exe*). The seventh parameter, *dwCreationFlags*, which specifies flags that define options for the created process, contains the value *0x800000C*. This value corresponds to the flags *CREATE_NO_WINDOW*, *DETACHED_PROCESS*, and *CREATE_SUSPENDED*. Thus, *wermgr.exe* will be created in a suspended state, without a console window. This is the beginning of process hollowing, a technique used to inject and execute malware in a legitimate process.

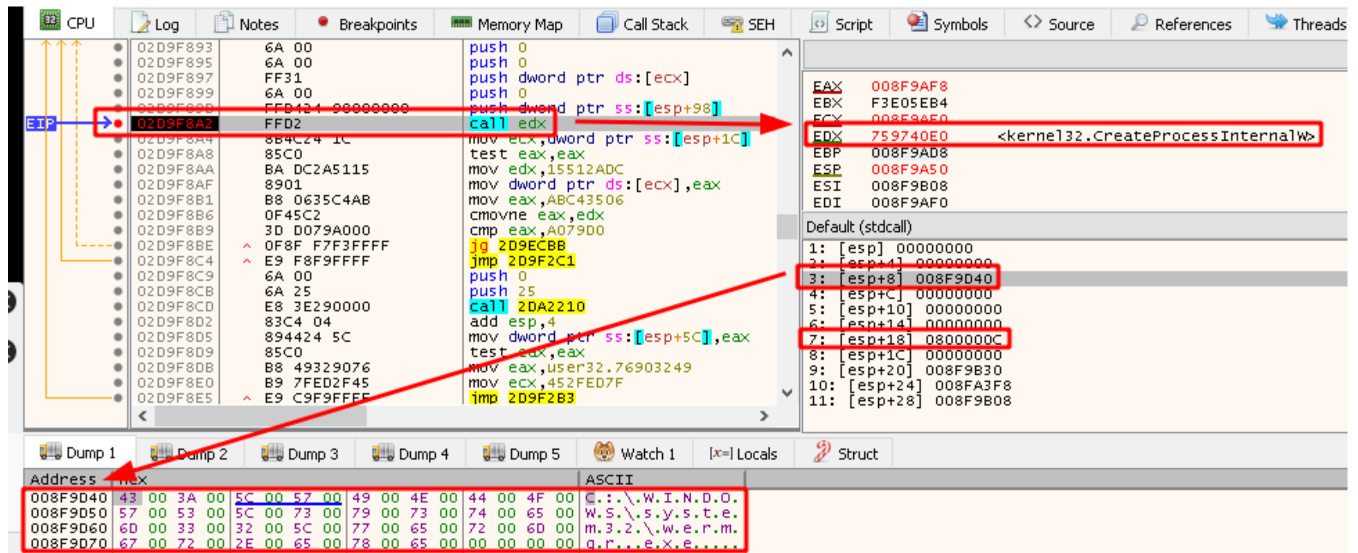


Figure 5: Create Suspended wermgr.exe Process

At this point, we expected to simply continue stepping through the disassembled code and observe the remaining process hollowing steps. However, as displayed in Figure 6, the shellcode makes a *far call* to *0x33:2F60011*, which in turn, makes a call to *0x10001000*, which is where the 64-bit code from the decrypted PE's .text section is located. Also displayed in *Figure 6*, on the left, is Process Hacker, which shows the process wermgr.exe outlined in black, signifying that it is suspended. When we try to step into the far call, instead of stepping into the instructions at address *0x2F60011*, the debugger executes for a few moments, then the instruction pointer returns to the previous function. Afterwards, *wermgr.exe* is no longer outlined in black, meaning that something has resumed the process, but we could not observe it being resumed or whether code was injected into it before it was resumed. Furthermore, setting breakpoints on API calls associated with process hollowing did not cause the debugger to pause. So, what just happened? Enter Heaven's Gate.

Figure 6: Far Call

Heaven's Gate

Heaven's Gate, first introduced in 2009, is a technique used to execute 64-bit code from a 32-bit process by using a far call, far return, or far jump. Unlike regular calls, jumps, and returns, which only specify the memory address, far ones also specify the code segment, allowing them to call, jump, or return to a different code segment. *0x23* specifies a 32-bit code segment whereas *0x33* specifies a 64-bit code segment. Thus, when *0x33* is specified with a *far call* within a 32-bit process, it switches the context of the 32-bit process to that of a 64-bit process. Since we are analyzing the sample with x32dbg, which can only analyze 32-bit code, the debugger is not capable of handling the process after it switches to 64-bit, and we only regain control of the process when it returns from the far call and reverts back to 32-bit. Most debuggers will behave in the same manner, except for WinDbg, a debugger created by Microsoft that can debug both 32-bit and 64-bit code. Using WinDbg, we can step seamlessly through Heaven's Gate and analyze the 64-bit code being executed. *Figure 7* displays the disassembly in WinDbg before and after crossing Heaven's Gate. We can see that the registers before the call pertain to a 32-bit architecture whereas 64-bit registers are being used after the call. Moreover, the code segment (CS) register now holds the value *0x33*.

Figure 7: Stepping Through Heaven's Gate

Even though WinDbg can handle the context switch, it is still confused in regard to breakpoints on API calls. This fact is illustrated in *Figure 8*. When a breakpoint is set for *ntdll.NtWriteVirtualMemory*, WinDbg sets it for the 32-bit *ntdll.dll*, as revealed by the x86 identifier and *ntdll.dll's* address, *0x77912d70*, which falls in the 32-bit address space. However, the actual version of *ntdll.NtWriteVirtualMemory* being called by TrickGate is 64-bit, as its address, *0x7ff8'570ed4a0*, lies in the 64-bit address space. Therefore, the debugger will not

pause at the requested breakpoint unless it is manually set at the appropriate address. This exemplifies just how pernicious Heaven's Gate can be. By hiding API calls, it makes malware detection and analysis very difficult. This is why Heaven's Gate was initially used by many malware authors. However, the use of Heaven's Gate has greatly declined since Microsoft introduced Control Flow Guard (CFG) in Windows 8.1. CFG places restrictions on addresses called by executing code and, as such, can mitigate Heaven's Gate. There has been some malware in recent years, such as HawkEye Reborn Keylogger and Remcos RAT, abusing Heaven's Gate to avoid detection. Publications on the topic state that malware still using Heaven's Gate does so to target legacy machines, since CFG should terminate the execution on modern systems. However, we at GoSecure Titan Labs ran TrickGate on a Windows 10 machine with CFG enabled, and it fully executed.

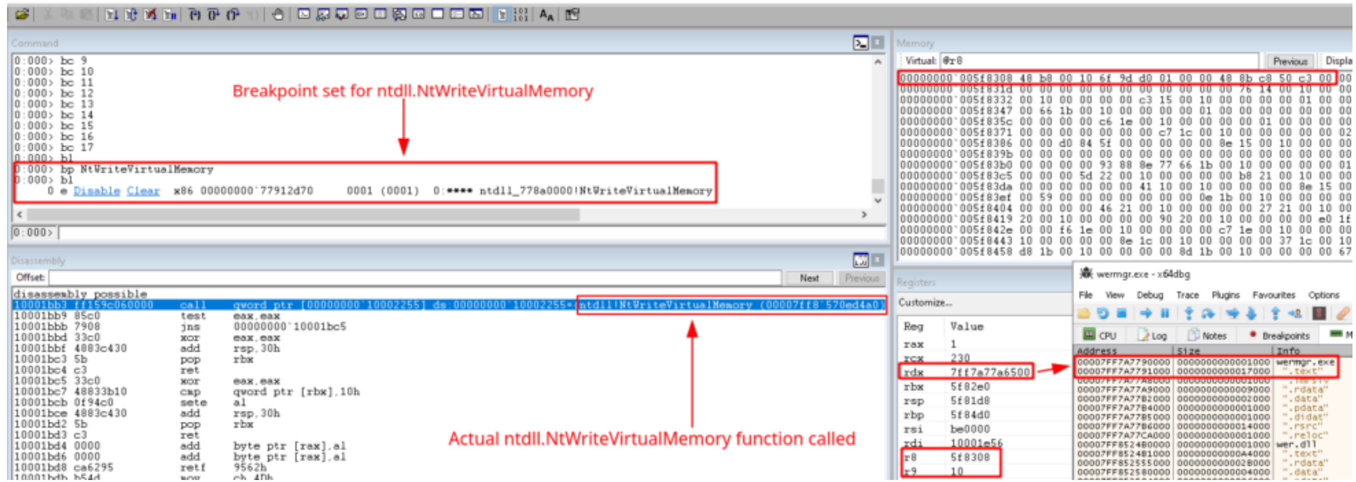


Figure 8: Call to ntdll.NtWriteVirtualMemory

As anticipated, the 64-bit shellcode in Heaven's Gate completes the process hollowing that begin in the 32-bit shellcode. Looking once again at *Figure 8*, we see that the value in rdx is *0x7ff7a77a650*. This is the second argument passed to *ntdll.NtWriteVirtualMemory* and it specifies the base address to where data should be written. Also displayed in *Figure 8* is the memory map view in x64dbg, which we had opened at this point and attached the suspended *wermgr.exe* process to. It can be seen that the base address to be written to falls within the *.text* section of *wermgr.exe*. *r8* contains an address to the buffer containing the bytes to be written and *r9* contains the number of bytes to be written, which is *0x10*, or *16* in decimal. The memory window in the top right corner displays the data stored at the address in *r8*. Therefore, the call to *ntdll.NtWriteVirtualMemory* writes the bytes 4*8 b8 00 10 6f 9d d0 01 00 00 40 0b c0 50 c3 00* to the *.text* section of *wermgr.exe*. The 64-bit shellcode then calls *ntdll.NtResumeThread* to resume the execution of *wermgr.exe*, completing the process hollowing. Before *wermgr.exe* was resumed, we placed a breakpoint on the address in *wermgr.exe* where the bytes were written. As displayed in *Figure 9*, these bytes replace the return address of the current function with the address *0x1D09D6f1000* and then returns, passing execution to that address.

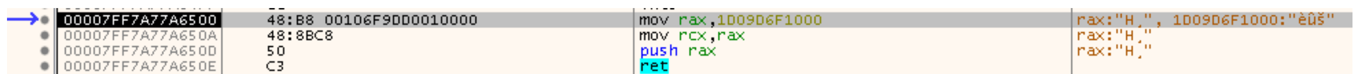


Figure 9: Injected Code in wermgr.exe

So, what exactly is stored at this address? Back in Trickgate's 64-bit shellcode, another call to *ntdll.NtWriteVirtualMemory* was made before resuming *wermgr.exe*. As can be observed from *Figure 10*, *0x28bd4* bytes, which is a little over 166 KB, was written to memory beginning at address *0x1D09D6f0000*. This written shellcode is TrickBot (8da11d870336c1c32ba521fd62e6f55b), the entry point to which is at address *0x1D09D6f1000*. Thus, TrickGate's 64-bit shellcode injected code into *wermgr.exe* so that it would execute a section of memory containing TrickBot. Therefore, TrickBot is executed disguised as Microsoft's Windows Error Reporting Manager.

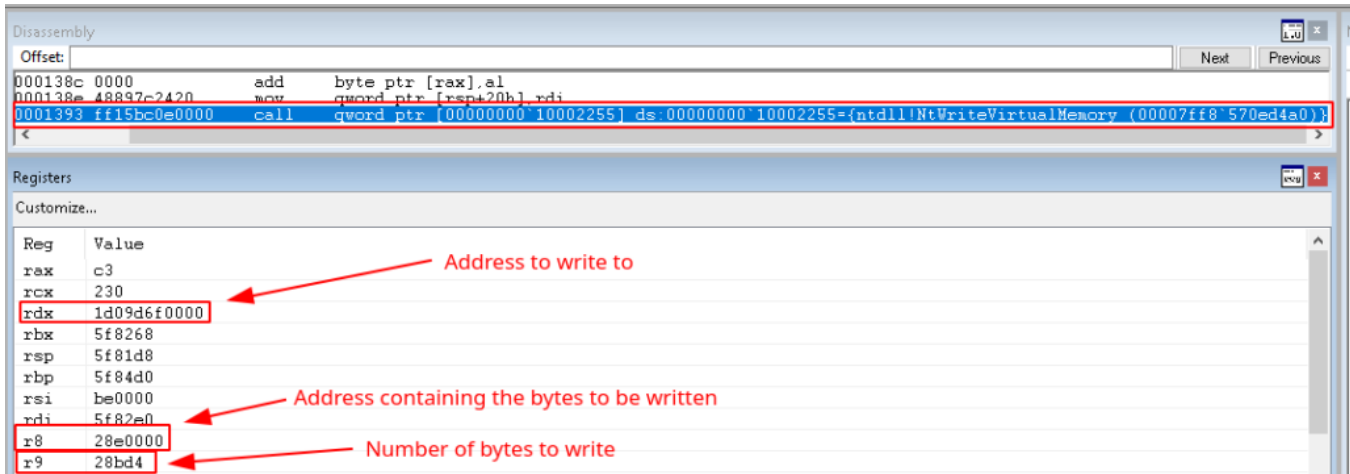


Figure 10: ntdll.NtWriteVirtualMemory Writing TrickBot To Memory

TrickBot's Latest Variant

As TrickBot is very well-known malware, discussed in many publications, we will only focus on interesting aspects of the current TrickBot variant. It creates a folder in the *C:\Users\<username>AppData\Roaming* directory. The folder's name is UniLiteGames with 4 characters appended to it, such as *UniLiteGames5UIH*. It then copies the original PE, TrickGate, and an obfuscated batch file, named *command.bat* to this folder. The batch file, shown in *Figure 11*, is obfuscated with simple string replacements. Once deobfuscated, the file contains the command start *C:\Users\<username>\AppData\Roaming\UniLiteGames<4-characters>\<trickgate-pe-name>*.

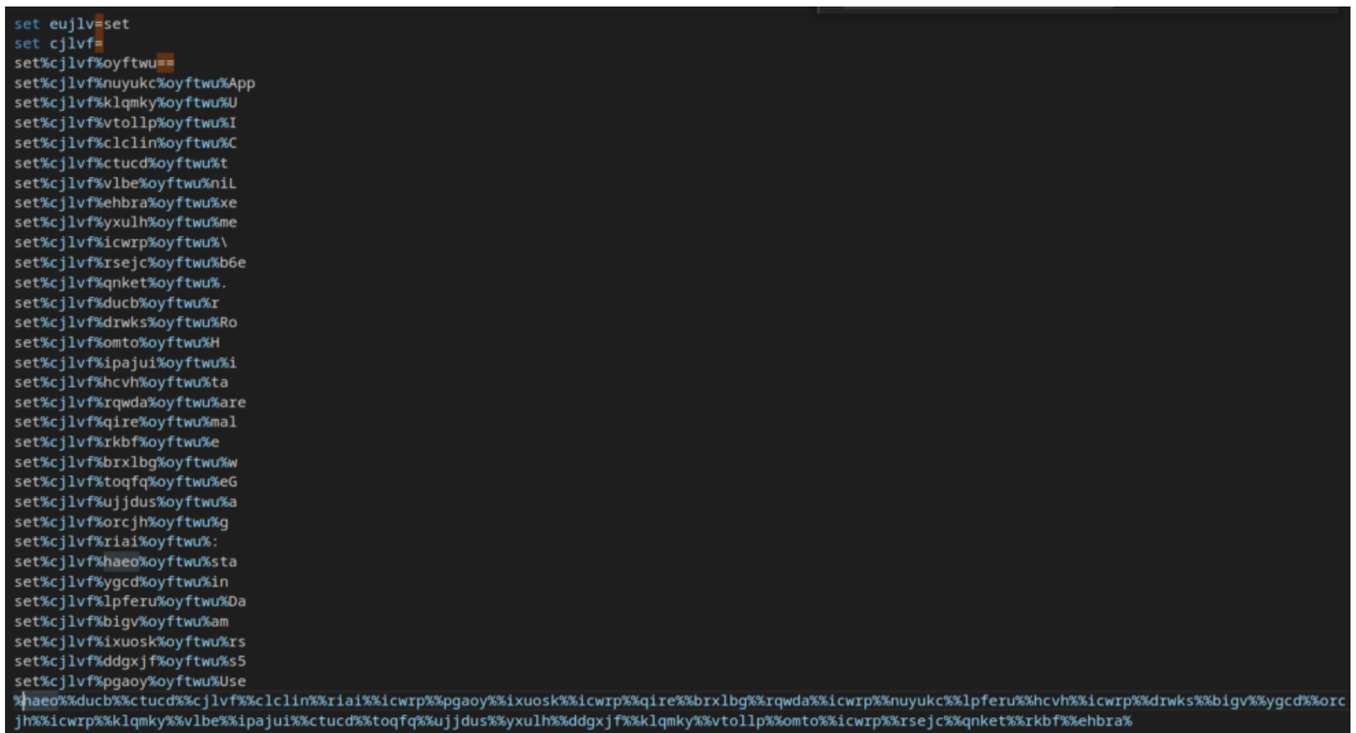


Figure 11: Obfuscated Batch File

TrickBot then creates a COM object for an interface of Task Scheduler, which it uses to create a scheduled task to run *command.bat* every time the user logs on, as depicted in *Figure 12*. The name of the scheduled task is UniGamesSoft followed by the same 4 characters used when creating the aforementioned folder, and the Author is UniGamesSoft.

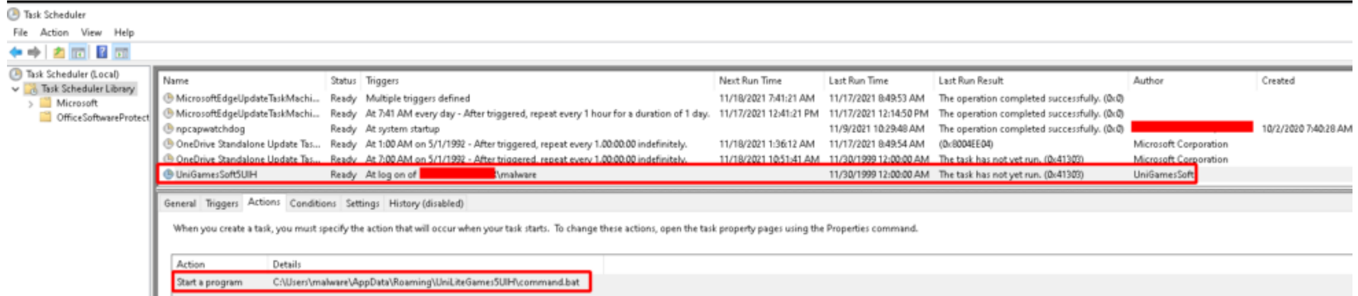


Figure 12: TrickGate Scheduled To Execute at Logon

TrickBot contains 18 command and control (C2) IP addresses, listed in the IoCs section below. All C2 communication occurs over HTTPS and uses Windows HTTP Services (WinHTTP), as can be seen in *Figure 13,* which displays the initial check-in. The third argument passed to *winhttp.HttpOpenRequest*, which creates the HTTP request handle, is */rob128/<computer_name>_W10019077.19D16C537142D197E33B9D65DF03B33E/5/file/,* which specifies the path on the target server. All following information sent to the C2 server is sent in similar GET requests. For example, information pertaining to the victim machine's network address translation (NAT) status is sent as */rob128/<computer_name>_W10019077.33A1A5DD03BBFF0FD7BA9BB14F9FBCDF/14/NAT%20status/client%20is%20behind%20NAT/0/*. As this demonstrates, the data sent to and from the C2 server is not encrypted or obfuscated in any way, presumably since TrickBot is using HTTPS to encrypt communication.

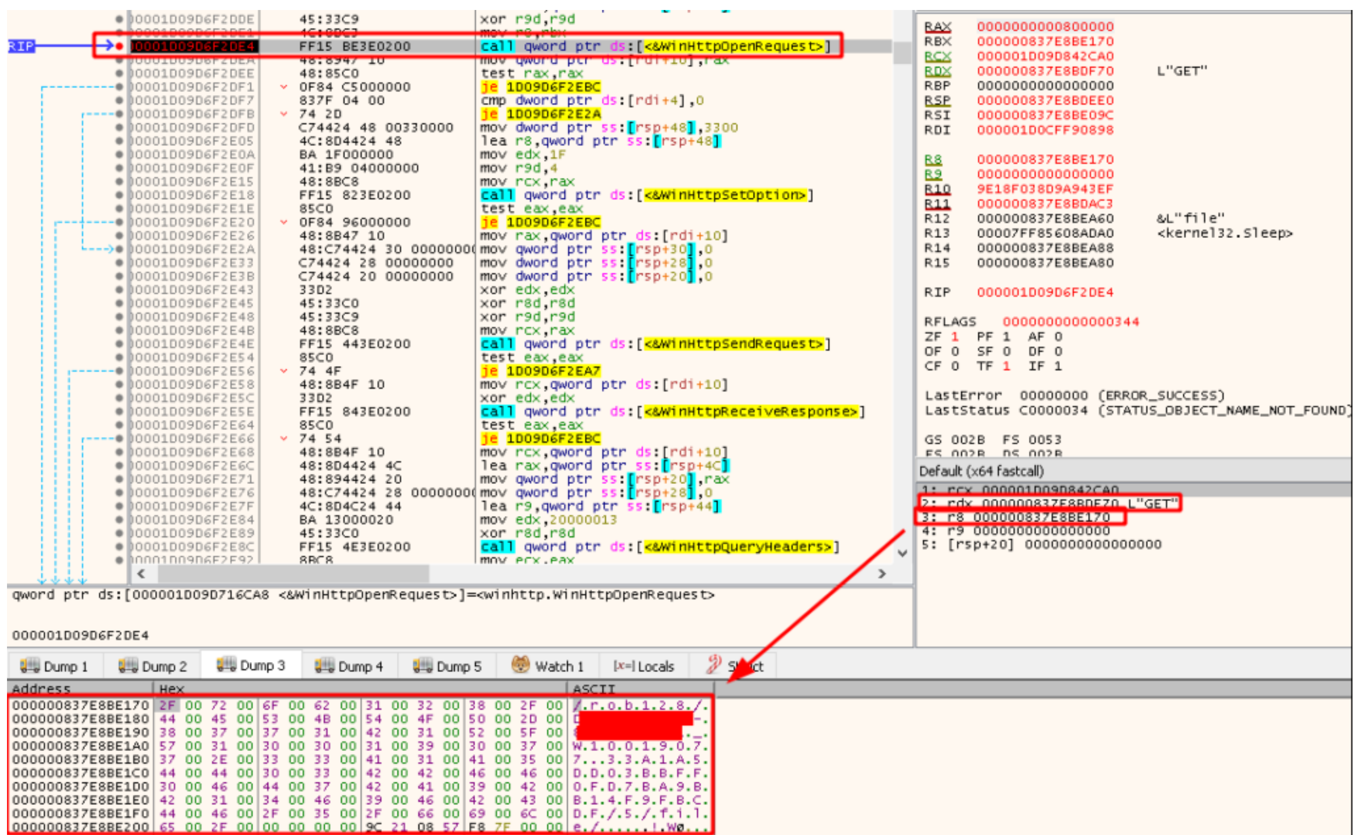


Figure 13: TrickBot's Check-in Request

The C2 URL path follows the same format observed in previous variants of TrickBot. *rob128* follows TrickBot's convention of using alphabetic characters followed by a decimal value at the beginning of the path. *rob128* was observed in all other samples of the current campaign. Next is the computer name of the compromised machine, followed by *_W*, which is hardcoded in all TrickBot samples we have encountered. A decimal number always follows the *_W*. Next is a decimal followed by a hexidecimal string 32 characters long. This string is created based on system time and involves using the function *kernel32.GetTickCount* and the instruction *RDTSC*, which is a time stamp counter. The next value in the path appears to signify the type of request being made and corresponds to values used in a switch statement that controls the flow of requests, displayed in *Figure 14*. For example, the initial check-in, which was created in case *5*, has the value *5* for this part of its path. Likewise, the URL containing the *NAT* status uses the value *14*, as it was created in the function corresponding to case *14*.

```
do {
  if (true) {
    switch(switch_flag) {
    case 0:
      iVar2 = C2_communication_0(param_1,keyWord,param_4);
      break;
    case 1:
      iVar2 = C2_communication_1(param_1);
      break;
    case 5:
      iVar2 = C2_communication_5(param_1,keyWord,param_4,param_5);
      break;
    case 10:
      iVar2 = C2_communication_10(param_1);
      break;
    case 0xe:
      iVar2 = C2_communication_14(param_1,keyWord,param_4);
      if ((1 < iVar2 - 0x193U) && (iVar2 != 200)) {
        (*pcVar1)(10000);
      }
      break;
    case 0x17:
      local_res18 = (undefined4)keyWord;
      iVar2 = C2_communication_23(param_1,local_res18,param_4,param_5,param_6);
      break;
    case 0x19:
      iVar2 = C2_communication_25(param_1,keyWord);
      break;
    case 0x3f:
      iVar2 = C2_communication_63(param_1);
      break;
    case 0x40:
      iVar2 = C2_communication_64(param_5,param_1 + 0x18,keyWord,param_4,param_5,param_6,param_7,
                                  param_8);
    }
```

Figure 14: C2 Communication Switch Statement

An interesting feature observed in this variant is that after TrickBot obtains the public IP address of the victim machine, it will query IP blacklist services to determine the reputation of the IP address. As we can see in *Figure 15*, TrickBot calls *ws2_32.getaddrinfo*, which queries information about a specified IP. The value passed to its first parameter is *.zen.spamhaus.org*. *zen.spamhaus.org* is a domain name system blacklist (DNSBL) service. Prepended to this is the victim machine's IP address in reverse order. TrickBot also uses other DNSBL services to check the victim machine's IP address. These include *cbl.abuseat.org*, *b.barracudacentral.org*, *dnsbl-1.uceprotect.net*, and *spam.dnsbl.sorbs.net.*

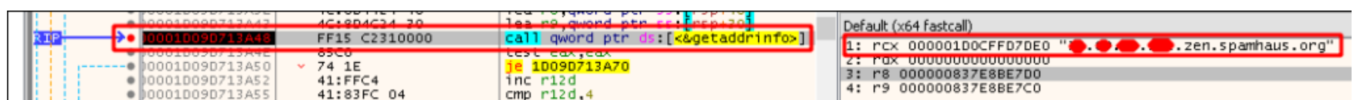


Figure 15: IP Reputation Check

TrickBot will then send a request to its C2 server stating the results of the reputation checks. As displayed in *Figure 16*, an example of the URL path generated for such requests is *rob128/<computer_name>_W10019077.33A1A5DD03BBFF0FD 7BA9BB14F9FBCDF/14/DNSBL/not%20listed/0/*. Of course, if any of the DNSBL services report the IP as blacklisted, *not%20listed* will be changed to *listed* in the URL path.

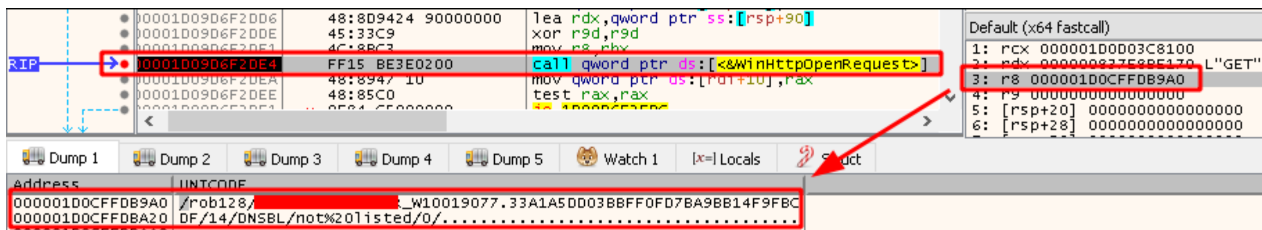


Figure 16: Reporting DNSBL Status to C2 Server

Conclusion

The notorious botnet and information stealer, TrickBot, has remained active since 2016 and continues to live up to its name, as it regularly incorporates new tricks into its already long list of abilities. TrickGate Loader is the latest addition to those tricks, and a very impressive one at that, since its use of Heaven's Gate allows it to effectively conceal API calls used to load TrickBot.

Through close monitoring, analyzing, and reverse engineering, GoSecure Titan Labs, as part of our GoSecure Titan Managed Detection and Response offering, have created signatures to detect the emerging threats discussed in this report. One such signature, listed below in the Detection section, is a file detection signature for the TrickBot shellcode entitled *malware_trickbot_4*, which was created using binlex, an opensource genetic binary trait lexer library and utility. By unpacking TrickBot shellcode from numerous samples of TrickGate, we were able to utilize binlex to extract the common traits and thus, to create an effective signature.

Increased work from home and remote work have led to a rise in these types of threats for users. Tools like GoSecure Titan IDR, which can be installed in desktop, mobile and web applications, allow users to send suspicious emails for expert analysis. This can help identify and remove potentially harmful threats from the environment before they spread—while also delivering samples to experts for documentation and reverse-engineering.

Malware Analyst: Sean Mahoney

Indicators of Compromise

type	indicator	decription
md5	906379938be59269713995cf29058f42	Malspam Email
md5	6e49d82395b641a449c85bfa37dbbbc2	LNK Downloader
md5	442f1e3d2825d51810bf9929f46439d2	TrickGate Loader
md5	87dc309108bbf70e3e67efbf9d4c09da	TrickGate Loader Shellcode
md5	8da11d870336c1c32ba521fd62e6f55b	64-bit PE
md5	0d9febdee78018daea87101c0d1a5362	Trickbot Shellcode
ip	97[.]83[.]40[.]67	TrickBot C2
ip	46[.]99[.]175[.]217	TrickBot C2
ip	46[.]99[.]175[.]149	TrickBot C2
ip	128[.]201[.]76[.]252	TrickBot C2
ip	103[.]105[.]254[.]17	TrickBot C2
ip	179[.]189[.]229[.]254	TrickBot C2
ip	24[.]162[.]214[.]166	TrickBot C2
ip	65[.]152[.]201[.]203	TrickBot C2
ip	62[.]99[.]76[.]213	TrickBot C2
ip	216[.]166[.]148[.]187	TrickBot C2
ip	184[.]74[.]99[.]214	TrickBot C2
ip	185[.]56[.]175[.]122	TrickBot C2
ip	181[.]129[.]167[.]82	TrickBot C2
ip	60[.]51[.]47[.]65	TrickBot C2
ip	46[.]99[.]188[.]223	TrickBot C2
ip	82[.]159[.]149[.]52	TrickBot C2
ip	45[.]36[.]99[.]184	TrickBot C2
ip	62[.]99[.]79[.]77	TrickBot C2

Detection

GoSecure Titan Labs are providing the following signatures to help the community in detecting and identifying the threats discussed in this report.

```
rule other_lnk_download_and_execute_0{
    meta:
      author      = "Titan Labs"
      company     = "GoSecure"
      description = "LNK downloading and executing a file"
      hash        = "6e49d82395b641a449c85bfa37dbbbc2"
      created     = "2021-10-14"
      tlp         = "white"
      os          = "windows"
      type        = "other"
      rev         = 1
    strings:
      $lnk        = { 4C 00 00 00 01 14 02 00 }
      $file_1     = ".exe" ascii wide nocase
      $file_2     = ".dll" ascii wide nocase
      $file_3     = ".scr" ascii wide nocase
      $file_4     = ".pif" ascii wide nocase
      $file_5     = "This program" ascii wide nocase
      $file_6     = "TVqQAA" ascii wide nocase
      $execute_1  = "cmd.exe" ascii wide nocase
      $execute_2  = "/c echo" ascii wide nocase
      $execute_3  = "/c start" ascii wide nocase
      $execute_4  = "/c set" ascii wide nocase
      $execute_5  = "%COMSPEC%" ascii wide nocase
      $execute_6  = "rundll32.exe" ascii wide nocase
      $execute_7  = "regsvr32.exe" ascii wide nocase
      $execute_8  = "Assembly.Load" ascii wide nocase
      $execute_9  = "[Reflection.Assembly]::Load" ascii wide nocase
      $execute_10 = "process call" ascii wide nocase
      $download_1 = "bitsadmin" ascii wide nocase
      $download_2 = "certutil" ascii wide nocase
      $download_3 = "ServerXMLHTTP" ascii wide nocase
      $download_4 = "http" ascii wide nocase
      $download_5 = "ftp" ascii wide nocase
      $download_6 = ".url" ascii wide nocase
      $download_7 = "curl" ascii wide nocase
    condition:
      $lnk at 0 and
      any of ($file_*) and
      any of ($execute_*) and
      any of ($download_*)
}
rule malware_trick_gate_loader_0 {
    meta:
        author      = "Titan Labs"
        company     = "GoSecure"
        description = "Tickbot Loader using Heaven's Gate"
        hash        = "442f1e3d2825d51810bf9929f46439d2"
        created     = "2021-11-04"
        os          = "windows"
        type        = "malware.loader"
        tlp         = "white"
        rev         = 1
    strings:
        $get_base_address = {
            55 8b ec 83 ec 14 89 4? ?? 8b 4? ?? 8b 4? ?? 89
            48 08 6a 40 8b 5? ?? 8b 42 08 50 ff 15 ?? ?? ??
            ?? 85 c0 74 ?? 8b 4? ?? c7 01 00 00 00 00 8b 5?
            ?? c7 42 04 00 00 00 00 e9 ?? ?? ?? ?? 8b 4? ??
            8b 4? ?? 8b 51 08 89 10 8b 4? ?? 8b 08 8b 51 3c
            89 5? ?? 8b 4? ?? 83 78 08 00 74 ?? 83 7? ?? 00
            74 ?? 8b 4? ?? 8b 51 08 03 5? ?? 89 5? ?? eb ??
            eb 07 c7 4? ?? 00 00 00 00 68 f8 00 00 00 8b 4?
            ?? 50 ff 15 ?? ?? ?? ?? 85 c0 74 ?? 8b 4? ?? c7
            41 04 00 00 00 00 eb ?? 8b 5? ?? 8b 02 8b 48 3c
            89 4? ?? 8b 5? ?? 83 7a 08 00 74 ?? 83 7? ?? 00
            74 ?? 8b 4? ?? 8b 48 08 03 4? ?? 89 4? ?? eb ??
            eb 07 c7 4? ?? 00 00 00 00 8b 5? ?? 8b 4? ?? 89
            42 04 8b 4? ?? 8b e5 5d c2 04 00}
        $resolve_api_call = {
            55 8b ec 6a ff 68 ?? ?? ?? ?? 64 a1 00 00 00 00
            50 64 89 25 00 00 00 00 81 ec 94 00 00 00 89 8?
            ?? ?? ?? ?? 8b 4? ?? 50 8b 8? ?? ?? ?? ?? e8 ??
            ?? ?? ?? c7 4? ?? 00 00 00 00 8b 4? ?? 8b 11 8b
            42 04 8b 4? ?? 8b 54 01 0c 89 5? ?? 33 c0 83 7?
            ?? 00 0f 94 ?? 0f b6 c8 85 c9 74 ?? 8b 5? ?? 8b
            02 8b 48 04 8b 5? ?? 8b 44 0a 3c 89 4? ?? 83 7?
            ?? 00 74 ?? 8b 4? ?? 8b 11 8b 42 04 8b 4? ?? 8b
            54 01 3c 89 5? ?? 8b 4? ?? e8 ?? ?? ?? ?? 8b 4?
            ?? 8b 08 8b 51 04 8b 4? ?? 8b 4c 10 0c 89 8? ??
```

```
            ?? ?? ?? 33 d2 83 b? ?? ?? ?? ?? 00 0f 94 ?? 8b
            8? ?? ?? ?? ?? 88 50 04 c7 4? ?? ff ff ff ff 8b
            8? ?? ?? ?? ?? 8b 4? ?? 64 89 0d 00 00 00 00 8b
            e5 5d c2 04 00}
        $heap_writing_function = {
            5? 8b ?? 6a ?? 68 ?? ?? ?? ?? 64 a1 ?? ?? ?? ??
            5? 64 89 ?? ?? ?? ?? ?? 5? 81 e? ?? ?? ?? ?? 5?
            5? 5? 89 ?? ?? c7 4? ?? ?? ?? ?? ?? 8b ?? ?? 89
            ?? ?? ?? ?? ?? 8b ?? ?? ?? ?? ?? 83 c? ?? 89 ??
            ?? ?? ?? ?? 8b ?? ?? ?? ?? ?? 8a ?? 88 ?? ?? ??
            ?? ?? 83 8? ?? ?? ?? ?? ?? 80 b? ?? ?? ?? ?? ??
            75 ?? 8b ?? ?? ?? ?? ?? 2b ?? ?? ?? ?? ?? 89 ??
            ?? ?? ?? ?? 8b ?? ?? ?? ?? ?? 33 ?? 89 ?? ?? 89
            ?? ?? 8b ?? ?? 8b ?? 8b ?? ?? 8b ?? ?? 8b ?? ??
            ?? 89 ?? ?? 8b ?? ?? ?? 89 ?? ?? 83 7? ?? ?? 7c
            ?? 7f ?? 83 7? ?? ?? 76 ?? 8b ?? ?? 8b ?? 8b ??
            ?? 8b ?? ?? 8b ?? ?? ?? 89 ?? ?? 8b ?? ?? ?? 89
            ?? ?? 8b ?? ?? 3b ?? ?? 7c ?? 7f ?? 8b ?? ?? 3b
            ?? ?? 76 ?? 8b ?? ?? 8b ?? 8b ?? ?? 8b ?? ?? 8b
            ?? ?? ?? 89 ?? ?? 8b ?? ?? ?? 89 ?? ?? 8b ?? ??
            2b ?? ?? 8b ?? ?? 1b ?? ?? 89 ?? ?? ?? ?? ?? 89
            ?? ?? ?? ?? ?? eb ?? c7 8? ?? ?? ?? ?? ?? ?? ??
            ?? c7 8? ?? ?? ?? ?? ?? ?? ?? ?? 8b ?? ?? ?? ??
            ?? 89 ?? ?? 8b ?? ?? ?? ?? ?? 89 ?? ?? 8b ?? ??
            5? 8d ?? ?? e8 ?? ?? ?? ?? c7 4? ?? ?? ?? ?? ??
            0f b6 ?? ?? f7 d? 1b ?? f7 d? 83 e? ?? 83 f? ??
            75 ?? 8b ?? ?? 83 c? ?? 89 ?? ?? e9 ?? ?? ?? ??}

    condition:
        uint16(0) == 0x5a4d and
        uint32(uint32(0x3c)) == 0x00004550 and
        all of them
}

rule malware_trick_gate_loader_shellcode_0 {
    meta:
        author      = "Titan Labs"
        company     = "GoSecure"
        description = "Shellcode decrypted from TrickGate's resource section"
        hash        = "87dc309108bbf70e3e67efbf9d4c09da"
        created     = "2021-11-04"
        os          = "windows"
        type        = "malware.loader"
        tlp         = "white"
        rev         = 1
    strings:
        $decryption_routine = {
            5? 4? 75 ?? 5? 8b ?? 8b ?? 05 ?? ?? ?? ?? 68 ??
            ?? ?? ?? 89 ?? ?? 5? 8b ?? 4? 8b ?? 4? 8b ?? 66
            ad 85 ?? 74 ?? 3b ?? 77 ?? 2b ?? c1 e? ?? 5? 8b
            ?? 03 ?? 81 c? ?? ?? ?? ?? 8b ?? 5? 03 ?? 5? eb
            ?? 89 ?? ?? b? ?? ?? ?? ?? 03 ?? 8b ?? 2b ?? 2b
            ?? 8b ?? 89 ?? ?? 8b ?? 83 e? ?? 8b ?? c7 4? ??
            ?? ?? ?? ?? 89 ?? 5? ff d?}
    condition:
        $decryption_routine
}
rule malware_trickbot_4 {
    meta:
        author      = "Titan Labs"
        company     = "GoSecure"
        description = "Unpacked Trickbot Shellcode"
        created     = "2021-11-26"
        type        = "malware.botnet"
        os          = "windows"
        tlp         = "white"
        hash        = "0d9febdee78018daea87101c0d1a5362"
        rev         = 1
    strings:
        $heap_write     = {
            90 90 90 90 90 90 90 90 90 90 90 90 90 0f b6 1a
            88 18 0f b6 5a ?? 88 58 ?? 0f b6 5a ?? 88 58 ??
            0f b6 5a ?? 88 58 ?? 0f b6 5a ?? 88 58 ?? 0f b6
            5a ?? 88 58 ?? 0f b6 5a ?? 88 58 ?? 49 83 c0 f8
            0f b6 5a ?? 48 8d 52 ?? 88 58 ?? 48 8d 40 ?? 75
            bc}
        $requestOptions = {
            c7 44 24 ?? 00 33 ?? ?? 4c 8d 44 24 ?? ba 1f ??
            ?? ?? 41 b9 04 ?? ?? ?? 48 8b c8 ff 15 ?? ?? ??
            ?? 85 c0 0f 84 96}
```

```
$createProcess  = {
    c7 44 24 ?? 68 ?? ?? ?? 48 8b ce ff 15 ?? ?? ??
    ?? 48 8b 8c 24 ?? ?? ?? ?? ?? 89 7c 24 48 48 89
    74 24 ?? 48 c7 44 24 ?? ?? ?? ?? ?? 48 c7 44 24
    ?? ?? ?? ?? ?? c7 44 24 ?? ?? ?? ?? ?? c7 44 24
    ?? ?? ?? ?? ?? 33 d2 45 33 c0 45 33 c9 ff 15 ??
    ?? ?? ?? 85 c0 74 68}
$get_path       = {
    33 db 48 8d 8c 24 ?? ?? ?? ?? ba 05 01 ?? ?? 45
    33 c9 4c 8b c6 ff 15 ?? ?? ?? ?? 85 c0 48 8b fe
    48 0f 44 fb 48 85 ff 75 0a}
$incrementVars  = {
    33 ff 48 8d 6c 24 ?? 90 90 90 90 90 90 90 90 90
    90 ff c7 66 83 7d ?? ?? 48 8d 6d ?? 75 f3}
$readFile_1     = {
    48 c7 44 24 ?? ?? ?? ?? ?? 4c 8d 4c 24 ?? 48 8b
    cb 49 8b d4 44 8b c5 ff 15 ?? ?? ?? ?? 33 ff 85
    c0 0f 95 c0 74 08}
$readFile_2     = {
    4c 8b 25 ?? ?? ?? ?? 33 f6 33 d2 45 33 c0 41 b9
    02 ?? ?? ?? 48 8b cb 41 ff d4 8b e8 89 6c 24 ??
    33 d2 45 33 c0 45 33 c9 48 8b cb 41 ff d4 85 ed
    74 58}
$query_headers  = {
    48 8b 4f ?? 48 8d 44 24 ?? 48 89 44 24 ?? 48 c7
    44 24 ?? ?? ?? ?? ?? 4c 8d 4c 24 ?? ba 13 00 00
    20 45 33 c0 ff 15 ?? ?? ?? ?? 8b c8 b8 01 ?? ??
    ?? 85 c9 75 38}
$return_static  = {
    55 48 8b ec 48 83 e4 f8 48 8d 0d ?? ?? ff ff 48
    8d 05 ?? ff ff ff 48 2b c1 48 8b e5 5d c3}
$logic_1        = {
    44 89 5c 24 ?? 4c 89 7c 24 ?? 48 89 54 24 ?? 48
    8b 44 24 ?? 48 8b 6c 24 ?? 48 3b e8 bd 4a c7 43
    0a 41 0f 42 ee 48 8b 44 24 ?? 81 fd af b7 12 f5
    7f 43}
$logic_2        = {
    48 83 7c 24 ?? ?? b8 95 6a 7d b9 b9 9f f8 cd a9
    0f 45 c1 e9 73 fb ff ff}
$logic_3        = {
    48 89 9c 24 ?? ?? ?? ?? 89 74 24 ?? 48 8b 84 24
    ?? ?? ?? ?? 80 38 ?? b8 0d 1a 75 84 b9 e1 21 8b
    44 0f 45 c1 e9 6f fa ff ff}
$logic_4        = {
    48 8b 44 24 ?? 48 89 44 24 ?? 48 8b 84 24 ?? ??
    ?? ?? 48 89 84 24 ?? ?? ?? ?? 48 8b 84 24 ?? ??
    ?? ?? 48 89 84 24 00 ?? ?? ?? 48 8b 84 24 ?? ??
    ?? ?? 48 89 84 24 ?? ?? ?? ?? 48 8b 84 24 ?? ??
    ?? ?? 48 89 84 24 ?? ?? ?? ?? 49 8b 07 0f b7 08
    89 4c ?? 24 48 83 c0 02 49 89 07 48 89 84 24 ??
    ?? ?? ?? 83 7c ?? 24 ?? b8 6b 7f a2 a5 b9 96 d1
    66 15 0f 45 c1 e9 a0 fc ff ff}
$logic_5        = {
    48 8b 44 24 ?? 8a ?? 48 8b 6c 24 ?? 88 45 ?? 48
    8b 44 24 ?? 80 38 ?? bd 9e 58 3a b3 41 0f 44 e9
    48 8b 44 24 ?? eb b5}
$logic_6        = {
    48 8b 6c 24 ?? 48 ff c5 48 89 6c 24 ?? 48 8b 6c
    24 ?? 48 ff c5 48 89 6c 24 ?? 48 8b 6c 24 ?? 8a
    5d ?? 88 5c 24 ?? 80 7c 24 ?? ?? bd 64 55 26 9d
    41 0f 45 ea e9 33 ff ff ff}
$logic_7        = {
    48 8b 84 24 ?? ?? ?? ?? 8b ?? 48 8b 4c 24 ?? 48
    8d 14 01 48 89 94 24 ?? ?? ?? ?? 8b 54 01 ?? 4c
    8b c3 8b df 44 8b ce 48 8b 74 24 ?? 48 03 f2 48
    89 b4 24 ?? ?? ?? ?? 49 8b d8 8b 54 01 ?? 48 8b
    74 24 ?? 48 03 f2 48 89 b4 24 ?? ?? ?? ?? 41 8b
    f1 8b 44 01 ?? 48 03 44 24 ?? 48 89 84 24 ?? ??
    ?? ?? b8 f7 db c4 c5 49 8b cd 48 89 4c 24 ?? e9
    96 f9 ff ff}
$logic_8        = {
    49 63 45 ?? 4c 89 6c 24 ?? 48 8b 4c 24 ?? 48 8d
    84 01 ?? ?? ?? ?? 48 89 84 24 ?? ?? ?? ?? 48 8b
    84 24 ?? ?? ?? ?? 48 89 84 24 ?? ?? ?? ?? 48 8b
    84 24 ?? ?? ?? ?? 83 38 ?? b8 4d 1d f8 fb b9 c3
    f8 ec ?? 0f 45 c1 e9 05 f7 ff ff}
$logic_9        = {
    49 8b 07 0f b7 08 89 4c 24 ?? 48 83 c0 02 48 89
    84 24 ?? ?? ?? ?? 48 8b 84 24 ?? ?? ?? ?? 49 89
    07 83 7c 24 ?? ?? b8 18 ab 0a 99 b9 28 86 7a 7a
```

```
            0f 45 c1 e9 76 f8 ff ff}
        $logic_10       = {
            83 7c 24 ?? ?? b8 79 5c ba 4b b9 a9 a6 56 b9 0f
            4f c1 8b 4c 24 ?? 89 4c 24 ?? e9 50 fa ff ff}
        $logic_11       = {
            8b 44 24 ?? 48 89 84 24 ?? ?? ?? ?? 48 8b 84 24
            ?? ?? ?? ?? 48 8b 8c 24 ?? ?? ?? ?? 48 8d 04 88
            48 89 84 24 ?? ?? ?? ?? 48 8b 84 24 ?? ?? ?? ??
            8b ?? 89 44 24 ?? 83 7c 24 ?? ?? b8 38 55 90 88
            b9 29 a5 0e be 0f 45 c1 e9 12 fe ff ff}
        $logic_12       = {
            8b 44 24 ?? 48 89 84 24 ?? ?? ?? ?? 48 8b 84 24
            ?? ?? ?? ?? 48 8b 8c 24 ?? ?? ?? ?? 8b 04 81 48
            03 44 24 ?? 48 89 84 24 ?? ?? ?? ?? b8 b9 23 c7
            33 33 f6 48 8b 9c 24 ?? ?? ?? ?? e9 96 fe ff ff
            }
        $logic_13       = {
            8b 44 24 ?? 89 44 24 ?? 48 8b 84 24 ?? ?? ?? ??
            48 83 c0 18 48 89 84 24 ?? ?? ?? ?? 48 8b 84 24
            ?? ?? ?? ?? 8b 4c 24 ?? 3b 08 b8 5c b1 a7 79 b9
            26 e6 ba 02 0f 42 c1 e9 21 fb ff ff}
        $logic_14       = {
            8b 44 24 ?? 89 44 24 ?? 8b 44 ?? 24 8b 4c 24 ??
            3b c8 b8 d0 78 7a 87 b9 07 2b 67 eb 0f 42 c1 c7
            44 24 ?? ?? ?? ?? ?? e9 58 f9 ff ff}
        $logic_15       = {
            8b 44 ?? 24 8b 4c 24 ?? 3b c8 b8 65 79 3f 9a b9
            3c 26 ab 78 0f 44 c1 8b 4c 24 ?? 89 4c 24 ?? e9
            ff f7 ff ff}
        $logic_16       = {
            8b 44 24 ?? 8b c8 f7 d1 81 e1 4d 5e 9b 89 25 b2
            a1 64 76 0b c1 35 dc fe ed bc 89 84 24 ?? ?? ??
            ?? b8 e1 65 b5 e3 33 ff e9 85 f7 ff ff}
        $logic_17       = {
            8b 74 24 ?? f7 de bf 01 ?? ?? ?? 2b fe 48 8b 74
            24 ?? 8a 5c 24 ?? 88 1e 48 8b 74 24 ?? 48 ff c6
            bd f3 dc 32 70 eb 2a}
        $logic_18       = {
            f6 44 24 ?? 01 b8 5d ff b5 c1 b9 7a 22 79 97 0f
            45 c1 e9 0d ff ff ff}
    condition:
        filesize < 328KB and 10 of them
}
alert http any any -> $EXTERNAL_NET any (
  msg:"GS MALWARE Trickbot C2 Communication";
  content:"_W"; http_uri; fast_pattern;
  pcre:"/^\/\w+\d+\/[^\/]+_W\d+\.[A-F0-9]{32}\/\d+\//U";
  flow:to_server, established;
  metadata:created 2019-06-06, updated 2021-11-25, type malware.botnet, os windows, tlp white, id 3;
  classtype:trojan-activity;
  sid:300000464;
  rev:3;
)
```