APT31: Pakdoor

TECHNICAL REPORT

TLP:WHITE

Version: 2.1
Registration date: 2021-12-15
Number of pages: 16

Contents

1 Introduction **3**

2 Compromission vector **4**

3 Launcher **5**

4 Backdoor **6**
4.1 Configuration 6
4.2 Communication 7
4.2.1 Initialization 7
4.2.2 Peers management 7
4.2.3 Asynchronous event handlers 7
4.2.4 Messages structure 8
4.3 Tasks 9
4.3.1 Peer discovery 9
4.3.2 Traffic relay 10
4.3.3 Administration 13

5 Conclusion **14**

A Hashes **15**
A.1 Backdoor 15

1 Introduction

In early 2021, several reports indicated that APT31 was compromising SOHO (Small Office Home Office) routers, especially some manufactured by Pakedge[1], to build its Command & Control (C&C) infrastructure. More recently, it was observed that routers of other brands like Cyberoam and Cisco were also compromised by APT31.

In order to remotely manage compromised routers and to make them communicate with each other, the attacker installs a sophisticated backdoor on them. As the latter seemed to be unknown publicly, we internally named it **Pakdoor**. The main purpose of this implant is to provide a dedicated anonymization service to APT31.

Observations have shown that APT31 uses this infrastructure both for recon and for communication with compromised targets, and thus data exfiltration. Moreover, compromised routers are divided in different pools where a router belonging to one pool can only communicate with routers in the same pool.

This document attempts to detail the inner workings of Pakdoor and its different components, and thus, to give a closer look at one of the C&C infrastructures of APT31. Also, indicators of compromission are available at the end of this report.

[1] https://www.pakedge.net

2 Compromission vector

When a router is compromised, the attackers drop at least three files on the filesystem:

- an ELF executable (written in C language) that implements most of the features of Pakdoor, hereinafter referred as the **backdoor**;
- a bash script responsible for continuously executing the backdoor and providing extra features to the latter, hereinafter referred as the **launcher**;
- an encrypted configuration file containing the different peers which the implant can communicate with.

3 Launcher

This script is executed by the `/etc/rc.local` script, which can execute commands after system services are loaded at startup.

The purpose of this script is to ensure the persistence of the backdoor, but also to provide some extra features (see Table 1) to the latter, such as the capacity to open ports in the local firewall (`Netfilter` in this case). Moreover, two certificates and one private key are hardcoded in the script. Both are used by the backdoor to setup TLS sockets (see Section 4.2 for more details).

The script takes the name of the command to execute as a command-line argument (six are implemented) and one more parameter if the command requires it.

Also, two paths are hardcoded at the beginning of the script in the following variables:

- `file_name`, path to the backdoor binary;
- `port_file`, path to a text file containing the port numbers used by the backdoor to relay traffic.

The Table 1 sums up the purpose of each command implemented in the script.

Command	Description
add	Adds an `iptables` rule to allow network traffic to the port number given as second parameter.
del	Deletes the `iptables` rule allowing network traffic to the port number given as second parameter. If the latter is not specified, all the rules corresponding to the ports present in the port file are deleted.
wl	Executes the `add` command for each port present in the port file every 20 seconds.
port	If the port given as parameter is not present in the port file, it executes the `add` command with the port as parameter and adds it to the port file.
ost	Cleans the port file (`del` command), executes the backdoor and the `wl` command, waits for the execution of the backdoor to finish, removes the current directory if the port file does not exist, kills the script instance that was executing the `wl` command, sleeps five seconds and terminates the script execution if the backdoor does not exist anymore.
st	Executes the `ost` command. It seems to be the entry point of the script as this is the command called in `/etc/rc.local`.

Table 1: Commands implemented by the launcher script

4 Backdoor

The backdoor takes one command line argument that corresponds to a port number (see Section 4.2 for its usage).

4.1 Configuration

The backdoor configuration is read and decrypted from the `conf` file (present in the same directory as the backdoor). Depending on the version of the backdoor, the configuration is either encrypted with ChaCha20 (2019 version) or with ChaCha20-Poly1305 (2021 version). The encryption key is hardcoded in the implant.

The encrypted configuration matches the structure detailed in Figure 1. Obviously, the `tag` field is only present if the encryption algorithm is ChaCha20-Poly1305.

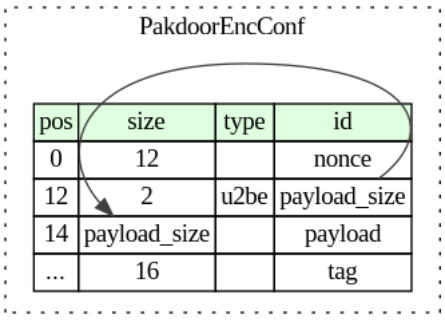

Figure 1: Structure of the encrypted configuration

The payload size is present in the encrypted configuration but it is also hardcoded in the backdoor. In all the analysed samples, the hardcoded size is equal to 372 bytes and not checked against the size present in the configuration file.

The decrypted configuration contains mainly pairs of IP address and port to which the backdoor can establish a TLS socket. One implant cannot have more than ten peers at a given time.

4.2 Communication

Communication between nodes (compromised routers) is done through TLS sockets where each side needs to authenticate with a valid certificate signed by a certificate authority created by APT31. Each node opens one TLS listening socket to receive sent or forwarded tasks from other nodes.

4.2.1 Initialization

At the beginning of its execution, the implant parses the launcher script to retrieve the certificates and the private key. More precisely, there is a certificate for the certificate authority (CA) and another one (from here on referred to as **node certificate**) used either to authenticate the node as a server or as a client. The node certificate is signed by the CA certificate and the private key is the one of the node certificate. From what was observed, there is one certificate authority by pool of compromised routers to ensure the segmentation between pools. The latter are identified by the common name of the issuer field in the certificates.

These certificates are used by the backdoor to create a listening TLS socket on the port given as command line argument.

4.2.2 Peers management

As the implant can communicate with several peers concurrently, it implements its own hash table to manage active peers. Note that the same hash table is also used for traffic relay management (see Section 4.3.2 for more details). The key of a slot in the table is generated from the public key of the peer and the hash algorithm used to compute the index is a slight variant of the djb2[2] algorithm.

As collisions can occur (a same slot in the table for two different public keys), each slot is a linked list where a node of the list is a fixed-size structure of 482 bytes that contains several details on a given peer, including its IP address, listening port, public key and the last POSIX time at which it was active.

4.2.3 Asynchronous event handlers

The backdoor makes heavy use of the `libev` library to handle I/O on the communication sockets. The man page of libev[3] is very complete, however the structures `ev_io` and `ev_timer` deserve special attention as they are regularly used in the code.

The `ev_io` structure is used to register a callback if one or several events occur on a given file descriptor. For example, one can create a watcher on a network socket for the `EV_READ` event, thus if the socket becomes readable, the callback defined in the watcher will be called. As its name suggests, the `ev_timer` structure works in the same way, but instead of waiting for a particular event, the callback is called after the amount of time defined in the watcher has elapsed. Also, a timer watcher can be repeated for a specific number of times once the callback has been executed.

[2] http://www.cse.yorku.ca/ oz/hash.html
[3] https://linux.die.net/man/3/ev

Figure 2 shows how the backdoor uses `ev_io` watchers to register callbacks that read from (`f_read_from_tls_socket_callback()`) or write to (`f_write_to_tls_socket_callback()`) the TLS listening socket.

```
// Register write watcher on socket.
p_write_watcher->fd = socket;
p_write_watcher->watcher_list.watcher.pending = 0;
p_write_watcher->watcher_list.watcher.active = 0;
p_write_watcher->watcher_list.watcher.ev_cb_declare = (int)f_write_to_tls_socket_callback;
p_write_watcher->watcher_list.watcher.priority = 0;
p_write_watcher->events = EV__IOFDSET|EV_WRITE;
ev_io_start(p_ev_loop, p_write_watcher);

// Register read watcher on socket.
p_read_watcher = peer_ctx->p_read_watcher;
p_read_watcher->watcher_list.watcher.ev_cb_declare = (int)f_read_from_tls_socket_callback;
p_read_watcher->fd = socket;
p_read_watcher->events = EV__IOFDSET|EV_READ;
p_read_watcher->watcher_list.watcher.pending = 0;
p_read_watcher->watcher_list.watcher.active = 0;
p_read_watcher->watcher_list.watcher.priority = 0;
ev_io_start(p_ev_loop, p_read_watcher);
```

Figure 2: IDA decompiler view of watchers used to handle I/O on listening socket

4.2.4 Messages structure

Messages exchanged over TLS sockets respect a specific structure described in Figure 3.

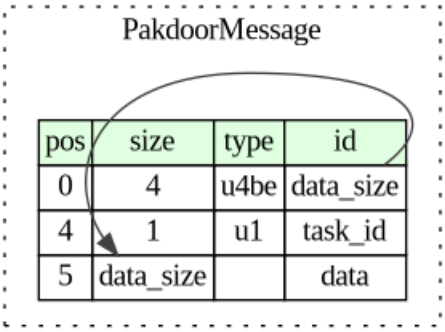

Figure 3: Structure of messages exchanged between peers

Communication between nodes always starts with a 5-bytes long message that contains the identifier of the task to be executed by the peer as well as the size of additional data if required. The latter is sent to the peer immediately after the message.

4.3 Tasks

The backdoor can handle different tasks to discover and manage its peers, relay operators traffic, and for administration purposes.

4.3.1 Peer discovery

Before being able to handle any other task, the backdoor initiates a communication with the first valid peer in the configuration file. This discovery process has two purposes:

- being acknowledged as a valid peer by the peer in the configuration;
- retrieving the active peers of the latter.

As the backdoor only accepts tasks from peers present in the active peers list, this process is critical for the backdoor to work properly.

Figure 4 illustrates a successful discovery process between two nodes.

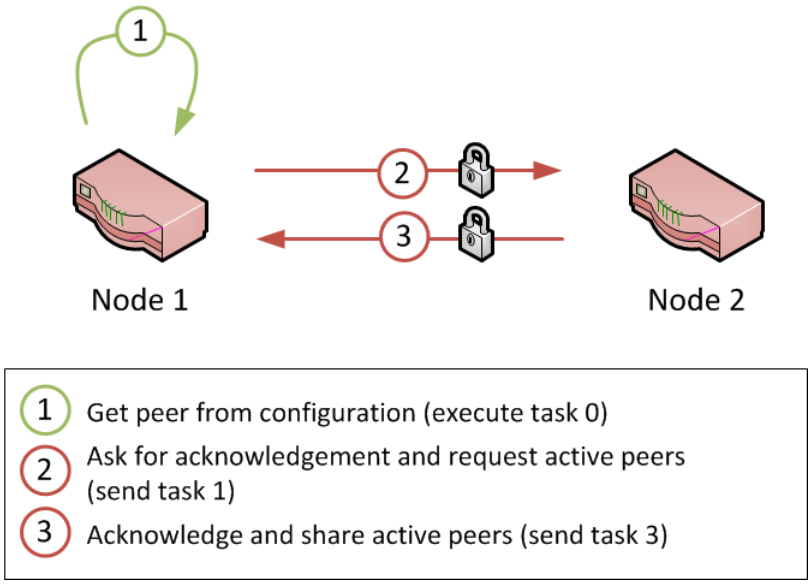

Figure 4: Successful peer discovery process

When the node 2 (see Figure 4) executes the task #1, it does several checks before inserting the node 1 to its active peers list.

1. The public key checksum in the node 1 information (sent along with task #1) has to match the actual checksum of the public key. If they do not match, the node 1 is simply ignored.
2. The last active time in the node 1 informations must not exceed three hours. Otherwise, the node 2 sends the task #2 to the node 1 to ensure it is still alive. The latter would then reply with the task #4 to acknowledge and terminate the discovery process. However,

in that specific case, the node 2 acknowledges the node 1 as a valid peer but it does not share its active peers.

3. The active peers list should not contain more than ten peers (see Figure 5). If it is not true, then the node 2 (see Figure 5) sends the task #2 to the first peer in its list (node 3) to check if it is still alive. If the latter replies with the task #4, then the node 1 is ignored, otherwise it is inserted in the list. In both cases, the node 2 sends its active peers to the node 1.

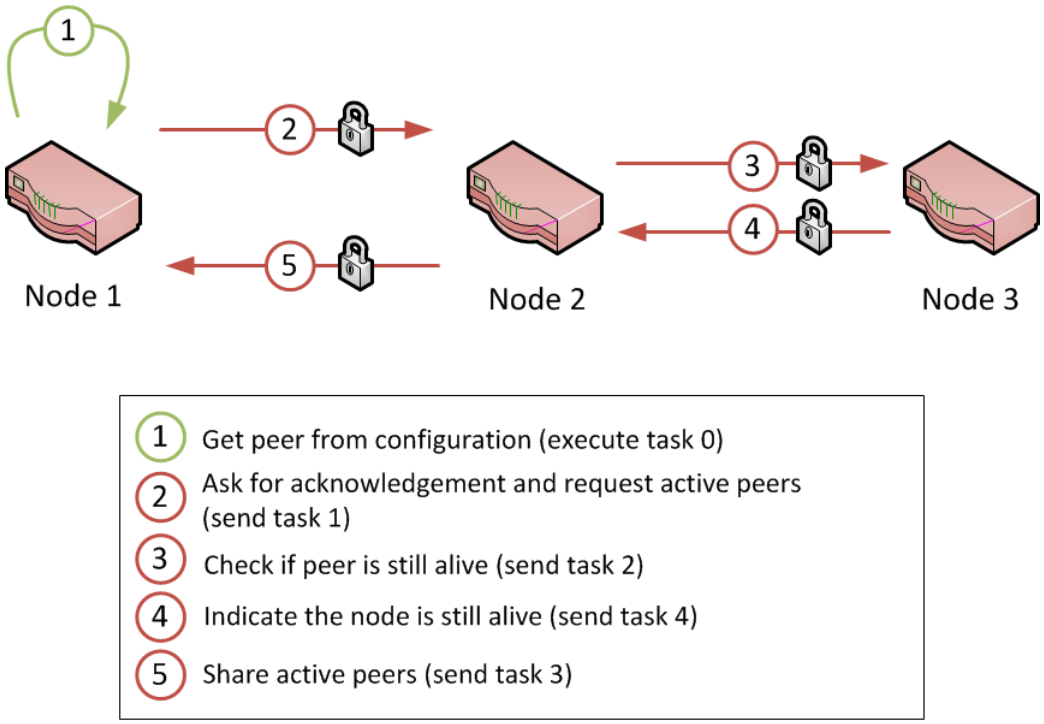

Figure 5: Peer discovery process where a node has already too many peers

For cases 2 and 3, the backdoor reads the next peer in the configuration and repeats the process until it has ten active peers.

4.3.2 Traffic relay

The main feature of the backdoor is to provide the capability to relay traffic from a node to another. The operators can define any traffic chain as long as any two consecutive nodes in the chain have acknowledged each other.

A traffic relay chain is composed of two types of nodes:

- one or several middle relays that forward traffic transparently;
- a single exit relay that encrypts or decrypts traffic with a session key, and forwards it.

The number of nodes in the relay chain may vary from one node (a single exit relay) to four nodes (three middle relays and one exit relay).

The traffic relay session lifecycle can be split into three distinct phases. Each one respectively corresponds to the colors red, blue and black on Figure 6.

1. To create a traffic relay session, the operators send the task #5 to the first middle node. Then, the latter sends the task #5 to the next node and so on until the exit node. Additional data for the task #5 contains the configurations of the relays (the corresponding structures are detailed in the next sections). Each one of them is encrypted with the public key of the recipient node. The task #5 consists in creating UDP sockets (on ports arbitrarily chosen by the operating system) to manage the relay session. The backdoor replies with task #6 to indicate the sockets have been created.

2. The operators initiate the relay session by sending a specific message on the listening UDP socket of the first middle relay. The latter forwards the message to the next relay and so on. The `port` command of the launcher script is used to open the port on the local firewall. Once the initialization process is over, the exit relay replies with another relay message to indicate it is ready to relay traffic.

3. The operators send encrypted traffic to the first middle relay. Depending on the version of the backdoor, traffic is either encrypted with the ChaCha20 algorithm (2019 version) or with the ChaCha20-Poly1305 variant (2021 version). The session key is only present in the configuration of the exit relay.

Before starting a relay session, the backdoor forks itself so the parent process can continue to process other tasks. The PID of the child process is stored into the hash table used for the active peers list. The corresponding slot key is the listening port of the UDP socket used to manage the relay session. Because of this parallelization, a given node of the infrastructure can be used for several relay sessions at the same time, both as a middle and exit relay for example.

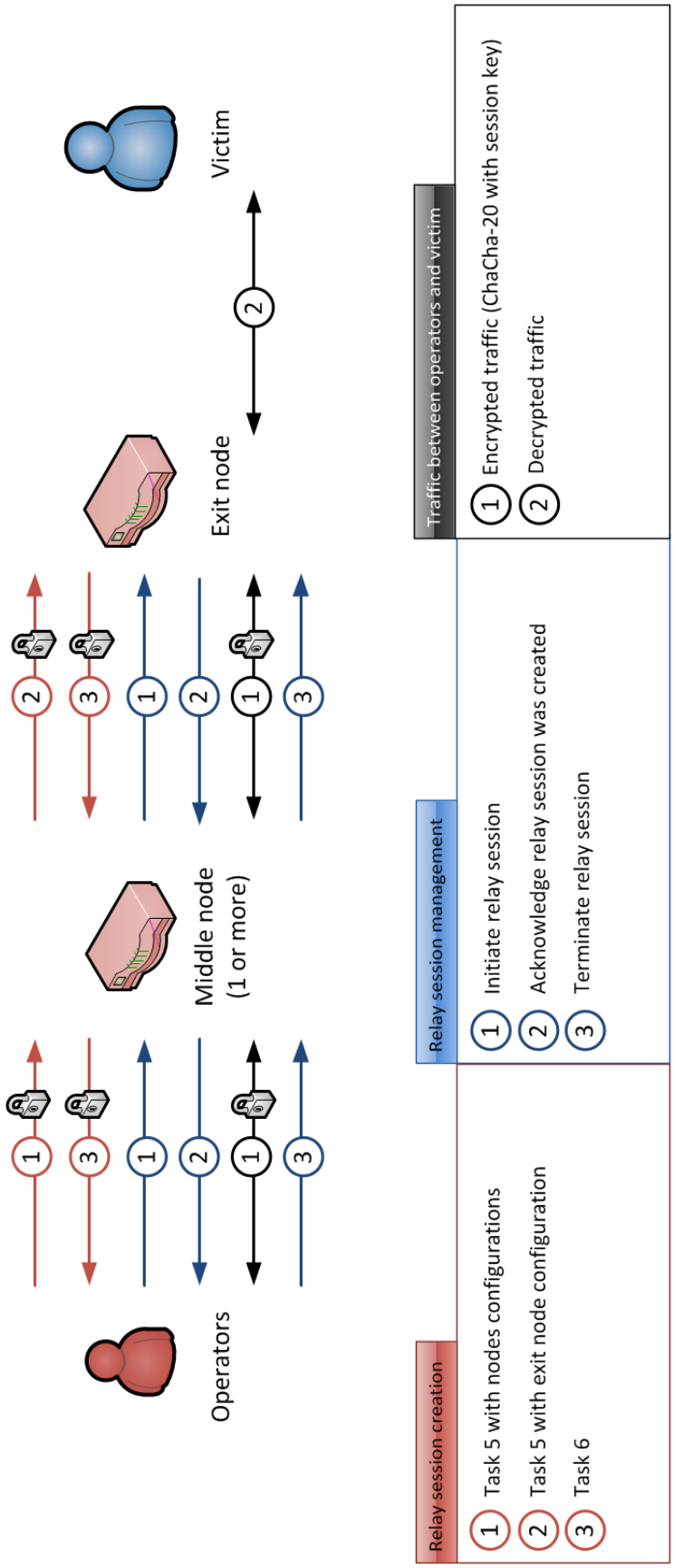

Figure 6: Overview of the lifetime of a traffic relay session

4.3.3 Administration

The operators can manage the different nodes of the infrastructure using tasks #7 and #8. The former implements standard backdoor commands such as:

- read or write a file;
- execute a shell command;
- get the current configuration.

To ensure the integrity of the payload contents, the CRC-64 checksum of the latter is checked before processing the command. Moreover, the checksum is signed with the private key of the certificate authority (see Figure 7), thus, even if one control a node of the infrastructure, it cannot send commands to another node without knowing the operators private key.

```
// Verify checksum signature.
if ( mbedtls_pk_verify(
       &p_task->p_main->p_cert_chain->p_ca_cert->pk,
       0,
       p_cmd_data->payload_checksum,
       15,
       p_cmd_data->checksum_sig,
       256) )
{
  return -1;
}
f_inverse_data_endianess(p_cmd_data);

// Verify checksum.
checksum = f_compute_crc64_checksum(p_cmd_data->payload, p_cmd_data->payload_size);
if ( memcmp(p_cmd_data->payload_checksum, &checksum, 8) )
  return -1;
```

Figure 7: IDA decompiler view of the verification of the payload checksum signature

5 Conclusion

Here are some key points about the analysis detailed in this report.

- This implant is used since at least 2019 (according to the submission date of one of the samples to VirusTotal) by APT31.
- Its purpose is to build a peer-to-peer C&C infrastructure where each peer has to authenticate with a TLS certificate to communicate with another peer.
- Most of the infrastructure lifecycle is automated using scheduled tasks, whether it is for peer discovery or to ensure peers are still alive (if not the backdoor automatically removes itself).
- The operators use the resulting infrastructure to build anonymous proxy chains that support TCP, UDP and raw communications with the victim. Besides, process forking allows the backdoor to run several proxy chains in parallel.
- The implementation of standard backdoor commands (read or write files and execute shell command lines) provides an easy way for the operators to manage compromised routers. Besides, the checksum of the command payload is signed with the operators private key to ensure the authenticity of administration commands.

A Hashes

A.1 Backdoor

MD5	77c73b8b1846652307862dd66ec09ebf
SHA1	cadf644815f758b78774c1285245e9be13b098fe
SHA256	1d60edb577641ce47dc2a8299f8b7f878e37120b192655aaf80d1cde5ee482d2
Size in bytes	509952
Comment	Available on VirusTotal

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION

ANSSI - 51, boulevard de la Tour-Maubourg - 75700 PARIS 07 SP

www.ssi.gouv.fr