The Dragon Who Sold His Camaro: Analyzing Custom Router Implant

research.checkpoint.com/2023/the-dragon-who-sold-his-camaro-analyzing-custom-router-implant/

May 16, 2023

Research by: Itay Cohen, Radoslaw Madej, and the Threat Intelligence Team

Over the past few months, Check Point Research has closely monitored a series of targeted attacks aimed at European foreign affairs entities. These campaigns have been linked to a Chinese state-sponsored APT group we track as ***Camaro Dragon***, which shares similarities with previously reported activities conducted by state-sponsored Chinese threat actors, namely Mustang Panda.

Our comprehensive analysis of these attacks has uncovered a malicious firmware implant tailored for TP-Link routers. The implant features several malicious components, including a custom backdoor named "Horse Shell" that enables the attackers to maintain persistent access, build anonymous infrastructure and enable lateral movement into compromised networks.

The discovery is yet another example of a long-standing trend of Chinese threat actors to exploit Internet-facing network devices and modify their underlying software or firmware. This blog post will delve into the intricate details of analyzing the "Horse Shell" router implant. We will share our insights into the implant's functionality and compare it to other router implants associated with Chinese state-sponsored groups. By examining this implant, we hope to shed light on the techniques and tactics utilized by the Camaro Dragon APT group and provide a better understanding of how threat actors utilize malicious firmware implants in network devices in their attacks.

Key Findings

- Checkpoint Research has discovered and analyzed a custom firmware image affiliated with the Chinese state-sponsored actor “Camaro Dragon”.
- The firmware image contained several malicious components, including a custom MIPS32 ELF implant dubbed “Horse Shell”. In addition to the implant, a passive backdoor providing attackers with a shell to infected devices was found.
- “Horse Shell”, the main implant inserted into the modified firmware by the attackers, provides the attacker with 3 main functionalities:
    - Remote shell — Execution of arbitrary shell commands on the infected router
    - File transfer — Upload and download files to and from the infected router.
    - SOCKS tunneling — Relay communication between different clients.
- Due to its firmware-agnostic design, the implant’s components can be integrated into various firmware by different vendors
- The deployment method of the firmware images on the infected routers is still unclear, as well as its usage and involvement in actual intrusions.

Background

Since January 2023, Check Point Research is tracking sophisticated attacks targeting officials in multiple European countries. The campaign leveraged a wide variety of tools, among them implants commonly associated with Chinese state-sponsored threat actors. This activity has significant infrastructure overlaps with activities publicly disclosed by our fellow researchers in Avast and ESET, linking it to “Mustang Panda”. This cluster of activity is currently tracked by CPR as “Camaro Dragon”.

Through our detailed analysis of files and infrastructure associated with this campaign, we have discovered a trove of files and payloads used by the group. Among these files, there were two that caught our attention. These were two modified TP-Link router firmware images. As we dug further, it became evident those were tempered with, adding several malicious components to the original firmware, including a custom implant dubbed “Horse Shell”.

The implanted components were discovered in modified TP-Link firmware images. However, they were written in a firmware-agnostic manner and are not specific to any particular product or vendor. As a result, they could be included in different firmware by various vendors. While we have no concrete evidence of this, previous incidents have demonstrated that similar implants and backdoors have been deployed on diverse routers and devices from a range of vendors.

Uncovering the Implants

When faced with a large number of files, it is necessary to quickly triage and filter them to identify those that are more relevant for further inspection. To do this, there are several strategies that can be employed, one of which involves understanding the type of files that are being dealt with.

It is important to note that certain file types are more likely to contain relevant information than others. For instance, graphic images and icons may not be as significant as executable and firmware files. Therefore, to filter through the large number of files in question, we decided to employ the Linux `file` command, which helped us determine the file types.

Upon running the command, we discovered that two of the files were TP-Link firmware images of a rather dated model, WR940, that was initially released around 2014.

```
9404.bin: firmware 940 v4 TP-LINK Technologies ver. 1.0, version 3.16.9, [...]
9406.bin: firmware 940 v6 TP-LINK Technologies ver. 1.0, version 3.20.1, [...]
```

The output of our query showcased that both files pertained to the same model of TP-Link router, albeit intended for different hardware versions, specifically v4 and v6, respectively. The presence of these router firmware files, situated alongside dubious files and tools at the hands of an advanced threat actor, undoubtedly raised suspicion and warranted a thorough investigation.

As the firmware claimed to be for the TP-Link router model WR940N, we aimed to compare the original firmware of both v4 and v6 with the ones we had obtained, analyzing any potential differences. To do so, we procured the original firmware for this model from the TP-Link website, meticulously scrutinizing each component to identify any discrepancies.

Upon inspection, we discovered that the kernel and the uBoot of both firmware versions were identical, indicating that they had not been tampered with by the attackers. However, the filesystems were notably distinct, prompting us to extract and compare them. The firmware are using a custom implementation of SquashFS. To extract the filesystem we used sasquatch. We carried out a meticulous file-by-file comparison. Our aim was to identify which files had been modified, added, or removed, if any.

By conducting a meticulous analysis of each file, we aimed to discern which files had been modified, added, or removed from the suspicious firmware we had encountered. In doing so, we hoped to uncover any potential alterations made by the threat actor.

And indeed, we found that multiple files were added to the firmware we obtained, and a couple of files were modified:

Files added:

- /usr/bin/sheel
- /usr/bin/shell
- /usr/bin/timer
- /usr/bin/udhcp

Files Modified:

- /etc/rc.d/rcS
- /web/userRpm/SoftwareUpgradeRpm.htm

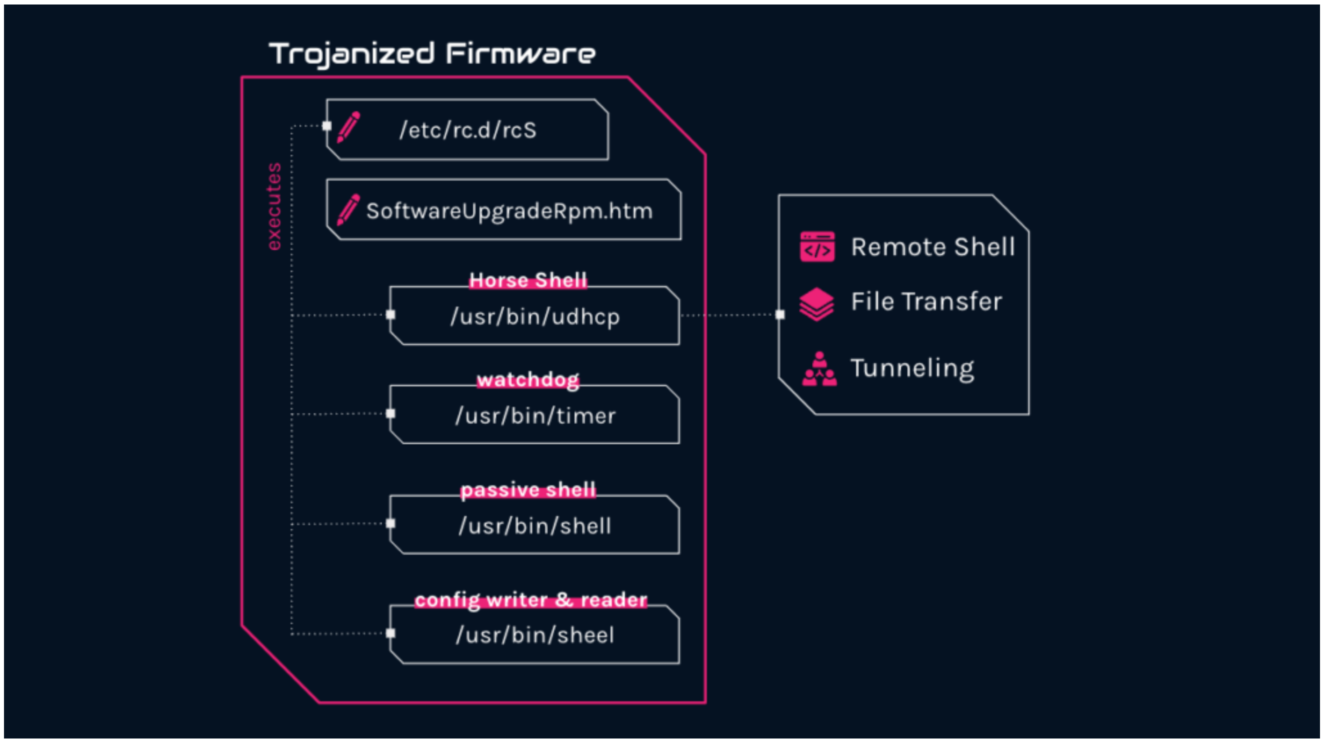


Figure 1: Overview of the different components in the malicious implant.

Initial Infection

We are unsure how the attackers managed to infect the router devices with their malicious implant. It is likely that they gained access to these devices by either scanning them for known vulnerabilities or targeting devices that used default or weak and easily guessable passwords for authentication. The goal of the attackers appears to be the creation of a chain of nodes between main infections and real command and control, and if so, they would likely be installing the implant on arbitrary devices with no particular interest.

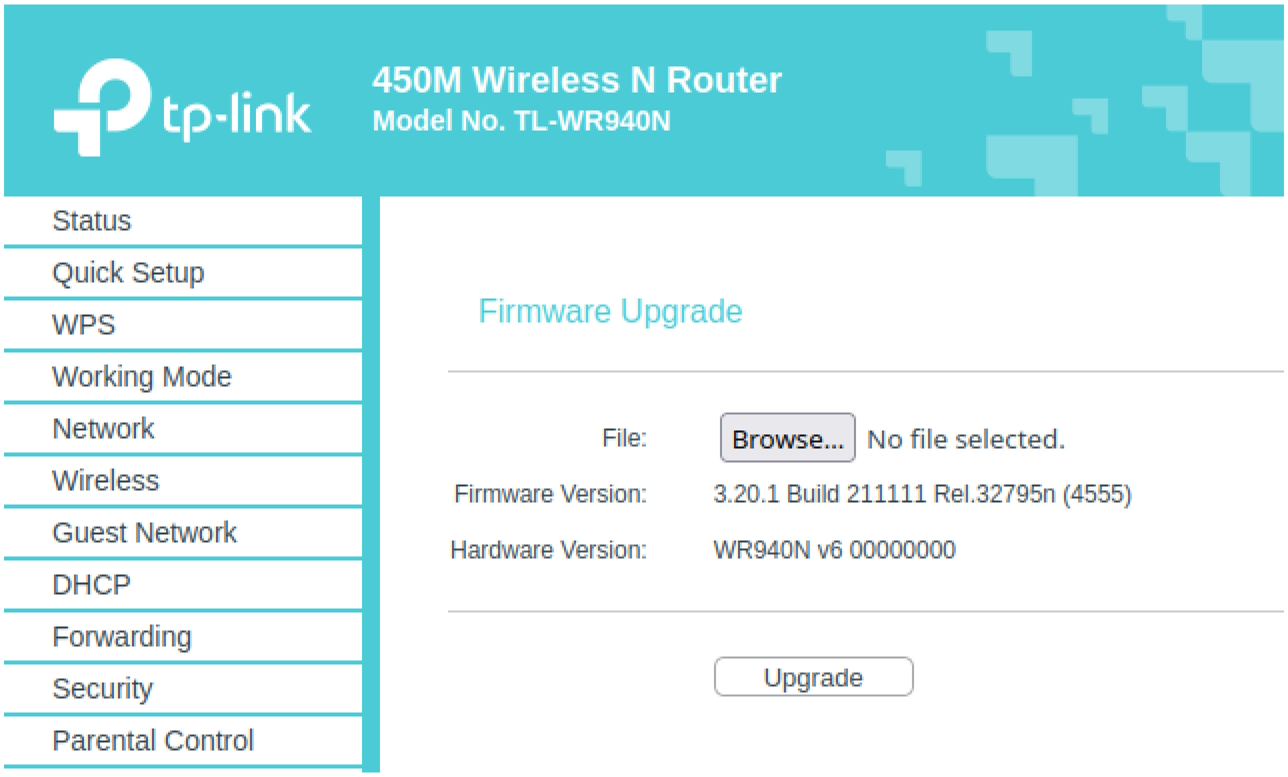
It is worth noting that this kind of attack is not aimed specifically at sensitive networks, but rather at regular residential and home networks. Therefore, infecting a home router does not necessarily mean that the homeowner was a specific target, but rather that their device was merely a means to an end for the attackers.

Inspecting the Modified Files

SoftwareUpgradeRpm.htm

The TP-Link router, like many routers, has a web interface that allows its users to configure the router and manage it. One of the features of the management website provides the user with the option to manually upgrade their device's firmware version. The web form for uploading a new firmware exists in `SoftwareUpgradeRpm.htm`.

This page, on the original and legitimate firmware we obtained from the official TP-Link website, is shown in the image below.



File:	No file selected.
Firmware Version:	3.20.1 Build 211111 Rel.32795n (4555)
Hardware Version:	WR940N v6 00000000

Figure 2: SoftwareUpgradeRpm.htm as shown in the original interface
However, in the modified version of the firmware we obtained, a small CSS property was inline added to the HTML form. This property, `display:none`, will hide the form from a user entering the page.

```
<FORM action="../incoming/Firmware.htm" enctype="multipart/form-data" method="post"
onSubmit="return doSubmit();" style="display: none;">
```

Hiding the form, will not remove it or the feature from the HTML itself, so users can technically still manually upgrade their firmware version. Although now, it will be harder to perform the upgrade or even know that this feature exists.

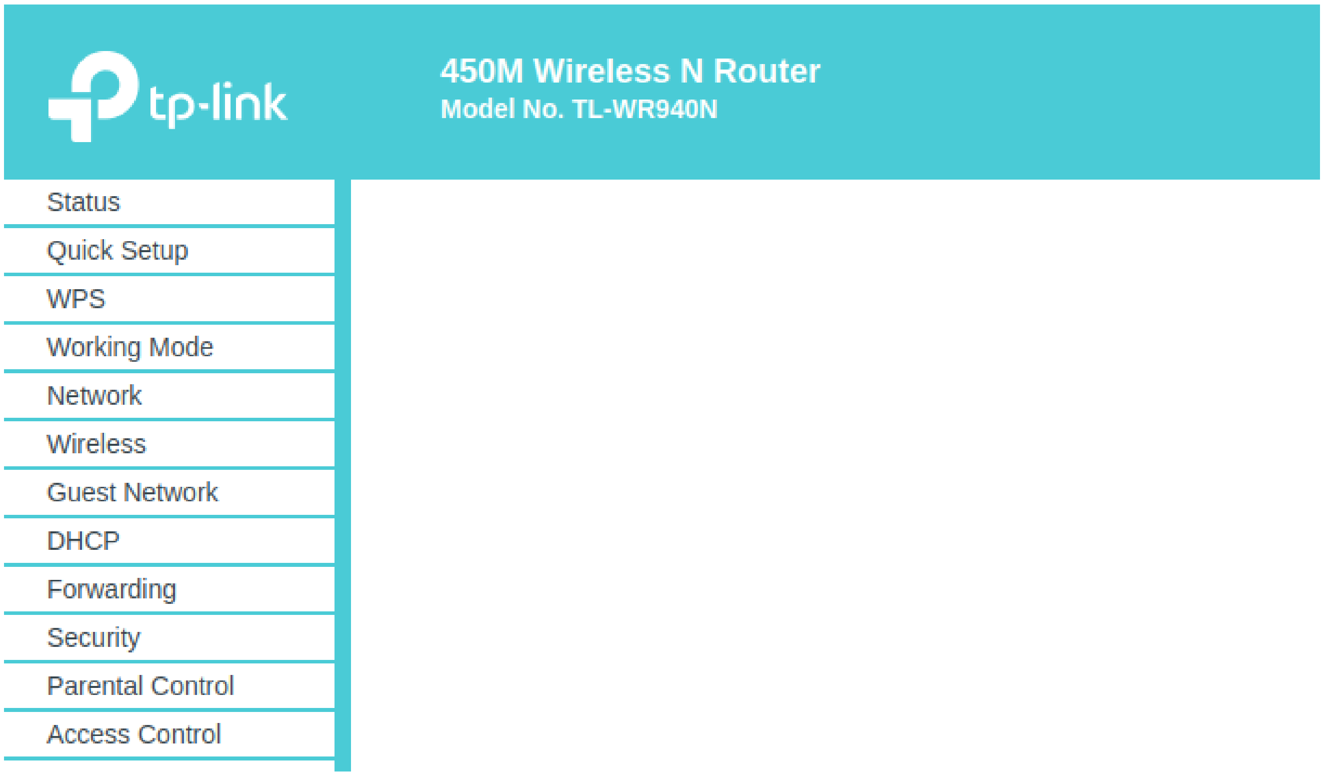


Figure 3: The malicious image hides from a user the ability to flash another firmware image

/etc/rc.d/rcS

The attackers modified the `/etc/rc.d/rcS` which is part of the operating systems' boot scripts. To this initialization script, the attackers added the following three shell commands to execute three of the files added to the modified firmware.

```
/usr/bin/udhcp &
/usr/bin/shell &
/usr/bin/timer 60 &
```

The `rcS` script is usually one of the first scripts to be executed during the system boot process, as it performs tasks that are essential to bringing up the rest of the system. Upon system boot-up, the `rcS` script would automatically launch all three binaries, thereby ensuring the persistence of the infection on the compromised device.

Analyzing the Added Files

By now, we saw that the attackers modified two files and added 4 files to the altered router firmware, 3 of them are executed by the modified initialization script. To understand what they do, we need to analyze each of the files. Since the router is a MIPS device, the binaries we'll analyze are all compiled for MIPS32BE architecture. Let's start.

shell — Passive Backdoor

The `shell` binary is a simple password-protected bind shell that will bind to all IPv4 network interfaces on port 14444. The password can be revealed with the highly advanced, exceedingly unique tool called `strings`.

Should you require the password, simply run the following command:

```
$ strings shell
[..]
password:
J2)3#[email protected]
success!
/bin/sh
[..]
```

As you can see, the password is hidden away in plain sight, waiting to be extracted by the adept researcher. With this information in hand, access to the elusive shell is granted, allowing for unrestricted entry into the system. May the force be with `strings`!

sheel

The `sheel` binary is a utility for configuration writing and reading. It was meant to be executed manually as it wasn't written to the modified init script. It reads and writes to the `/dev/mtdblock4` device. Why would it do so, a curious reader might ask? Before we answer this question, we first need to set the scene. The `/dev/mtdblock4` partition on this particular model of the router is in fact a so-called ART partition, which stands for Atheros Radio Test. It is supposed to contain calibration data for the WiFi chipset.

Curiously, the `sheel` binary uses this partition to store data in a raw format. And not just any data – its purpose is to write and read the C2 domains used by the main implant (`udhcp`) which is described further below. The obvious reason for writing data in a raw format on a block device is to make it less obvious to be spotted by a router administrator.

The `sheel` binary allows to write addresses of up to five C2 servers inside the partition. In case the operator didn't know how to use it, authors included helpful hint, even marking the optional arguments in brackets:

```
./sheel -h server_ip -p server_port -i update_index[0-4] [-r]
```

timer

The `timer` executable is a basic watchdog that is initiated during the boot process. It operates by attempting to execute the added `udhcp` executable at regular intervals, where the length of these intervals is determined by a number passed to it as a command line argument. The `udhcp` executable is the main implant in the modified firmware, as we will discuss shortly. When `udhcp` is launched, it verifies the presence of a file named `/var/udhcp`. If the file exists and is locked, `udhcp` terminates as it understands that another instance of itself is already running. However, if it does not exist, `udhcp` creates the file and writes its own process ID to it. The `timer` binary, by executing `udhcp` again and again, provides an additional layer of persistence, ensuring that the primary implant remains active.

The implementation is very simple, and as a reconstructed pseudo-code, it will look like this:

```
int32_t main(int32_t argc, char** argv, char** envp)
{
    daemon(1, 0);
    int32_t seconds;
    if (argc >= 2)
    {
        seconds = atoi(argv[1]);
    }
    else
    {
        seconds = 3600;
    }
    while (true)
    {
        sleep(seconds);
        system("/usr/bin/udhcp");
    }
}
```

Analyzing Horse Shell (udhcp)

The `udhcp` file is the main implant inserted into the modified firmware by the attackers. Parts of it are internally named `Horse Shell` so we use it to name the implant as a whole. The implant provides the attacker with 3 main functionalities: remote shell, file transfer, and tunneling.

In the following parts, we will dive deeper into the implementation of the different components, we'll explain the functionality of Horse Shell and how it is implemented.

Static Analysis

`udhcp` is a binary compiled for MIPS32 MSB operating system and written in C++. Many embedded devices and routers are running MIPS-based operating systems, and TP-Link routers are no different.

```
$ file ./udhcp
udhcp: ELF 32-bit MSB executable, MIPS, MIPS32 rel2 version 1 (SYSV), dynamically
linked, interpreter /lib/ld-uClibc.so.0, stripped
```

Even though the implant is not easy to analyze, the static information embedded in it makes the analysis a little bit simpler. In spite of it being shown as “stripped”, it is full of meaningful strings such as source file names, debug log messages, function names, names of global variables, and assert messages. Executing `strings` against the binary will reveal meaningful information that can give a researcher a good idea of what they’re dealing with.

Initializing

Horse Shell execution begins by instructing the system not to terminate it when receiving the `SIGPIPE`, `SIGINT` or `SIGABRT` signals. Then it calls a function named `horse_main` which is the main function of the implant. In this context, “horse” may refer to Trojan Horse.

Upon invocation, the implant issues a `daemon(1, 0)` call, which instructs the operating system to detach it from the controlling terminal and run it in the background as a daemon. It then verifies the existence of the file `/var/udhcp`. If the file exists, Horse Shell assumes that another instance of the implant is already running and immediately terminates. Conversely, if the file is non-existent, the implant creates it, setting its permissions to `rw-r--r--`. The newly created file then serves as a type of mutex that the Horse Shell writes the current PID to, helping to avoid concurrency issues.

The implant creates a file `/var/udhcp.cnf` and writes the command `kill -9 [PID]` to it, [PID] being udhcp’s process ID. It’s unclear how the file is used or what purpose it serves. One suggestion is that it can be used by the attackers to easily terminate the running implant.

Configuration

Most of Horse Shell’s configuration is hard coded. However, some of the entries are dynamically configurable. The instance obtained by us is using `m.cremessage[.]com` on port 80 as its default command and control server. It will write this domain to `/dev/mtdblock4`. For non-default peers, it reads a list of peer hosts from `/dev/mtdblock4`. On an actively infected device, this MTD block can contain values inserted into it by using the aforementioned `sheel` utility, or by old versions of the implant that were flashed to the device. It will resolve every host to its IP address and check if it’s up and running. If it does, it will continue the initialization of the configuration.

Horse Shell operates as a single-threaded application and adopts an event-driven methodology to direct its execution. It makes extensive use of the open-source library, `libev`, for I/O events and invokes callback functions in response to specific events. In essence, the program’s progression is dictated by the events that occur, hence analyzing the implant warrants consideration of the events and their associated callbacks. During the configuration initialization phase, it sets up various events and associates callbacks to respond to circumstances such as reading and writing to sockets or establishing a connection.

In its configuration, the implant stores information such as IPs and Port of the command and controls, swap initializes `libev` structures for network and timer events, cryptographic context, callbacks, pointers to important structures like a linked list that holds active connections, etc.

Initial Connection

Upon finishing configuring itself, Horse Shell will start a `ev_timer` structure that will trigger a callback function periodically. When triggered, the function will check when it was last executed, and send a heartbeat message to all the established connections.

Then, the implant will try to connect to the command and control. When the initial connection is successfully established, Horse Shell will send to the peer a list of information about the infected device. This information is being sent frequently and not only once. The information sent by the implant contains:

- User name
- System name
- OS version
- OS time
- CPU architecture
- Number of CPUs
- Total RAM
- IP address
- MAC address
- Features supported by the implant (remote shell, tunneling, file transfer)
- Number of active connections

Some of the information sent, such as support functionalities and CPU architecture, can suggest that the implant has other versions that support different devices (i.e. non MIPS devices) and a different set of functionalities.

Communication

Horse Shell communicates with its peers and server on a port specified for each of them individually. By default, it is using port 80 for communication. Regardless of the port, it uses HTTP communication with hard-coded HTTP headers. Every communication by the implant is encrypted using a custom or modified encryption scheme that is based on Substitution-Permutation Network. Every message is encrypted upon sending and decrypted when arrives at the implant.

A request sent from the implant will have this structure:

```
POST http:/[domain]/index.php HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml,
image/pjpeg, application/x-ms-xbap, application/x-shockwave-flash,
application/msword, application/vnd.ms-powerpoint, application/vnd.ms-excel, */*
Accept-Language: en-US, zh-CN;q=0.5
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; qdesk
2.4.1265.203; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729;
InfoPath.3)
Accept-Encoding: gzip, deflate
Host: [domain]
Connection: Keep-Alive
```

[encrypted message]

The hard-coded headers hasn't much to do with the actual data sent. In fact, searching this header online led us to see the exact same HTTP headers on several coding forums and repositories on Chinese websites like CSDN. The `Accept-Language` header field in all messages transmitted from the implant includes the language code `zh-CN`, except for one instance. This occurs when the implant sends its initial transmission message containing details about the compromised device, where the Chinese language code is absent from the request. Instead, the attackers have included the HTTP headers with `Accept-Language: en-US`. It is possible that the attackers intentionally omitted the language code from the request to conceal any indication of their identity that might be inferred from the language used.

Horse Shell is designed to communicate with numerous peers simultaneously. As it lacks multi-threading capabilities, the program employs list containers to segregate the various connected peers as individual list items. Each peer has a distinct structure, with assigned events and callbacks specific to it. This approach guarantees that the communication with each peer remains distinct, utilizing its unique callbacks and event handlers, and does not become intertwined with other peers.

The message structure differs between different types of communication conducted by the implant. Although the overall structure is similar, each functionality within the implant has its distinct nuances and particulars. For instance, the structure of communication related to tunneling will differ from that of a remote shell. However, the HTTP header in the requests remains consistent across all communication types.

Commands and Functionalities

Each functionality has its own list of supported commands. When a new connection is received, it will be parsed and handled by the callback function that handles read events triggered from the peer's socket. It will check if the packet is requesting to open a new type of connection from the following options:

Command	Subcommand	Description
0x1	0x2	Start remote shell ("Horse Shell")
0x2	0x2	Start SOCKS tunneling
0x3	0x2	Start file transfer

Remote Shell

When a peer requests to initiate a new remote shell instance, the program will verify the existence of `/bin/bash` or `/bin/sh` on the device. If either of them exists, the program will generate a new session using the `tsession` structure implementation from the Telnet open-source project. This Telnet-based connection provides the attacker with complete shell access to the compromised device.

It's important to note that the remote shell feature utilizes an embedded Telnet library while it still functions through the implant's HTTP-based communication. However, the communication between the compromised device and the peer seems unencrypted.

Supported commands

Command ID	Name	Description
0x1	REQ_CONNECT_PORT	Create a new shell connection

File Transfer

The file transfer module supports downloading and uploading files to and from the infected device, as well as basic file manipulation functionality.

This functionality is important as the attackers may need to upload new modules or tools onto a compromised system to perform specific tasks, such as conducting reconnaissance, stealing data, or moving laterally within a target network. These modules or tools may be customized for the specific target or scenario, and may not be present on the compromised system initially.

In addition, although not very useful for devices such as routers, the threat actors can use this module for data exfiltration or collect different logs from the device.

Supported Commands

Command ID	Name	Description
0x1	FILE_TRANSFER_REQ_CONNECT_PORT	Initiate connection

Command ID	Name	Description
0x2	FILE_TRANSFER_OPER_UPLOAD_CHECK	Check for active Upload task
0x3	FILE_TRANSFER_OPER_DOWNLOAD_CHECK	Check for active Download task
0x4	FILE_TRANSFER_OPER_QUERY	Query directory list
0x6	FILE_TRANSFER_OPER_DELETE	Delete a file from the device
0x7	FILE_TRANSFER_OPER_UPLOAD	Create a file on the device
0x8	FILE_TRANSFER_OPER_DOWNLOAD	Download a file from the device
0x9	FILE_TRANSFER_OPER_CHECK_EXISTS	Check if the file exists
0xa	FILE_TRANSFER_OPER_CANCEL_UPLOAD	Cancel Upload task
0xb	FILE_TRANSFER_OPER_CANCEL_DOWNLOAD	Cancel Download task
0xc	FILE_TRANSFER_TRANS_FILE_DATA	Write file contents to the device
0x14	REQ_MODULE_HEARTBEAT	Heartbeat

Tunneling

The implant can relay communication between two nodes. By doing so, the attackers can create a chain of nodes that will relay traffic to the command and control server. By doing so, the attackers can hide the final command and control, as every node in the chain has information only on the previous and next nodes, each node being an infected device. Only a handful of nodes will know the identity of the final command and control.

By using multiple layers of nodes to tunnel communication, threat actors can obscure the origin and destination of the traffic, making it difficult for defenders to trace the traffic back to the C2. This makes it harder for defenders to detect and respond to the attack.

In addition, a chain of infected nodes makes it harder for defenders to disrupt the communication between the attacker and the C2. If one node in the chain is compromised or taken down, the attacker can still maintain communication with the C2 by routing traffic through a different node in the chain.

Supported commands

Command ID	Name	Description
0x1	SOCKS_TUN_REQ_CONNECT_PORT	Check if the port is available for connection
0x4	SOCKS_TUN_NATPORT_COMM_CMD_OPEN	Open connection on port
0x5	SOCKS_TUN_NATPORT_COMM_CMD_CONNECT	Establish a connection between two nodes ip1:port1 <–> ip2:port2
0x6	SOCKS_TUN_NATPORT_COMM_CMD_DATA	Transfer data between connected nodes
0x7	SOCKS_TUN_NATPORT_COMM_CMD_DISCONNECT	Disconnect tunnel between two nodes
0x8	SOCKS_TUN_NATPORT_COMM_CMD_CLOSE	Mark tunnel as closed
0xa	SOCKS_TUN_NATPORT_COMM_CMD_CHECK	Check for new commands
0x14	SOCKS_TUN_REQ_MODULE_HEARTBEAT	Heartbeat

Characteristics

Router implants are not something very popular. Sure, there are infamous malware like Mirai and its numerous offshoots, and a handful of Linux-based botnets still lingering out there, but let’s be honest – it’s not exactly the most happening party in town.

However, in recent years we see Chinese threat actors’ increasing interest in compromising edge devices, aiming to both build resilient and more anonymous C&C infrastructures and to gain a foothold in certain targeted networks. In the following section, we list some interesting and unique development decisions taken by the Horse Shell developers and will compare them to another well-known implant used by Chinese espionage group APT31.

Usage of Open Source Projects

The implant smartly integrated multiple open-source libraries in its code. Its remote shell is based on Telnet, events are handled by `libev`, it has `libbase32` in it, `ikcp` too, and its list containers are based on TOR's `smartlist`, implementation. It might get inspiration from other projects such as `Shadowsocks-libev` and `udptun` for some of its functionality. Even its exact HTTP headers were taken from open-source repositories.

Structures and Event-driven flow

Horse Shell's functionality isn't groundbreaking, but certainly not run-of-the-mill either. However, its reliance on `libev` to create a complex event-driven program, and its penchant for complex structures and list containers, make our job of analyzing it all the more challenging. But, let's not mince words – the code quality is impressive, and the implant's ability to handle multiple tasks across a range of modules and structures demonstrates the kind of advanced skills that make us stand up and take notice.

Unused Code

The vast majority of the functions in the implant are being used. However, a thorough examination has revealed that there are certain functions and submodules that have been neglected and unused, like a lone sock lost in the laundry. We saw unused functions from the JSON and IKCP open-source libraries, custom functions built for UDP handling, and more.

While it's possible that these forsaken functionalities are simply leftovers from earlier versions or perhaps orphans that belong to other variants for different devices, their purpose remains a mystery to us.

Custom Crypto

Oh, the thrill of creating your very own cryptographic scheme! But alas, it's not typically the wisest endeavor. However, the daring individuals behind Horse Shell have forged ahead with a custom or tweaked encryption scheme, built upon Substitution-Permutation Network. This scheme is utilized by the implant to encrypt and decrypt the data it transmits and receives.

Despite this being far from a best practice, we must begrudgingly admit that our investigations have thus far failed to reveal any conspicuous flaws in the implementation.

Comparison to Other Implants

The Horse Shell implant is written in C++ and compiled for MIPS32-based operating systems. There aren't many implants written for network devices and so we went to look for other examples, to see if the implant we're looking at is a variant of an already known implant. Spoiling the surprise, we were unable to find another implant that we could confidently classify as a version of Horse Shell. Nonetheless, we did come across other implants that share some similarities and were also associated with Chinese state-sponsored actors. It remains unclear whether they are different variations of the same implant or not.

On July 2021, CERT-FR reported a large campaign conducted by the Chinese-affiliated threat actor APT31. They discovered that the actor used a mesh network of compromised routers orchestrated using malware they dubbed "Pakdoor". A follow-up report that was released in December 2021 shares more information about the campaign as well as a technical analysis of Pakdoor. Security researcher @imp0rtp3 thoroughly analyzed Pakdoor and share their great analysis on their blog.

Like Horse Shell, the Pakdoor implant also infects MIPS router devices, using event-driven execution flow based on `libev` and makes heavy use of structs and open-source libraries. It seems like the two implants are sharing the same goal of tunneling information between nodes as part of a chain of infected devices. The two also have the capability to act as a Remote Access Tool, providing the attacker with a remote shell on the infected device. The code itself, however, isn't similar between the two implants, although it has some mutual design and architectural decisions.

We don't know for sure whether the two implants were written by the same developers and we don't have evidence to suggest that this is the case. Pakdoor was used by APT31 and Horse Shell was seen in an operation by Camaro Dragon, two seemingly distinct groups.

Attribution

We found the Horse Shell implant while analyzing sophisticated attacks targeting officials in multiple European countries. The campaign leveraged a wide variety of tools, among them tools commonly associated with Chinese state-sponsored threat actors. The activity we analyzed has significant overlaps with activities publicly disclosed by Avast and Eset, linking it to the Chinese-affiliated APT group "Mustang Panda". We attribute this activity to a Chinese state-sponsored group we call Camaro Dragon. There is enough evidence to suggest that Camaro Dragon has significant overlaps with Mustang Panda, alas we can't say that this is a full overlap or that these two are the exact same group.

Following are some aspects worth paying attention to regarding the attribution of the tool.

Server and infrastructure

Not only that we found the implant on a server related to the Camaro Dragon activity, we also found out that the IP address (91.245.253[.]72) to which Horse Shell's C&C resolves to is listed on Avast's report on their analysis of the Mustang Panda campaign. Given the significant overlaps between Mustang Panda and the group we call Camaro Dragon, it is likely that the router implant was deployed by other campaigns of the group.

Chinese HTTP Request

We described how when Horse Shell transmits data from the infected device, they use a hard-coded HTTP headers. When we searched for this header online we found the exact same HTTP headers appearing on several Chinese websites like CSDN in what seems rather esoteric posts. We did not find the same headers on global forums and platforms such as GitHub or Stack Exchange. This suggests that the authors of the implant may have searched for these headers on Chinese forums or used Chinese search queries to arrive at these examples.

Typos

As we started analyzing Horse Shell, we understood very quickly that the binary is full of debug logs and string artifacts. When doing attribution we try to pay a lot of attention to the language used by the attackers on their implants. While overall the level of English in the implant was quite good, we did notice some typos, some of them repeated again and again across different functions and log strings in the binary. Some of them are:

- "**tatal** len" — instead of "total"
- "call file_get_http_**filed**" — instead of "field"
- "s_dbgMsg = "write pid **faile**." — instead of "failed"
- "file transfer download open file %s **fialed**!" — instead of "failed"
- "delete file:%s **fialed**,open **fialed** ret=%d" — instead of "failed"
- "**unkown** file transfer sub cmd" — instead of "unknown"
- "not enough **sapce** to save lan ipv4 and port" — instead of "space"
- "not enough **sapce** to malloc a port relay info!"— instead of "space"

Such mistakes can suggest the authors of the implant are not native English speakers as these mistakes should be very visible to developers with a higher level of written English.

Victims

Our investigation of the Camaro Dragon activity was of a campaign targeted mainly at European foreign affairs entities. However, even though we found Horse Shell on the attacking infrastructure, we don't know who are the victims of the router implant. Learning from history, router implants are often installed on arbitrary devices with no particular interest, with the aim to create a chain of nodes between main infections and real command and control. In other words, infecting a home router does not mean that the homeowner was specifically targeted, but rather that they are only a means to a goal.

Focus on Network Devices

Earlier in this report we discussed similarities between Horse Shell and another router MIPS implant called Pakdoor (or SoWat). Although the two share some commonalities, it is unclear whether one was developed from the other or if these are two distinct malware implants. Nevertheless, Pakdoor — being deployed by the Chinese state-sponsored group APT31 — together with other known instances of zero-day exploits and custom firmware and backdoors for routers and security gateways, demonstrates that such capabilities and types of attacks are of consistent interest and focus of Chinese-affiliated threat actors.

Detection and Protection

The discovery of Camaro Dragon's malicious implant on TP-Link routers highlights the need for individuals and organizations to take measures to protect themselves from similar attacks. Here are some protection and detection recommendations.

Network Protections

Horse Shell communicates with its peers using HTTP with hard-coded headers. Although the Headers were most likely copied from online forums, they are quite unique and can be used for the detection of communication from potentially infected devices. Traffic using this user agent is likely to be malicious. Use such detection signature with caution, as theoretically, it can block non-malicious traffic.

```
POST http://[host name]/index.php HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml,
image/pjpeg, application/x-ms-xbap, application/x-shockwave-flash,
application/msword, application/vnd.ms-powerpoint, application/vnd.ms-excel, */*
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; qdesk
2.4.1265.203; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729;
InfoPath.3)
Accept-Encoding: gzip, deflate
Host: [host name]
Connection: Keep-Alive
```

Software Updates

It's important to emphasize how important it is to keep your network devices' firmware version up-to-date. Make sure to regularly update routers and other devices' firmware and software to prevent vulnerabilities exploited by attackers.

Default Credentials

Always change the default login credentials of any device connected to the internet to stronger passwords and use multi-factor authentication whenever possible. Attackers are scanning the internet for devices that kept the default credentials of their device.

Check Point's network security solutions provide advanced threat prevention and real-time network protection against sophisticated attacks like those used by the Camaro Dragon APT group. This includes protection against exploits, malware, and other advanced threats. Check Point's **Quantum IoT Protect** automatically identifies and maps IoT devices and assesses the risk, prevents unauthorized access to and from IoT/OT devices with zero-trust profiling and segmentation and blocks attacks against IoT devices. Check Point's Threat Emulation detects these threats as `APT.Wins.HorseShell.A` and `APT.Wins.HorseShell.B`.

Conclusion

Our analysis of the Chinese state-sponsored APT group Camaro Dragon's attacks on European foreign affairs entities has uncovered a malicious firmware implant tailored for TP-Link routers. The implant features a custom backdoor called "Horse Shell" which enables the attackers to perform actions like remote shell, file transfer, and network tunneling, making it easier for them to anonymize their communication through a chain of infected nodes.

Through our investigation, we have gained a deeper comprehension of the ways in which attackers are employing malware to target edge devices, particularly routers. Our efforts have led us to uncover several of the tactics and tools utilized by Camaro Dragon in their attacks. Our findings not only contribute to a better understanding of the Camaro Dragon group and their toolset but also to the broader cybersecurity community, providing crucial knowledge for understanding and defending against similar threats in the future.

Furthermore, our discovery of the firmware-agnostic nature of the implanted components indicates that a wide range of devices and vendors may be at risk. We hope that our research will contribute to improving the security posture of organizations and individuals alike. In the meantime, remember to keep your network devices updated and secured, and beware of any suspicious activity on your network — you never know who might be lurking in the dragon's lair!

Check Point Customers Remain Protected

Check Point's network security solutions provide advanced threat prevention and real-time network protection against sophisticated attacks like those used by the Camaro Dragon APT group. This includes protection against exploits, malware, and other advanced threats. Check Point's **Quantum IoT Protect** automatically identifies and maps IoT devices and assesses the risk, prevents unauthorized access to and from IoT/OT devices with zero-trust profiling and segmentation, and blocks attacks against IoT devices.

Check Point IoT Embedded with Nano Agent® provides on-device runtime protection enabling connected devices with built-in firmware security. The Nano Agent® is a customized package which provides the top security capabilities and prevents malicious activity on routers, network devices and other IoT devices. Check Point IoT Nano Agent® has advanced capabilities of memory protection, anomaly detection, and control flow integrity. It operates inside the device, and serves as a frontline to secure IoT devices.

Appendix A – IOCs

SHA256	File Name
998788472cb1502c03675a15a9f09b12f3877a5aeb687f891458a414b8e0d66c	udhcp
7985f992dcc6fcce76ee2892700c8538af075bd991625156bf2482dbfebd5a5a	sheel
ed3d667a4fa92d78a0a54f696f4e8ff254def8d6f3208e6fe426dbe7fb3f3dd0	shell
66cc81a7d865941cb32ed7b1b84b20270d7d667b523cab28b856cd4e85f135b6	timer
8a2e9f6c2b0c898090fdce021b3813313e73a256a5de39c100bf9868abc09dbb	9406.dat
da046a1fe6f3b94e48c24ffd341f8d97bfc06252ddf4d332e8e2478262ad1964	9404.dat

Written Files

File Name	Description
/vat/udhcp.cnf	Contains kill -9 [pid] command that has the pid of the running implant
/var/udhcp	A mutex like file that will be created when the implant is running.
.remote_shell.log	Log file of the remote shell functionality of the implant

Infrastructure

IoC	Description
m.cremessage[.]com	Command and Control
91.245.253[.]72	Hosts TPLink implant C2 domain m[.]cremessage[.]com

Appendix B: Yara Signatures

```
rule apt_CN_CamaroDragon_horseshell_strings {
        meta:
                author = "Itay Cohen @ Check Point Research"
                date = "2023-04-01"
                description = "Detects CamaroDragon's HorseShell implant for routers
based on embedded strings. This rule is broad."
                hash =
"998788472cb1502c03675a15a9f09b12f3877a5aeb687f891458a414b8e0d66c"
                reference = ""
        strings:
                // Crypto
                $crypto_1 = "wzsw_srand"
                $crypto_2 = "wzsw_rand"
                $crypto_3 = "wzsw_init"
                $crypto_4 = "wzsw_crypto_free"
                $crypto_5 = "wzsw_crypto_new"
                $crypto_6 = "wzsw_encrypt_buf"
                $crypto_7 = "wzsw_crypto_reset"


                // File names
                $filename_1 = "common/wzsw_crypto.c"
                $filename_2 = "http/http_socks_tun.cc"
                $filename_3 = "http/http_trans_file.cc"
                $filename_4 = "http/http_horse_shell.cc"
                $filename_5 = "http/http_online.cc"

                // Debug strings
                $debug_1 = "add_file_transfer_info_to_list"
                $debug_2 = "before trans data need connect"
                $debug_3 = "before trans data need connect2"
                $debug_4 = "cancel current task!"
                $debug_5 = "cancel task from task list!"
                $debug_6 = "cancel task task id"
                $debug_7 = "check file file_type is %d, neither file nor dir !"
                $debug_8 = "check_file_transfer_conn_heart_beat"
                $debug_9 = "check_module_heart_beat"
                $debug_10 = "check_socks_tun_conn_heart_beat"
                $debug_11 = "conn_marked_close"
                $debug_12 = "conn_peek socket (sock=%d) closed"
                $debug_13 = "conn_peek socket (sock=%d) recv error,err=%d"
                $debug_14 = "conn_read socket (sock=%d) closed"
                $debug_15 = "conn_read socket (sock=%d) recv error,err=%d"
                $debug_16 = "conn_readfrom socket (sock=%d) recv error,err=%d"
                $debug_17 = "connect lan '%s:%d' in progress!"
                $debug_18 = "create_shell_conn_session"
                $debug_19 = "create_shell_pty_session"
                $debug_20 = "current task %p is not download!"
                $debug_21 = "file transfer download open file %s succeed!"
                $debug_22 = "file transfer oper %d neither download nor upload"
                $debug_23 = "file transfer process connect port cmd->port"
                $debug_24 = "file transfer upload open file %s succeed"
```

```
$debug_25 = "file_transfer_data_request...cur_len=%d"
$debug_26 = "file_transfer_data_request...SEND_RSP"
$debug_27 = "file_transfer_data_request"
$debug_28 = "file_transfer_delete_request"
$debug_29 = "file_transfer_download_request"
$debug_30 = "file_transfer_free_cb"
$debug_31 = "file_transfer_info port %d already started!"
$debug_32 = "file_transfer_process_connect_port"
$debug_33 = "file_transfer_query_request"
$debug_34 = "file_transfer_send_file_data"
$debug_35 = "file_transfer_upload_request"
$debug_36 = "find ipv4=%s, port=%d by lan success!"
$debug_37 = "find_connected_port_relay_info"
$debug_38 = "find_port_relay_info_by_lan"
$debug_39 = "find_start_connect_port_relay_info"
$debug_40 = "free file transfer info!"
$debug_41 = "get_file_transfer_info_by_conn"
$debug_42 = "get_file_transfer_info_by_port"
$debug_43 = "get_free_port_relay_info"
$debug_44 = "get_port_relay_info_from_list_by_port_relay_conn"
$debug_45 = "get_socks_tun_info_from_list_by_conn"
$debug_46 = "get_socks_tun_info_from_list_by_port"
$debug_47 = "horse_shell_start"
$debug_48 = "http online data length %d too long"
$debug_49 = "http_rsp:%s"
$debug_50 = "info->state=%d not connected!"
$debug_51 = "invalid NATPORT_COMM_CMD_CONNECT"
$debug_52 = "invalid NATPORT_COMM_CMD_DISCONNECT"
$debug_53 = "malloc file transfer info!"
$debug_54 = "neither cancel upload nor cancel download"
$debug_55 = "not connected, begin to connect!"
$debug_56 = "not enough sapce to malloc a port relay info!"
$debug_57 = "other file transfer task oper %d"
$debug_58 = "recv check file '%s' exists request, file %s"
$debug_59 = "remove current task!"
$debug_60 = "remove_file_transfer_info_from_list"
$debug_61 = "reverse_shell can not find bash or sh"
$debug_62 = "shell create connection to %s:%d succeed"
$debug_63 = "shell create_shell_pty_session succeed!"
$debug_64 = "shell process connect port cmd->port"
$debug_65 = "shell_pty_session_free_cb, socket=%d"
$debug_66 = "socks tun connect lan"
$debug_67 = "socks tun create connection %p to %s:%d succeed!"
$debug_68 = "socks tun port %d already opened!"
$debug_69 = "socks tun process connect port"
$debug_70 = "socks tun try connect lan"
$debug_71 = "trans file create connection to %s:%d succeed!"
$debug_72 = "trans_file_start"
$debug_73 = "tun_info->free_list"
$debug_74 = "tun_info->used_list"
$debug_75 = "unkown file transfer sub cmd"
$debug_76 = "unkown socks tun sub cmd"
```

```
$debug_77 = "shell_conn_session_connect_cb"
$debug_78 = "shell_conn_session_free_cb"
$debug_79 = "shell_get_body_from_http_rsp"
$debug_80 = "shell_get_http_body"
$debug_81 = "shell_get_http_filed"


// Commands
$command_1 = "SOCKS TUN REQ_CONNECT_PORT"
$command_2 = "SOCKS TUN NATPORT_COMM_CMD_CONNECT"
$command_3 = "SOCKS TUN NATPORT_COMM_CMD_OPEN"
$command_4 = "SOCKS TUN NATPORT_COMM_CMD_DATA"
$command_5 = "SOCKS TUN NATPORT_COMM_CMD_CLOSE"
$command_6 = "SOCKS TUN NATPORT_COMM_CMD_DISCONNECT"
$command_7 = "SOCKS TUN NATPORT_COMM_CMD_CHECK"
$command_8 = "SOCKS TUN REQ_MODULE_HEARTBEAT"
$command_9 = "NET_REQ_HORSE_SHELL REQ_CONNECT_PORT"
$command_10 = "FILE_TRANSFER REQ_CONNECT_PORT"
$command_11 = "FILE_TRANSFER_OPER_DOWNLOAD"
$command_12 = "FILE_TRANSFER_OPER_UPLOAD"
$command_13 = "FILE_TRANSFER_OPER_CANCEL_DOWNLOAD"
$command_14 = "FILE_TRANSFER_OPER_CANCEL_UPLOAD"
$command_15 = "FILE_TRANSFER_TRANS_FILE_DATA"
$command_16 = "FILE_TRANSFER_OPER_DELETE"
$command_17 = "FILE_TRANSFER_OPER_QUERY"
$command_18 = "FILE_TRANSFER_OPER_CHECK_EXISTS"
$command_19 = "REQ_MODULE_HEARTBEAT"

// Error strings
$error_1 = " > .remote_shell.log"
$error_2 = "calloc mem for hall_device_online_req failed!"
$error_3 = "conn_listening_conn bind sock=%d failed"
$error_4 = "conn_listening_conn listen sock=%d failed"
$error_5 = "conn_listening_conn set sock=%d address failed"
$error_6 = "conn_listening_conn set sock=%d nonblock failed"
$error_7 = "conn_listening_conn set sock=%d reuse address failed"
$error_8 = "conn_listening_conn set sock=%d tcp no delay failed"
$error_9 = "conn_tcp_conn connect sock=%d failed,err=%d"
$error_10 = "conn_tcp_conn set sock=%d address failed,err=%d"
$error_11 = "conn_tcp_conn set sock=%d nonblock failed,err=%d"
$error_12 = "conn_tcp_conn set sock=%d reuse address failed,err=%d"
$error_13 = "conn_tcp_conn set sock=%d tcp no delay failed,err=%d"
$error_14 = "conn_udp_conn bind sock=%d failed"
$error_15 = "conn_udp_conn set sock=%d address failed"
$error_16 = "conn_udp_conn set sock=%d nonblock failed"
$error_17 = "conn_udp_conn set sock=%d reuse address failed"
$error_18 = "create file_transfer_info failed!"
$error_19 = "create g_file_transfer_info_list failed!"
$error_20 = "create g_socks_tun_list failed!"
$error_21 = "create shell conn session failed!"
$error_22 = "crypto_decrypt_buf error,"
$error_23 = "delete file:%s fialed"
```

```
                $error_24 = "disconnect this http conn"
                $error_25 = "download file %s failed, safe read length=%d from fd=%d
failed, retlen=%d!"
                $error_26 = "file transfer download open file %s fialed!"
                $error_27 = "file transfer process connect port failed, ret = %d!"
                $error_28 = "file transfer upload open file %s failed"
                $error_29 = "find connected ipv4_port ipv4=%s, port=%d failed!"
                $error_30 = "find connected port_relay_info ipv4=%s, port=%d failed!"
                $error_31 = "find ipv4=%s, port=%d by lan failed!"
                $error_32 = "get a free port_relay_info  failed , not enough sapce to
save lan ipv4 and port"
                $error_33 = "http connect failed, errno=%d, reason=%s"
                $error_34 = "http data length %d too long"
                $error_35 = "init conf failed"
                $error_36 = "open fialed ret=%d"
                $error_37 = "peek http rsp failed!"
                $error_38 = "peek socks tun data from %s:%d sock=%d failed!"
                $error_39 = "read ip failed, %m"
                $error_40 = "resolve online domain failed"
                $error_41 = "send data to dst socket %d failed!"
                $error_42 = "send failed (sock=%d,err=%d)"
                $error_43 = "sendto failed (peer address=%s,err=%d)"
                $error_45 = "set socket opt failed, %m!"
                $error_46 = "shell create connection to %s:%d failed!"
                $error_47 = "shell create_shell_pty_session failed!"
                $error_48 = "shell process connect port failed, ret = %d!"
                $error_49 = "socks tun connect lan '%s:%d' failed!"
                $error_50 = "socks tun create connection to %s:%d failed!"
                $error_51 = "socks tun port %d already opened!"
                $error_52 = "socks tun process connect port failed, ret = %d!"
                $error_53 = "socks tun try connect lan '%s:%d' failed!"
                $error_54 = "socks_tun_connect_cb failed %p"
                $error_55 = "socks_tun_info_new failed!"
                $error_56 = "start shell '%s' failed!"
                $error_57 = "tcp online create conn failed!"
                $error_58 = "tcp online create http_online_info_t failed!"
                $error_59 = "trans file create connection to %s:%d failed!"
                $error_60 = "try connect failed"
                $error_61 = "try connect lan '%s:%d' failed!"
                $error_62 = "wzsw_init failed"

                // function names
                $function_1 = "shell_conn_session_connect_cb"
                $function_2 = "shell_conn_session_free_cb"
                $function_3 = "shell_get_body_from_http_rsp"
                $function_4 = "shell_get_http_body"
                $function_5 = "shell_get_http_filed"
                $function_6 = "send_file_transfer_conn_heart_beat"
                $function_7 = "send_file_transfer_http_data_net_packet"
                $function_8 = "send_horse_shell_http_data_net_packet"
                $function_9 = "send_module_heart_beat"
                $function_10 = "send_socks_tun_cmd_info_packet"
```

```
$function_11 = "send_socks_tun_conn_heart_beat"
$function_12 = "send_socks_tun_http_data_net_packet"
$function_13 = "send_socks_tun_net_packet"
$function_14 = "send_socks_tun_status_packet"
$function_15 = "socks_get_body_from_http_rsp"
$function_16 = "socks_get_http_body"
$function_17 = "socks_get_http_filed"
$function_18 = "socks_tun_conn_cb"
$function_19 = "socks_tun_conn_free_cb"
$function_20 = "socks_tun_conn_lan_cb"
$function_21 = "socks_tun_conn_lan_free_cb"
$function_22 = "socks_tun_connect_cb"
$function_23 = "socks_tun_connect_lan_cb"
$function_24 = "socks_tun_info_free"
$function_25 = "socks_tun_info_new"
$function_26 = "socks_tun_process_check"
$function_27 = "socks_tun_process_connect_port"
$function_28 = "socks_tun_process_connect"
$function_29 = "socks_tun_process_data"
$function_30 = "socks_tun_process_disconnect"
$function_31 = "socks_tun_process_open"
$function_32 = "socks_tun_start"
$function_33 = "socks_tun_try_connect_lan_port_cb"
$function_34 = "check_file_transfer_conn_heart_beat"
$function_35 = "check_socks_tun_conn_heart_beat"
$function_36 = "conn_init_tcp_conn"
$function_37 = "conn_marked_close"
$function_38 = "conn_read_buffer_length"
$function_39 = "conn_set_callback"
$function_40 = "conn_set_free_callback"
$function_41 = "conn_set_user_data"
$function_42 = "conn_start_read"
$function_43 = "conn_start_write"
$function_44 = "conn_stop_write"
$function_45 = "conn_tcp_conn"
$function_46 = "conn_uninit_containers"
$function_47 = "find_connected_port_relay_info"
$function_48 = "find_port_relay_info_by_lan"
$function_49 = "find_start_connect_port_relay_info"
$function_50 = "get_free_port_relay_info"
$function_51 = "get_port_relay_info_from_list_by_port_relay_conn"
$function_52 = "get_port_relay_info"
$function_53 = "horse_main"
$function_54 = "process_dev_online"
$function_55 = "process_http_read_events"
$function_56 = "process_shell_conn_session_read_events"
$function_57 = "process_shell_conn_session_write_events"
$function_58 = "put_port_relay_info"
$function_59 = "send_file_transfer_conn_heart_beat"
$function_60 = "send_socks_tun_conn_heart_beat"
$function_61 = "process_file_transfer_read_events"
$function_62 = "process_file_transfer_write_events"
```

```
                $function_63 = "process_pty_conn_read_events"
                $function_64 = "process_pty_conn_write_events"
                $function_65 = "process_shell_conn_connect_port"
                $function_66 = "process_shell_conn_session_read_events"
                $function_67 = "process_shell_conn_session_write_events"
                $function_68 = "process_socks_tun_lan_conn_read_events"
                $function_69 = "process_socks_tun_read_events"
                $function_70 = "process_transfile_task"
                $function_71 = "put_port_relay_info"
                $function_72 = "socks_get_body_from_http_rsp"
                $function_73 = "socks_get_http_body"
                $function_74 = "socks_get_http_filed"
                $function_75 = "socks_tun_conn_cb"
                $function_76 = "socks_tun_conn_free_cb"
                $function_77 = "socks_tun_conn_lan_cb"
                $function_78 = "socks_tun_conn_lan_free_cb"
                $function_79 = "socks_tun_connect_cb"
                $function_80 = "socks_tun_connect_lan_cb"
                $function_81 = "socks_tun_info_free"
                $function_82 = "socks_tun_info_new"
                $function_83 = "socks_tun_process_check"
                $function_84 = "socks_tun_process_connect_port"
                $function_85 = "socks_tun_process_connect"
                $function_86 = "socks_tun_process_data"
                $function_87 = "socks_tun_process_disconnect"
                $function_88 = "socks_tun_process_open"
                $function_89 = "socks_tun_start"
                $function_90 = "socks_tun_try_connect_lan_port_cb"

                // Globals
                $global_1 = "g_socks_tun_list"


        condition:
                filesize < 2MB and
                3 of ($crypto_*) or
                2 of ($filename_*) or
                3 of ($debug_*) or
                any of ($command_*) or
                3 of ($error_*) or
                3 of ($function_*) or
                $global_1 or
                5 of them
}

rule apt_CN_CamaroDragon_sheel_strings {
        meta:
                author = "Itay Cohen @ Check Point Research"
                date = "2023-04-01"
                description = "Detects CamaroDragon's sheel tool."
                hash =
"7985f992dcc6fcce76ee2892700c8538af075bd991625156bf2482dbfebd5a5a"
```

```
                reference = ""

        strings:
                $ = "write failed.open fail."
                $ = "open fail.%m"
                $ = "./sheel -h server_ip -p server_port -i update_index[0-4] [-r]"
                $ = "./sheel -h"
                $ = "update server list success!"

        condition:
                filesize < 12KB and
                3 of them
}
```

GO UP
BACK TO ALL POSTS