NORTHWAVE
CYBER SECURITY

LITTLELAMB.WOOLTEA: Stealthy Network Edge Device Backdoor

Date: December 2024

Authors: Michaël Schrijver, Alex Oudenaarden

Introduction

During a forensic investigation, we observed an active attack on a Palo Alto network firewall. These firewalls, classified as “network edge devices,” are high-value targets for advanced threat actors, as detailed in our Global Threat Landscape report.

Upon closer examination of the compromised device, we identified a novel, stealthy, and advanced backdoor, which we believe is associated with the LITTLELAMB.WOOLTEA malware. The delivery mechanism of this malware has been described by Google Mandiant[1] and Fortinet[2]. But, with no public technical details of the backdoor itself, we had to conduct our own research to estimate its impact and intentions. For future reference and use by other researchers, we compiled our results into this document.

For other information about the investigation itself, please refer to our related blog post.

Summary

A suspected nation state threat actor gained entry to a Palo Alto network device through **CVE-2024-9474**[3], shortly after details of the vulnerability were made public. They subsequently injected commands using curl to download a file **bwmupdate** to **/tmp**, make it executable and then execute it. The output of the injected commands was redirected to a file in **/var/appweb/htdocs/unauth**.

The **bwmupdate** file installs the backdoor, which disguises itself as the logd service, by copying the malware to **/usr/local/bin/logd**. This backdoor is then executed using execve(), which fully replaces any running legitimate logd process with the malicious one. First, to ensure persistence, the malware adds its path to the **/etc/rc.local** file and modifies packages.py, a part of the RedHat package manager used by PanOS. The modified function runs during system upgrades and ensures the backdoor remains installed across upgrades.

Next, the malware injects a small dynamic library into the running nginx process. This library hijacks the accept() function to check for incoming connections that contain a unique 48-byte pattern, known as a magic knock, used by the threat actor to identify itself to the backdoor. Once the threat actor is connected, the injected dynamic library creates a file called /tmp/clientsDownload.sock. This socket file is used to pass file descriptors belonging to the threat actor's connections back to the malicious logd process. By doing so, the malware avoids having to open its own port, instead piggybacking on an existing open port.

The logd process waits for a file descriptor to appear before starting its main backdoor loop. Once the loop begins, SSL is configured to use a local certificate for encrypting the backdoor traffic. This certificate, located at /opt/pancfg/etc/appweb/server.crt, belongs to the management interface of the device. With its SSL instance configured, the backdoor is ready to start its main functionality.

A full description of the backdoor functionality, together with all the payload formats, can be found below.

Malware sample details

Hashes and names

- 8f5956869541804fda72aaeddf7db21586669c22c1b970c5b0d381f9e45c5cf4 injector
- 83c73a27663954130a605e70bb99d9a0c2ff5d849626b5c6a563207f76e45864 libhttpd.so
- 5d5945550e6fd54afffa370cadbf92010c9011a49ecd973c3f49e610abc2bffc libhttpd.so.1
- 9671d82f141950461152b183f02c6d698d16942f7fb10ebb1a02dd90c9e882a6 logd

Functionality

The backdoor provides five core functions, consisting of over 30 associated commands. These commands are received through the hijacked accept() connection following the magic knock, as previously described. The backdoor's functionalities include:

- Reading and writing files
- Providing shell access
- Establishing a network tunnel where only a single listening port can be used but multiple outgoing connections
- Establishing network tunnels where multiple listening ports can be used and multiple outgoing connections
- Setting up a SOCKS5 proxy supporting multiple listening ports and multiple outgoing connections

Command Protocol

Each backdoor node can be administered directly from the operator's server or through a tunnel. The protocol used for sending the commands for administering the nodes is the same in either case. The available command set differs between nodes connected directly to the operator and ones that act as hops (only connected to other nodes). Operator connections are differentiated from inter-node links by an identifier field provided in the handshake message. For operator connections this field is set to zero, for node connections it is set to the initiating node's id.

This is an example of the protocol (header) that is used for interfacing with a backdoor node:

Basic Frame

```
06 00 de ad ca fe 00 00 00 08 00 00 00 01 00 00 ..............
ca fe                                           ..
```

Command	Outcome	Source identifier	Payload length	Route length	Destination identifier 1

The outcome field in the header is set to zero for success or per-message constants for specific errors. For messages not signaling a result this is always set to zero.

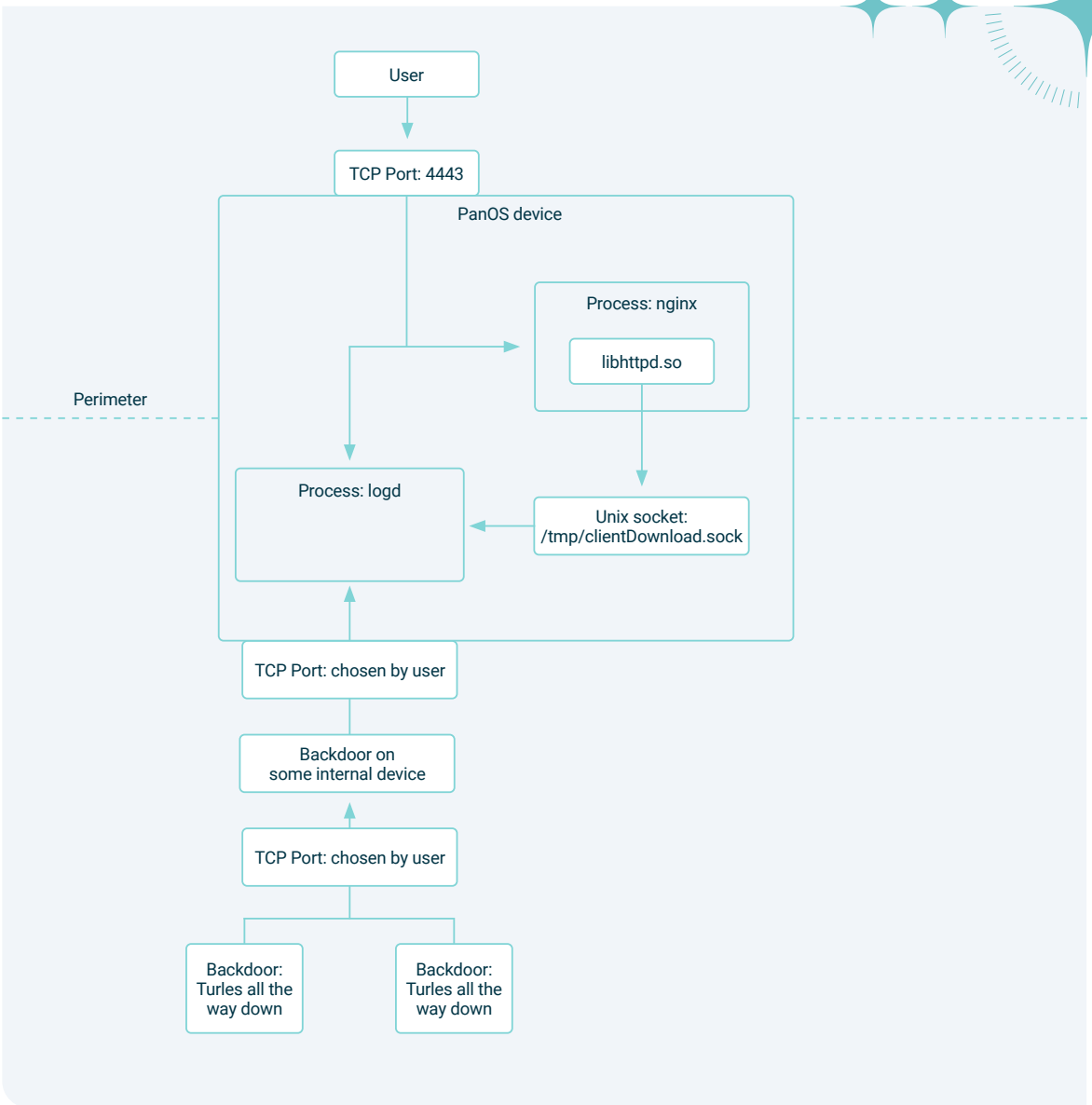
The start of the payload portion of the basic frame can be used to carry routing information. The first part of this routing information is a big-endian 32-bit integer specifying the number of hops in the route, followed by a big-endian 32-bit integer node id for each hop specifying the node it should be routed to.
If a node receives a frame with only a single hop it checks the specified node id matches its own node id and processes the message if so. Non-matches are silently discarded.

If the frame contains multiple hops the first one is stripped, and the message is forwarded to that node if the specified node is known by the current node. Non-matches here are also silently discarded.

In the sections below only the actual message payload is shown. The message header and routing block are omitted for brevity. Not all messages carry a payload. The payload table is omitted for these messages.

For certain types of operations 32-bit integer identifiers are used. These are generated using a PRNG seeded by **time()**. Examples of these are connection identifiers used in the various tunnel implementations.

The backdoor can maintain links to other nodes. In other to know on which connection to send messages destined for other (not directly connected) nodes each node maintains a view of the network. For new and lost connections messages are sent to communicate this to other nodes. The user node is somewhat special in this respect, each node has 1 connection which it uses to send message upstream to the user. If a new user connection is made to a node, it will completely reset the state of the node. The new user will also receive a list of known nodes if more than one node is active on the network. The image below illustrates the concept. Where multiple interior nodes use this functionality to connect back to the user.



Connection management

The messages in this section are mostly used to implement the inter-node connectivity behavior described in the previous section.

These messages are used to configure inter-node connections. While an inter-node connection could be set up directly to the backdoored port, it is also possible to open new listeners. This would allow the user to open a listening on the inside interface of a perimeter firewall and connect other nodes from inside the perimeter making connections look less suspicious.

Message 1: Handshake

This is the first message sent on a connection. The payload length must be zero. The identifier is set to zero for a user connection or to the initiating node's id for an inter-node connection.

Message 2: Node added

Submitted upstream by a node if it gets a new node connection.

Node Added

```
c2 95 f7 70 de ad ca f1 02 02 00 0a 00 00 e4 fe ...p............
```

Identifier	Added node identifier	Node address	Node port

Message 3: Node removed

Submitted upstream by a node if it loses a node connection.

Node Removed

```
cf 9e 6e 72 de ad ca f1 02 02 00 0a 00 00 e5 5e ..nr...........^
```

Identifier	Added node identifier	Node address	Node port

Message 4: Node list

Submitted upstream if a user connection is made.

Node List

```
00 a5 c6 f6 00 00 00 01 de ad ca f1 02 02 00 0a ................
00 00 d9 f7                                     ....
```

Node identifier	Number of entries	Entry 1 identifier	Entry 1 address	Entry 1 port

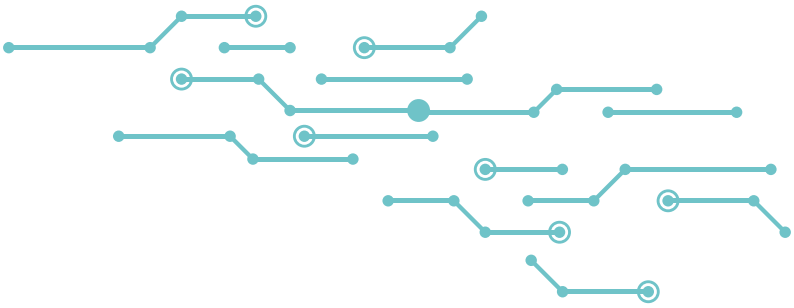
Message 37: Node connect

Initiate a node connection to the specified address and port.

Node Connect

```
01 00 00 7f 10 01 00 00 00 00 00 00 00 00       ..............
```

Address	Port	Unknown/unused



Message 38: Node listen

Open a TCP port accepting connections from other nodes (or users). If a listening port was already configured it will be shut down.

Node Listen

```
01 00 00 7f 10 01                                  ......
```

Address | Port

Message 39: Node listen response

Sent in response to a Node listen message.

Outcome	Meaning
8	Failed to open port.
9	Listener was already active.

Message 40: Node close listener

Close the listener.

Message 6: Uname Request

Request the hostname and kernel revision from the node.

Message 7: Uname Response

Response to Uname Request containing the hostname and kernel revision. As our test setup ran on Debian Linux instead of PanOS it shows a Debian kernel revision. The kernel revision wouldn't show Debian for an actual PanOS device.

Uname Response (7)

```
02 06 00 4a 00 64 65 62 69 61 6e 36 2e 31 2e 30 ...J.debian6.1.0
2d 32 38 2d 61 6d 64 36 34 20 23 31 20 53 4d 50 -28-amd64.#1.SMP
20 50 52 45 45 4d 50 54 5f 44 59 4e 41 4d 49 43 .PREEMPT_DYNAMIC
20 44 65 62 69 61 6e 20 36 2e 31 2e 31 31 39 2d .Debian.6.1.119-
31 20 28 32 30 32 34 2d 31 31 2d 32 32 29 20 78 1.(2024-11-22).x
38 36 5f 36 34                                   86_64
```

Field count | Hostname length | Version length | Hostname | Version

Message 9: Echo Request

Solicit an echo response from the other. Also keeps the connections alive.

Message 10: Echo Response

Response to an Echo Request.

Message 32: Close connection

Close the current connection from the node.

Shell

The backdoor supports running commands in a shell. The shell started is “sh”. Output from stdout or stderr is forwarded to the user with message 13, the payload of any message 13 received will be written to the shell stdin.

Message 11: Open Shell

Start a shell process.

Message 12: Open Shell Response

Signal whether a shell was successfully opened.

Outcome	Meaning
1	Shell opened succesfully
2	Failed to open shell

Message 13: Shell Data

Carries data to or from the shell process. Data sent from the user to the shell process gets “\n” appended. The payload “exitshell” is not forwarded, but closes the shell.

Shell Data

```
65 78 69 74 73 68 65 6c 6c                      exitshell
```

Data

File

The backdoor supports interacting with files on the system where the backdoor is installed. Write-append and read modes are supported. Only a single open file can be open at any time in the backdoor. The backdoor keeps track of the file size and the file pointer using 64-bit integers. These two values are also sent in most file related messages.

Message 14: Open for writing

Open a file for (repeated) writing

Open File For Write Append

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0c ................
2f 74 6d 70 2f 74 65 73 74                      /tmp/test
```

File pointer File size File name

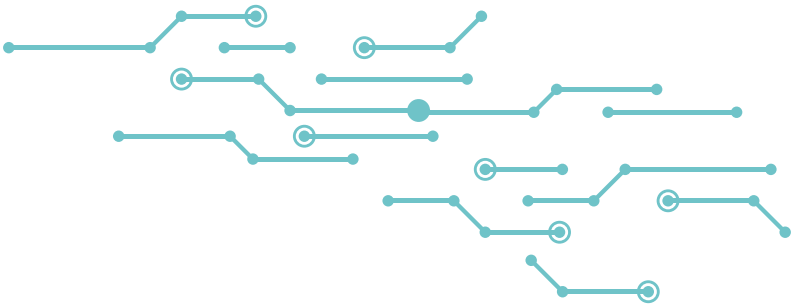
Message 15: Open and read

Open file and read its contents back to the c2 server

Open And Read File Contents

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
2f 65 74 63 2f 6f 73 2d 72 65 6c 65 61 73 65    /etc/os-release
```

File pointer File size File name



Message 16: Open response

Sent in response to message 14 or 15.

Outcome	Meaning
3	Error opening file
4	File pointer beyond end of file

Message 17: File data

Write to file opened using message 14 or data read from file opened using message 15.

File Data

```
68 65 6c 6c 6f 20 77 6f 72 6c 64 21            hello.world!
```

Data

Message 18: Close file

Close open file.

Tunnel 1

Set up a node on the current device to act as a single-hop tunnel between multiple target IPs and the c2 server. In our sample this functionality appears intentionally broken. Two functions called in the tunnel 1 configure message path, always return zero. This in turn makes the tunnel1 configure handler always fail. We believe this could be a compile-time option.

Message 21: Configure tunnel request

Configure and start the tunnel with 1 listener that forwards traffic to the c2. The configuration takes an internal port to bind to and a bool indicating a UDP or a TCP socket set-up.

Tunnel1 Configure Request

```
00 00 ca fe 00 00 00 00 00 01 01 00 00 7f 10 00 ................
```

Tunnel ID | Connection ID | Protocol | Listener | Address | Port

Message 22: Tunnel1 Configure response

Sent in response to a Tunnel1 configure request.

Outcome	Meaning
7	Failure to configure

Message 23: Tunnel1 stop

Stop all forwarders and then stop the tunnel.

Message 24: Tunnel1 connect

Add one reverse connect listener that forwards traffic to the c2. This command uses the IP and port from the configuration set in command 21 to connect to a remote system. By symmetry to the other tunnel implementations, we expected message 25 to be the response, but were unable to find it.

Message 26: Tunnel1 Close single connection

Terminates a single connection by connection ID.

Tunnel1 Close Connection

```
00 00 ca fe 00 00 de ad                          ........
```

Tunnel ID	Connection ID

Message 27: Tunnel1 send data

Sends data to a connection. The connection is identified by the unique connection ID.

Tunnel1 Data

```
00 00 ca fe 00 00 de ad 44 41 54 41              ........DATA
```

Tunnel ID	Connection ID	Data

Tunnel 2

Tunnel 2 is remarkably like tunnel 1. The difference is that it supports multiple instances simultaneously which allow the user to have multiple listening ports open at the same time. Tunnel2 seems to be a slightly more developed version of tunnel1 while keeping all the features of tunnel1. Its commands are also very similar. However, there are still some subtle bugs which prevent it from being used fully, it's not clear whether this is an intentional build configuration or just bugs.

Message 28: Configure tunnel

Set up a tunnel instance. If a tunnel is opened in listening mode it is also actually started by this message. Otherwise, the Tunnel2 connect message needs to be sent to initiate a new connection.

Tunnel2 Configure Request

```
00 00 ca fe 00 00 00 00 00 01 01 00 00 7f 10 00  ................
```

Tunnel ID	Connection ID	Protocol	Listener	Address	Port

Message 30: Close tunnel

Close the tunnel instance specified by instance and all associated connections.

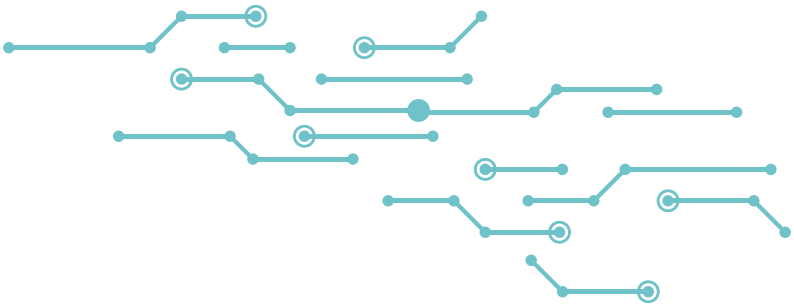
Message 32: Tunnel2 connect

Connect tunnel specified by instance and connection id. Message does nothing for listening tunnel instances.

Tunnel2 Connect

```
00 00 ca fe 00 00 de ad                          ........
```

Tunnel ID	Connection ID



Message 35: Close single connection

Close a single connection by specifying tunnel instance id and connection id.

Tunnel2 Close Connection

```
00 00 ca fe 00 00 de ad                         ........
```

Tunnel ID	Connection ID

Message 36: Send data over tunnel

This message once again is used bi-directionally. The tunnel instance id and connection id are passed to identify the connection.

Tunnel2 Data

```
00 00 ca fe 00 00 de ad 44 41 54 41             ........DATA
```

Tunnel ID	Connection ID	Data

SOCKS5

A basic socks5 implementation. It is not truly a SOCKS5 proxy implementation as data is backhauled over the node network, but it does support listening for SOCKS5 clients.

Listening instances expect SOCKS5 clients to connect. If authentication is configured the client needs to provide the configured username and password.

Once authenticated the client can request a connection, this request is forwarded over the C2 channel and must be accepted by the user. If accepted, further data is forwarded over the C2 channel.

Message 42: SOCKS5 configure

Configure a proxy instance. Both a listening and outgoing instances can be configured at the same time.

Socks5 Configure

```
00 00 ca fe 01 00 00 00 00 05 39 02 06 78 69 73 ..........9..xis
65 6b 72 31 74                                  ekr1t
```

Tunnel ID	Listener	Address	Port	Username length	Password length	Username	Password

Message 43: SOCKS5 configure response

Response to the configure message. It contains the allocated tunnel id which must be used in further messages.

Message 44: SOCKS5 close

Close an instance and all associated connections.

Socks5 Close

```
00 00 ca fe                                     ....
```

Tunnel ID

Message 46: SOCKS5 connect

Create an outgoing connection on a non-listening instance. Can be used bi-directionally.

Socks5 Connect (46)

```
00 00 ca fe 00 00 de ad 00 01 10 00 00 7f 00 00 ................
01                                               .
```

Tunnel ID | Connection ID | Protocol | Address type | Port | Unknown | Address

Message 47: SOCKS5 connect response

Response to a connect message. Can be used bi-directionally.

Socks5 Connect Response

```
00 00 ca fe 00 00 de ad                          ........
```

Tunnel ID | Connection ID

Message 48: SOCKS5 data

Carries data for the specified tunnel and connection id. It can be used bi-directionally.

Socks5 Data

```
00 00 ca fe 00 00 de ad 44 41 54 41              ........DATA
```

Tunnel ID | Connection ID | Data

Message 49: SOCKS5 close connection

Close a connection by specifying tunnel and connection id.

Socks5 Close Connection

```
00 00 ca fe 00 00 de ad                          ........
```

Tunnel ID | Connection ID

References

1. https://cloud.google.com/blog/topics/threat-intelligence/investigating-ivanti-exploitation-persistence/
2. https://www.fortinet.com/blog/psirt-blogs/importance-of-patching-an-analysis-of-the-exploitation-of-n-day-vulnerabilities
3. https://unit42.paloaltonetworks.com/cve-2024-0012-cve-2024-9474/

About Northwave Cyber Security

Founded in 2006, Northwave Cyber Security is the leading Dutch interdisciplinary specialist in cyber security, with offices in Utrecht, Leipzig, and Brussels. With their managed cyber security services, they enable European clients to remain in control while placed under the permanent protection of their confident cyber crew. Their integrated approach towards cyber risk mitigation delivers solid security and aims for cyber awareness and resilience.

Get in contact with us

Currently facing a security incident?
Call day and night: 00800 1744 000

Contact

E: info@northwave-cybersecurity.com
T: +31 (0) 30 303 1240
W: northwave-cybersecurity.com

NORTHWAVE
CYBER SECURITY