What's Different About the TrickBoot Version?

securityintelligence.com/posts/trickbot-survival-instinct-trickboot-version/

Home / Data Protection
TrickBot's Survival Instinct Prevails — What's Different About the TrickBoot Version?

Data Protection January 26, 2021
By Nir Shwarts 9 min read

October 2020 saw the TrickBot Trojan, a prominent cybercrime gang's tool of choice, suffer a takedown attempt by security vendors and law enforcement. Unfortunately, the takedown was not effective, and beyond coming back to life shortly after, TrickBot's operators released a new and more persistent version of the malware.

In this post, IBM Trusteer examines the new TrickBot version versus its precedent and looks into the components its developers kept or modified.

TrickBot Returns

What can be said about the TrickBot banking malware and botnet that has not been covered yet by the security community? After it emerged in 2016, built upon the ruins of the Dyre Trojan, TrickBot's developers have not laid down their cyber-arms. Looking back at the past two years, not only has TrickBot continued to plague the online banking applications that cater to businesses, it has also ventured into collaborations with other elite cybercrime gangs. Additionally, it has worked with Ryuk and other ransomware on big game hunting attacks, deployed cryptominers and was a concerning factor in election interference in 2020.

Like many other botnets, TrickBot has garnered a lot of attention from law enforcement and suffered a takedown attempt in October 2020. Unfortunately, despite losing a large proportion of its infrastructure, TrickBot not only re-emerged, but it also launched a new, and more persistent version than ever, using a UEFI/BIOS bootkit to help it remain undetected on infected devices. This bootkit was dubbed TrickBoot by those who first analyzed and reported about it.

Our team at IBM Trusteer research examined what changed between the two most recent versions of the TrickBot malware. Let's take a closer look at some of the components that keep TrickBot persistent and stealthy over time.

Compared Components

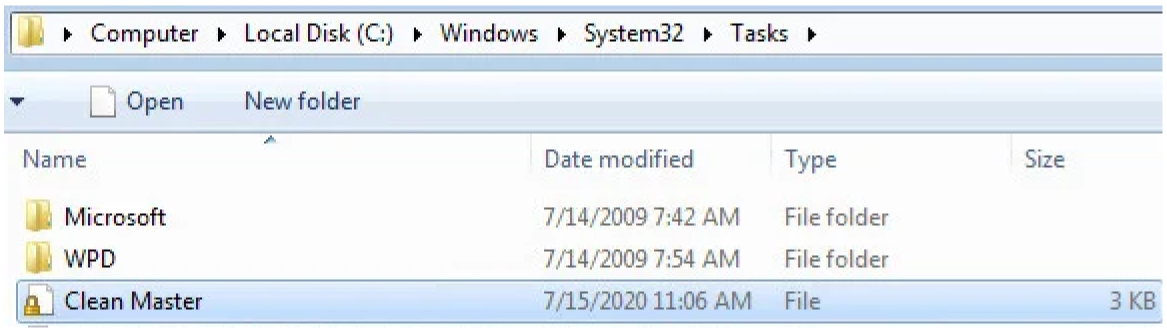
This comparison will look into a few major components from previous versions 1000512 and 1000513 versus the newer version 100003. Unusually, the version number jumped backward.

- Persistence mechanism
- Injection technique
- Bot configuration with a random twist
- Mutex naming concept
- Compromise check

Persistence Mechanism – Modified in New Version

Most malicious software developers implement some sort of persistence mechanism to ensure the malware is not wiped from the device on reboot. TrickBot's long-standing persistence method uses a scheduled task that is set to run on set time lapses. This tactic was modified in the newer version.

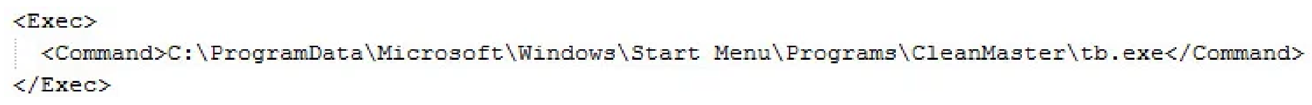
In the previous version, the scheduled task appeared under the name Clean Master.



Name	Date modified	Type	Size
Microsoft	7/14/2009 7:42 AM	File folder	
WPD	7/14/2009 7:54 AM	File folder	
Clean Master	7/15/2020 11:06 AM	File	3 KB

Figure 1: TrickBot's scheduled task created under the task directory

This task's executable file's path is directly in the 'Programs' menu:

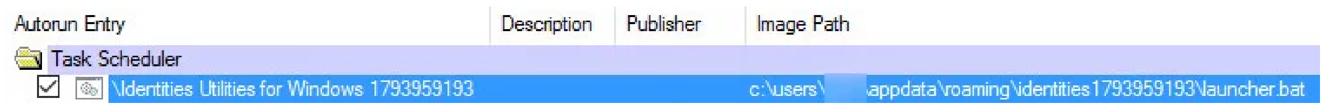


```
<Exec>
  <Command>C:\ProgramData\Microsoft\Windows\Start Menu\Programs\CleanMaster\tb.exe</Command>
</Exec>
```

Figure 2: Task's executable location

The malware further copies itself into a new subdirectory as part of its stealth tactics, thereby making Clean Master appear as yet another legitimate program and not suspicious at first sight.

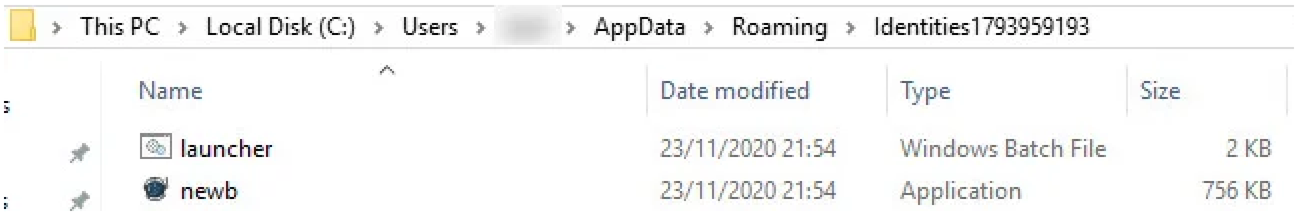
In the new version, although TrickBot maintained the persistence mechanism of using the Windows Task Scheduler as the way to reload the malware, the task's destination is now changed to a batch file. This is a stealthier choice by TrickBot's developers, helping the malware evade scans by antivirus and security tools that typically look for malicious executables in the autorun and task scheduler.



Autorun Entry	Description	Publisher	Image Path
Task Scheduler			
\Identities Utilities for Windows 1793959193			c:\users\[redacted]appdata\roaming\identities1793959193\launcher.bat

Figure 3: Autorun snippet of the new task

A quick look into the batch file location exposes TrickBot's executable, named 'newb' in this case.



This PC > Local Disk (C:) > Users > [redacted] > AppData > Roaming > Identities1793959193

Name	Date modified	Type	Size
launcher	23/11/2020 21:54	Windows Batch File	2 KB
newb	23/11/2020 21:54	Application	756 KB

Figure 4: TrickBot's batch file along its executable

Further in the batch file, we also see an obfuscated script:

```
set rnqsmn=set
%rnqsmn% irreg=
%rnqsmn%%irreg%rmqceo==
%rnqsmn%%irreg%jegqb%rmqceo%sta
%rnqsmn%%irreg%brnpv%rmqceo%ata
%rnqsmn%%irreg%ckuy%rmqceo%er
...
%rnqsmn%%irreg%oqldm%rmqceo%ppD
%rnqsmn%%irreg%mhjjn%rmqceo%.e
%rnqsmn%%irreg%kjwhcf%rmqceo%oam
%jegqb%%qplno%%irreg%%lyjlin%%mhjl%%oijb%%ckuy% ...
```

Figure 5: A part of TrickBot's obfuscated batch file

De-obfuscating the script reveals its more obvious purpose as running the TrickBot executable (newb.exe) from the following patch:

```
start C:\Users\[name here]\AppData\Roaming\Identities1793959193\newb.exe
```

Choice of Injection Mechanism – Retained

The injection of malicious code into legitimate Windows processes is one of the most common tactics malware developers use. The code injection can take different forms, but a highly popular way to implement code injection is process hollowing, in which legitimate content is removed and replaced by the attacker's code.

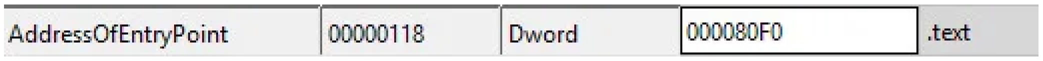
TrickBot uses process hollowing as its code injection tactic, and that has remained the same in the previous and current versions of the malware. This method has a couple of advantages:

- Allows for the termination of the main suspicious process
- Presents the malware's operations as being performed by legitimate processes

Here's what TrickBot's process hollowing does:

1. Creates process wermgr.exe in suspend mode using CreateProcessInternalW. This Windows process is inherently responsible for handling errors.
2. Allocates memory for its payload shellcode using NtAllocateVirtualMemory

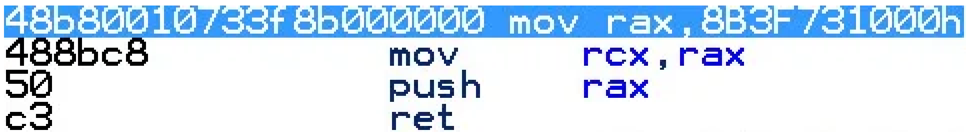
3. Writes into memory using NtWriteVirtualMemory
4. Reads remote process's Process Environment Block (PEB) using NtQueryInformationProcess
5. Reads first module wermgr.exe's address using the PEB structure
6. Goes to the 0xF0+0x28=0x118 offset in the wermgr.exe address to find its entry point:



AddressOfEntryPoint	00000118	Dword	000080F0	.text

Figure 6: Wermgr's entry-point address

1. Writes a trampoline to the entry point 0x80F0 and the module's base address to jump to the shellcode using NtWriteVirtualMemory. A trampoline is a small piece of code which is constructed on the fly on the stack when the address of a nested function is taken.
2. Calls NtResumeThread to resume the thread.



```
48b80010733f8b000000 mov  rax,8B3F731000h
488bc8               mov   rcx,rax
50                   push  rax
c3                   ret
```

Figure 7: Example of TrickBot's trampoline

Bot Configuration Receives Randomization Twist

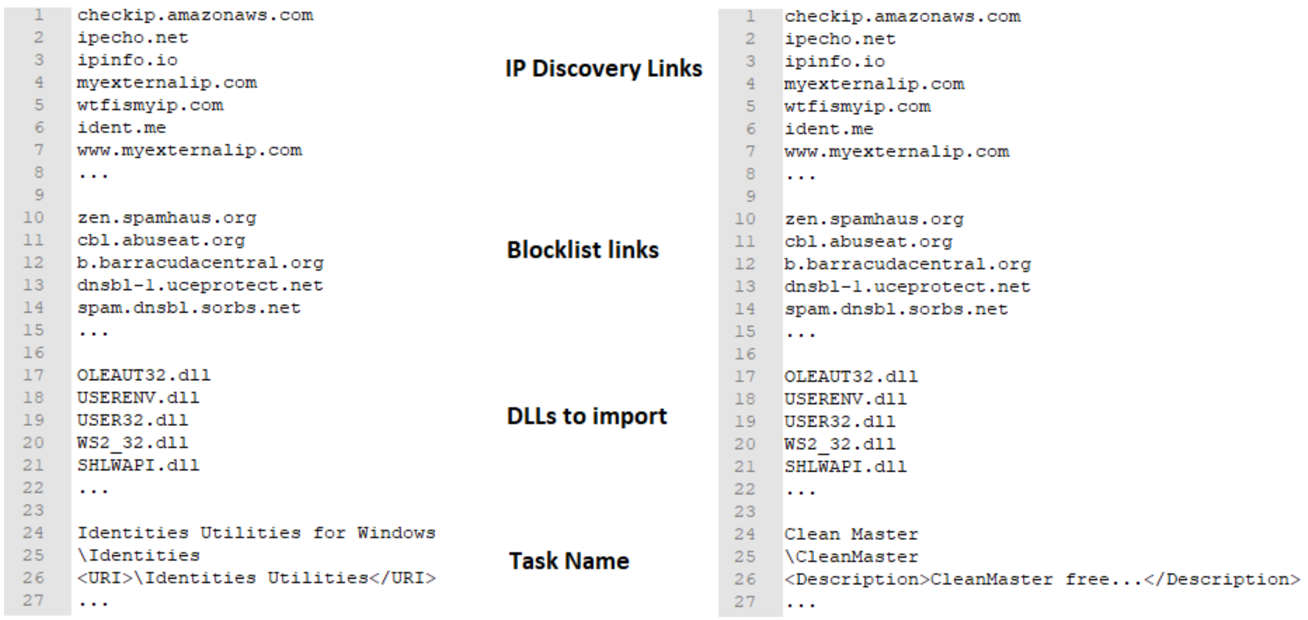
Bot configuration is the way the malware configures its own file on initial setup when a new machine is infected or when it is updated on infected devices. The initial configuration contains data such as encryption keys, IDs, credentials, URLs and communication patterns it will use to send information to its botmaster.

Bot configurations are easy to modify and are changed by malware operators in their various campaigns and attacks, offering agility and stealth.

TrickBot's initial configuration elements are traditionally split between its embedded XML 'mcconf' configuration file and Base64 encrypted strings stored within the unpacked TrickBot core. The 'mcconf' file contains a list of C2 server IP addresses, along with the TrickBot version number, Bot ID and default modules. Remaining configuration elements such as install directory and scheduled task name are stored as custom Base64 encoded strings.

TrickBot retained the same bot configuration scheme from its previous versions and continued to use the same one in current versions, only this time, it added some randomization to data inside the file to delay detection and analysis.

When we looked at the details, we saw that only task names have been modified, but all the data and purposes remain the same. We will come back to task names later.



```
checkip.amazonaws.com
ipecho.net
ipinfo.io
myexternalip.com
wtfismyip.com
ident.me
www.myexternalip.com
...

zen.spamhaus.org
cbl.abuseat.org
b.barracudacentral.org
dnsbl-1.uceprotect.net
spam.dnsbl.sorbs.net
...

OLEAUT32.dll
USERENV.dll
USER32.dll
WS2_32.dll
SHLWAPI.dll
...

Identities Utilities for Windows
\Identities
<URI>\Identities Utilities</URI>
...
```

```
checkip.amazonaws.com
ipecho.net
ipinfo.io
myexternalip.com
wtfismyip.com
ident.me
www.myexternalip.com
...

zen.spamhaus.org
cbl.abuseat.org
b.barracudacentral.org
dnsbl-1.uceprotect.net
spam.dnsbl.sorbs.net
...

OLEAUT32.dll
USERENV.dll
USER32.dll
WS2_32.dll
SHLWAPI.dll
...

Clean Master
\CleanMaster
<Description>CleanMaster free...</Description>
...
```

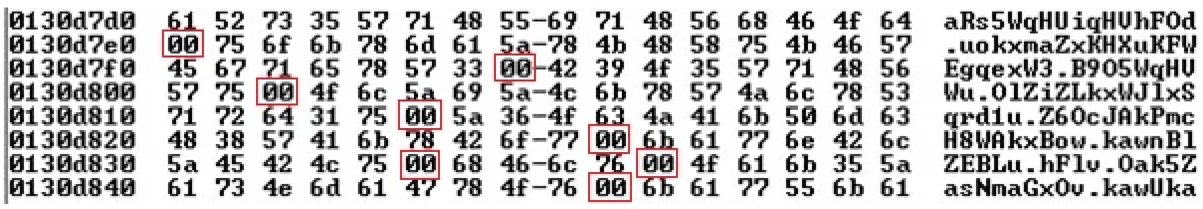
Figure 8: TrickBot configuration elements from decrypted base64 strings — excerpt comparison from previous and current versions

TrickBot's configuration is embedded within the unpacked TrickBot core binaries. The extraction process takes place as follows:

1. Upon execution, TrickBot is unpacked. TrickBot makes use of a variety of different packing techniques; in this case, the malware is stored as a resource of the parent executable and decrypted using a modified implementation of the RC4 cipher with a hardcoded key embedded within the file. This technique is also commonly used by Emotet.

The only difference between the standard RC4 and this modified version is the change of the standard RC4 constant value from 256 to 1147.

1. The resulting binary is the TrickBot Loader. This is a highly obfuscated binary that decrypts its other functions and payloads as required using a XOR operation. This loader is responsible for decrypting and loading the TrickBot core binaries. The loader decrypts the appropriate 32/64 bit core binary using XOR and decompresses it using an LZO algorithm.
2. Within the TrickBot core binary, the 'mcconf' configuration file is embedded and encrypted using AES and XOR. The binary also contains a list of custom base64 encoded strings separated by the NULL byte '\x00'.



```
0130d7d0  61 52 73 35 57 71 48 55-69 71 48 56 68 46 4f 64  aRs5WqHUiqHVhFOd
0130d7e0  00 75 6f 6b 78 6d 61 5a-78 4b 48 58 75 4b 46 57  .uokxmaZxKHXuKFW
0130d7f0  45 67 71 65 78 57 33 00-42 39 4f 35 57 71 48 56  EgqexW3.B9O5WqHV
0130d800  57 75 00 4f 6c 5a 69 5a-4c 6b 78 57 4a 6c 78 53  Wu.OlZiZLkxWJlxS
0130d810  71 72 64 31 75 00 5a 36-4f 63 4a 41 6b 50 6d 63  qrd1u.Z6OcJAkPmc
0130d820  48 38 57 41 6b 78 42 6f-77 00 6b 61 77 6e 42 6c  H8WAkxBow.kawnBl
0130d830  5a 45 42 4c 75 00 68 46-6c 76 00 4f 61 6b 35 5a  ZEBLu.hFlv.Oak5Z
0130d840  61 73 4e 6d 61 47 78 4f-76 00 6b 61 77 55 6b 61  asNmaGxOv.kawUka
```

Figure 9: Part of the encoded TrickBot configuration, separated by NULL bytes

1. The last string is picked to become a custom Base64 encoding index.
2. When required, every string the malware calls is then pushed to a Base64 decoding function that uses the custom index to decode the configuration. This way, the specific configuration is in plaintext only when it's used, which can make both dynamic and static analysis by outsiders take longer.

The Base64 binary data encoding uses an index table of 64 characters, which typically includes basic characters such as: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

With TrickBot's configuration, the index table used in our sample was:

3GjYRHFAKkzMws2IuZpiJOqamWgSBh14nE0N8x9LbXQt5yUPvedVcl6ofTD/+rC7

A closer look revealed that TrickBot's encryption methods are customized by its developers. For example, Trickbot uses different implementation for the RC4 cipher and a modified Base64 decoding algorithm, incorporating a custom table in order to make analysis more difficult. These choices are likely there since most crypto detection scripts are not likely to find such functions automatically.

Bot Configuration Changes

In the beginning of this section, we mentioned that only task names in the bot configuration were modified. These names change frequently and, in the newer samples, task names are appended a randomized 10-digit number that modifies the name each time. For example, for the task name 'identities' we will now see a string like 'Identities1793959193'. The developer likely uses this random twist to eliminate the ability to search for constants with detection tools.

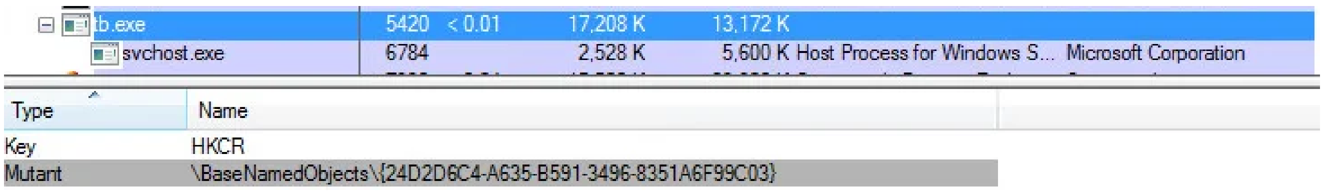
The randomization itself is achieved by a couple of additional calculations and XOR operations on two values that TrickBot fetches. In this case, it uses the result that is produced at that moment from the computer's processor's read time-stamp counter command (RDTSC[1]) and the result from the Windows GetTickCount [2] function.

Fully Randomized Mutex Naming – Retained

The mutex — the mutual exclusion object that synchronizes access to a resource — is very commonly used by malware developers. As a result, mutexes are part of the indicators of compromise to look for when trying to determine if a device has been infected by specific malware. Searching for mutexes is also something malware does to ensure it does not re-infect a device and cause instability, or to kill the processes of a rival malware.

TrickBot uses a mutex naming algorithm to create random, unique mutexes across infected devices that it manages to control. Since mutexes are a definitive indicator of compromise, TrickBot's developers were creative in creating the algorithm used here, staying away from relying on system artifacts that can be more easily figured out.

In current versions, TrickBot's mutexes are created as fully randomized 32-digits consisting of a 16- byte globally unique identifier (GUID).



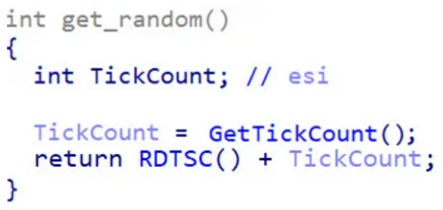
```
tb.exe        5420  < 0.01   17,208 K   13,172 K
  svchost.exe 6784           2,528 K    5,600 K Host Process for Windows S...  Microsoft Corporation

Type     Name
Key      HKCR
Mutant   \BaseNamedObjects\{24D2D6C4-A635-B591-3496-8351A6F99C03}
```

Figure 10: An example of a TrickBot mutex name

In the process of creating the mutex name, 'get_random', TrickBot splits it into two parts:

1. Random [8 Random bytes]
2. Related [8 Related bytes]

To randomize the strings, TrickBot uses the GetTickCount function and the RDTSC assembly instruction.



```
int get_random()
{
  int TickCount; // esi

  TickCount = GetTickCount();
  return RDTSC() + TickCount;
}
```

Figure 11: TrickBot's simple get_random function

The second part, or 'related part', is built upon the first. It's a combination of some XORing operations of the first, 'random part' with the computer's volume C serial number. Finally, the Windows StringFromCLSID [3] function is used to give the mutex its GUID form.

What's important in the mutex name here is not the value, but the correlation between its first and second halves.

Previous Version Checks

Before the TrickBot malware infects a device, it runs a detection process to ensure that device has not already been infected.

This detection takes us back to TrickBot's mutex, which is what it looks for on the device. Since mutex names are randomized, it cannot directly look for a name. Rather, it examines the relation between the first half and the second half of the mutex, and then infers whether the resulting GUID has or has not been generated in the past.

Let’s take a closer look at the mutex enumeration process that TrickBot performs:

1. It. begins by enumerating all system handles. In Windows-based devices, a handle is a pointer to an object, such as a mutex, a file, etc. The malware does that by calling the ZwQueryInformation Native API[4] with SYSTEM_HANDLE_INFORMATION (0x10) as the parameter.
2. Next, it uses the NtQueryObject function it queries for information about each object, and in our case each handle, to search mutex names.
3. Then it uses the OpenProcess function to open the process the mutex handle belongs to.
4. Lastly, it uses the DuplicateHandle function to duplicate the handle to that mutex so it can access it.

Note that the first two functions are Windows Native APIs belonging to the NTDLL.dll library. Usually, when calling a Windows API function from the KERNEL32.dll, it calls its Windows Native API counterpart function in NTDLL.dll that address the Windows kernel.

At this point, if the device is deemed infected, TrickBot initiates an ExitProcess to abandon a second installation of the malware.

TrickBot Lives On – A Reminder to Stay Vigilant

As we kick off 2021, the TrickBot Trojan and the cybercrime syndicate around it are in full swing, operating on the bank fraud front as well as high-stakes ransomware attacks.

If your security team detects TrickBot on networked devices, those should become high priority to clean or reimage as needed. This malware can be a foot in the door for a number of attacks, the worst of which can be an all-out ransomware and extortion operation against your organization.

IBM Trusteer fights threats like TrickBot and has been helping customers mitigate attacks since 2006. Our fraud detection and advanced risk-based authentication help banks protect consumers and businesses, and our research team is pivotal in detecting highly modular, modern-day malware like TrickBot and its components.

To learn more about IBM Trusteer, visit www.ibm.com/security/fraud-protection/trusteer.

---

[1] RDTSC loads the current value of the processor’s time-stamp counter into the EDX:EAX registers.

[2] Retrieves the number of milliseconds that have elapsed since the system was started, up to 49.7 days.

[3] The StringFromCLSID function converts a CLSID into a string of printable characters. Different CLSIDs will convert to different strings.

[4] The Native API is a lightweight application programming interface (API) used by Windows NT and user mode applications.

Malware | TrickBot | Trojan

Nir Shwarts
Malware Research-Reverse Engineering, IBM Security

Nir Shwarts is a contributor for SecurityIntelligence.