vb 2023 LONDON

4 - 6 October, 2023 / London, United Kingdom

# THE DRAGON WHO SOLD HIS CAMARO: REVERSING A CUSTOM ROUTER IMPLANT

Itay Cohen & Radoslaw Madej

*Check Point, Israel*

itayc@checkpoint.com
radoslawm@checkpoint.com

## ABSTRACT

During 2023 *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 exploiting Internet-facing network devices and modifying their underlying software or firmware. This paper will delve into the intricate details of analysing 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.


### *Key findings*

- We have discovered and analysed 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 the attackers with a shell to infected devices was found.
- 'Horse Shell', the main implant inserted into the modified firmware by the attackers, provides the attackers with three main functionalities.
- Due to its firmware-agnostic design, the implant's components can be integrated into various firmware from different vendors.
- The deployment method of the firmware images on the infected routers is still unclear, as are its usage and involvement in actual intrusions.

## BACKGROUND

Since January 2023 we have been 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 [1] and [2], linking it to 'Mustang Panda'. This cluster of activity is currently tracked by *Check Point Research* 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: two modified *TP-Link* router firmware images. As we dug further, it became evident they were tampered 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 can be included in different firmware from 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 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 showed 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, naturally 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 files we had obtained, analysing 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 uses a custom implementation of SquashFS. To extract the filesystem we used sasquatch [3].

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

Indeed, we found that multiple files had been added to the firmware we obtained, and a couple of files had been modified:

The following files had been added:

/usr/bin/sheel

/usr/bin/shell

/usr/bin/timer

/usr/bin/udhcp

The following files had been modified:

/etc/rc.d/rcS

/web/userRpm/SoftwareUpgradeRpm.htm

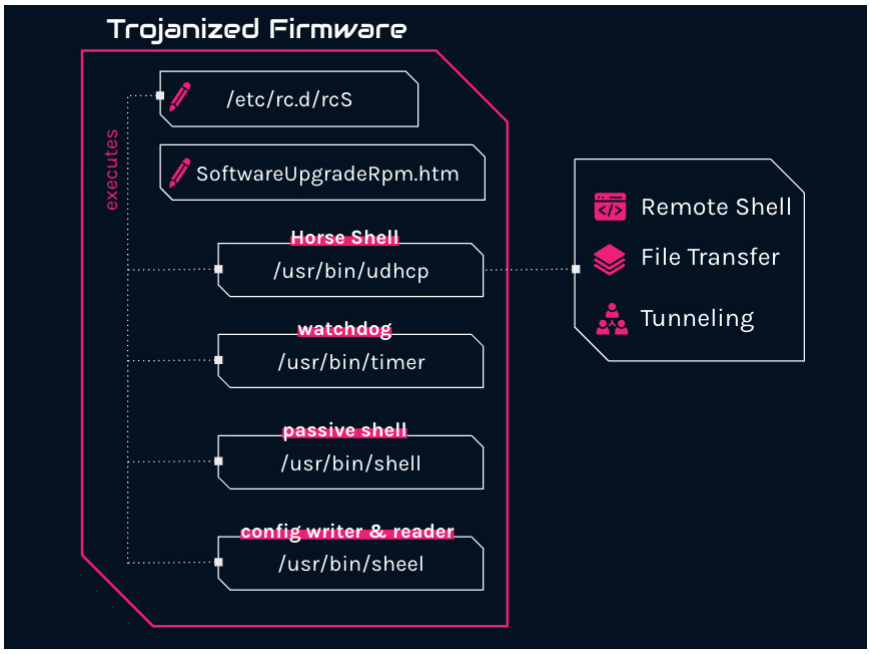


*Figure 1: Overview of the different components in the malicious implant.*

## 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 either by scanning them for known vulnerabilities or by 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 this is the case, they would likely be installing the implant on arbitrary devices with no particular interest in them.

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

## INSPECTING THE MODIFIED FILES

### SoftwareUpgradeRpm.htm

Like many routers, the *TP-Link* router has a web interface that allows its users to configure and manage it. One of the features the management website provides the user with is 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 Figure 2.

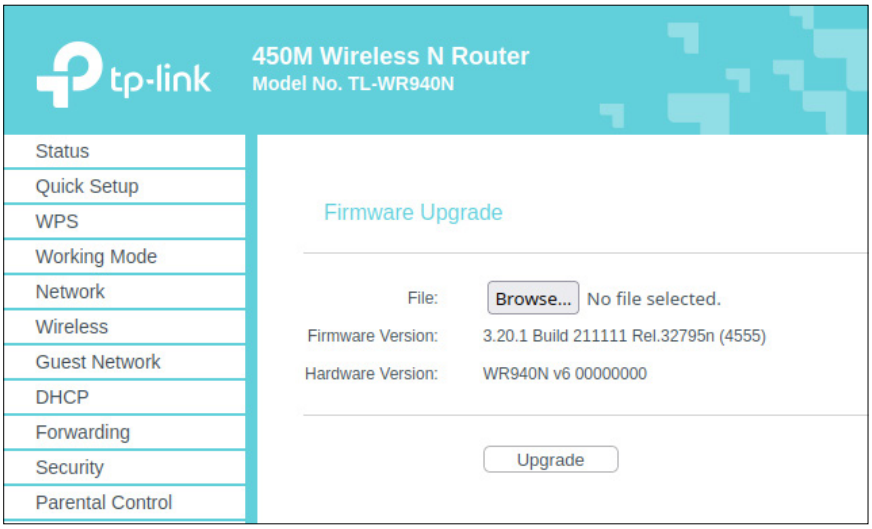


*Figure 2: SoftwareUpgradeRpm.htm as shown in the original firmware interface.*

However, in the modified version of the firmware we obtained, a small CSS property was added inline 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. However, now it will be harder to perform the upgrade or even know that this feature exists.

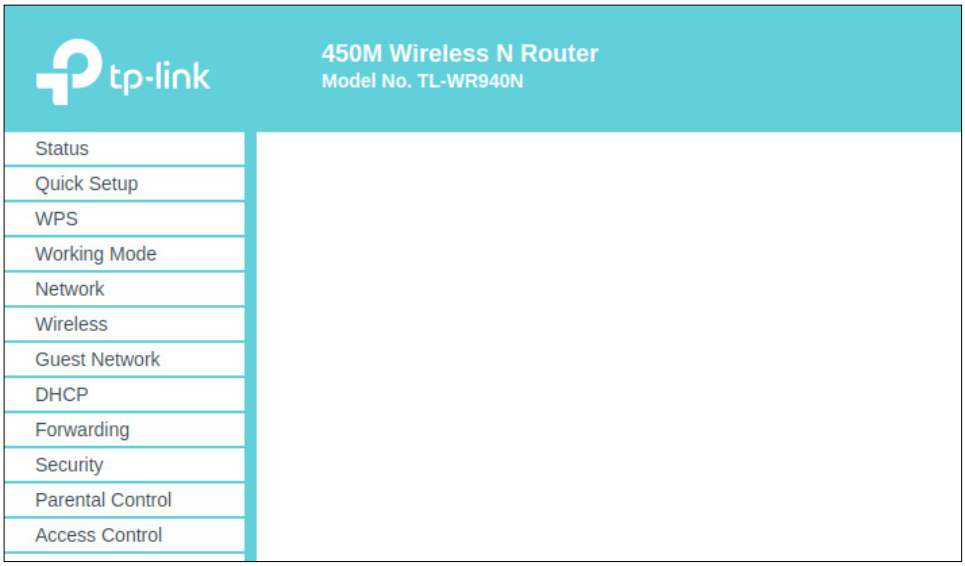


*Figure 3: The malicious image hides from a user the ability to flash another firmware image.*

### /etc/rc.d/rcS

The attackers modified the `/etc/rc.d/rcS`, which is part of the operating system's 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.

## ANALYSING THE ADDED FILES

By now, we could see that the attackers had modified two files and added four files to the altered router firmware, three of which are executed by the modified initialization script. To understand what they do, we need to analyse each of the files. Since the router is a MIPS device, the binaries we'll analyse 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#4G@Iie
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? 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 [4], a so-called ART partition, which stands for Atheros Radio Test [5]. It is supposed to contain calibration data for the Wi-Fi 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 likely to be spotted by a router administrator.

The `sheel` binary allows the addresses of up to five C2 servers to be written inside the partition. In case the operator didn't know how to use it, the authors included a 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 those 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 looks 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");
    }
}
```

## ANALYSING 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 this name for the implant as a whole. The implant provides the attacker with three main functionalities: remote shell, file transfer, and tunneling.

In the following sections 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 systems and written in C++. Many embedded devices and routers run 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 analyse, 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 could 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 uses `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 is, 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 analysing 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 an `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 a list of information about the infected device to the peer. This information is 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, may 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 uses 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 a substitution-permutation network. Every message is encrypted upon sending and decrypted when it arrives at the implant.

A request sent from the implant will have the following 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 don't have much to do with the actual data sent. In fact, researching these headers online led us to see the exact same HTTP headers on several coding forums and repositories on Chinese websites like *CSDN* [6]. The `Accept-Language` header field includes the language code `zh-CN` in all messages transmitted from the implant, with the exception of one instance. When the implant sends its initial transmission message containing details about the compromised device, the Chinese language code is absent from the request. Instead, the attackers include the HTTP header with `Accept-Language: en-US`. It is possible that the attackers intentionally omitted the language code from the request in an attempt to avoid any clues about 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 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 the opening of 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 initiation of a new remote shell instance, the program will check for 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 the downloading and uploading of 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 |
| 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 this, the attackers can hide the final command-and-control, as every node in the chain has information only about 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, the 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 very popular. Sure, there are infamous pieces of 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 have seen an increasing interest among Chinese threat actors in compromising edge devices, aiming to build both 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 compare them with another well-known implant used by Chinese espionage group APT31.

### *Usage of open-source projects*

The implant smartly integrates multiple open-source libraries in its code. Its remote shell is based on Telnet, events are handled by `libev`, it has `libbase32` in it, as well as `ikcp`, 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 ground breaking, but it's certainly not run-of-the-mill either. Its reliance on `libev` to create a complex event-driven program, and its penchant for complex structures and list containers, make our job of analysing it all the more challenging. 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 used. However, a thorough examination has revealed that there are certain functions and submodules that have been neglected and remain unused, like a lone sock lost in the laundry. There are 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! Alas, it's not typically the wisest endeavour. However, the daring individuals behind Horse Shell have forged ahead with a custom or tweaked encryption scheme, built upon a

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 begrudgingly admit that our investigations have, thus far, failed to reveal any conspicuous flaws in the implementation.

### *Comparison with 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, so we went to look for other examples, to see if the implant we're looking at is a variant of an already known one. 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 or not they are different variations of the same implant.

On 21 July 2021, CERT-FR reported [7] 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 [8] that was released in December 2021 shares more information about the campaign as well as a technical analysis of Pakdoor [9]. Security researcher @imp0rtp3 thoroughly analysed Pakdoor and shared a great analysis on their blog [10].

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 that the two implants share the same goal of tunneling information between nodes as part of a chain of infected devices. The two also both 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 they share some common 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 analysing 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 analysed 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, but we can't say that there is a full overlap or that the two are the exact same group.

The following subsections outline some aspects worth paying attention to regarding the attribution of the tool.

### *Server and infrastructure*

Not only did we find 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 is listed in *Avast*'s report [1] on 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, hard-coded HTTP headers are used. When we searched for this these headers online we found the exact same headers appearing on several Chinese websites, such as *CSDN* [6], in what seemed 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 analysing Horse Shell, we understood very quickly that the binary is full of debug logs and string artifacts. When considering attribution we try to pay a lot of attention to the language used by the attackers in their implants. While overall the level of English in the implant was quite good, we did notice some typos, some of which were repeated again and again across different functions and log strings in the binary. Some examples 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 might 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 focused on 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 the victims of the router implant are. 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 an end 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 consistent with the 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. The following 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 the firmware version of network devices up to date. The firmware and software of routers and other devices should be updated regularly 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 retain the default credentials.

## 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!

## REFERENCES


[1] Avast Threat Intelligence Team. Hitching a ride with Mustang Panda. Decoded avast.io. 2 December 2022. https://decoded.avast.io/threatintel/apt-treasure-trove-avast-suspects-chinese-apt-group-mustang-panda-is-collecting-data-from-burmese-government-agencies-and-opposition-groups/.

[2] Côté Cyr, A. MQsTTang: Mustang Panda's latest backdoor treads new ground with Qt and MQTT. WeLiveSecurity by ESET. 2 March 2023. https://www.welivesecurity.com/2023/03/02/mqsttang-mustang-panda-latest-backdoor-treads-new-ground-qt-mqtt/.

[3] Sasquatch. https://github.com/devttys0/sasquatch.

[4] OpenWrt. Debug output v1. https://openwrt.org/toh/tp-link/tl-wr940n#debug_output_v1.

[5] CodeFetch. Collection of Atheros Radio Test dumps. https://github.com/CodeFetch/art-collection.

[6] zerokkqq. Linux下发送HTTP协议请求. CSDN. https://blog.csdn.net/zerokkqq/article/details/79147360.

[7] CERT-FR. INDICATEURS DE COMPROMISSION DU CERT-FR. Objet: FR/GB [Maj] Campagne d'attaque du mode opératoire APT31 ciblant la France. 21 July 2021. https://www.cert.ssi.gouv.fr/ioc/CERTFR-2021-IOC-003/.

[8] CERT-FR. RAPPORT MENACES ET INCIDENTS DU CERT-FR. Objet: GB APT31 Intrusion set campaign: description, countermeasures and code. 15 December 2021. https://www.cert.ssi.gouv.fr/cti/CERTFR-2021-CTI-013/.

[9] CERT-FR. APT31: Pakdoor Technical Report. https://www.cert.ssi.gouv.fr/uploads/CERTFR-2021-CTI-013b.pdf.

[10] @imp0rtp3. https://imp0rtp3.wordpress.com/.


## 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 |