National Cyber Security Centre
a part of GCHQ

Malware Analysis Report

Pygmy Goat

TLP CLEAR

Network Device Backdoor.

Version 2

22nd October 2024

TLP MARKING: TLP:CLEAR

Pygmy Goat

Network Device Backdoor.

Executive summary

- Uses `LD_PRELOAD` to get loaded into `/bin/sshd` and hook its `accept` function.
- Listens on a raw socket for incoming ICMP packets to trigger a connect back, or uses the hooked `accept` function to search for a sequence of magic bytes in SSH connections.
- Functionality includes remote shell, packet capture, cron tasks, and creating a reverse SOCKS proxy server.

Introduction

Pygmy Goat is a native x86-32 ELF shared object that was discovered on Sophos XG firewall devices, providing backdoor access to the device.

The `LD_PRELOAD` environment variable is used to load the shared object into the `sshd` (SSH daemon) binary.

- Sample creates a raw ICMP socket to monitor for incoming packets, which contain an AES encrypted TCP callback IP and port for the sample to connect into for C2 functionality.
- Sample uses its `LD_PRELOAD` position to hook the socket `accept` function to peek at incoming traffic looking for a specific SSH protocol announcement, and then reusing that connection as an alternative means for C2.
- Sample uses a hardcoded embedded CA certificate masquerading as Fortinet to establish a TLS connection with the C2 and verify its peer.
- C2 commands enable the actor to establish a remote shell on the device, start a packet capture, create cron tasks, and create a reverse SOCKS proxy to send traffic to devices behind the firewall.

Malware details

Metadata

Filename	libsophos.so
Description	Malicious Shared Object loaded into /bin/sshd
Size	1,759,412 bytes
MD5	c71cd27efcdb8c44ab8c29d51f033a22
SHA-1	71f70d61af00542b2e9ad64abd2dda7e437536ff
SHA-256	6455de74ae15071fa98f18cdbc3148c967755e69df7dee747bc31d0387751162

Filename	libsophos.so
Description	Earlier variant of libsophos.so, missing the reverse proxy functionality, and using VMProtect to obfuscate the binary
Size	3,056,741 bytes
MD5	3f28196675dc8cb20cf5b5f80ea29310
SHA-1	7ace663c22b3e800fc17c1477d54b533f7002833
SHA-256	823b079c75f4e6a5905d9eea9a60c62e1f0995bfc25764d1ba0407a5bd78c962

Functionality

Persistence

Pygmy Goat expects to have been loaded into the `/bin/sshd` process using the `LD_PRELOAD` environment variable, as evident by a hooked `accept` function, and immediate unset of the `LD_PRELOAD` environment variable when the binary is loaded. This suggests that the actor achieves persistence on the victim device through setting the `LD_PRELOAD` environment on boot, for example by modifying a start-up script, with similar contents to:

```
LD_PRELOAD=”libsophos.so” /bin/sshd
```

This would ensure that the malicious `libsophos.so` file would be loaded into the next executed ssh daemon at system start, with the ability to overload existing functions in the `sshd` binary.

Backdoor

The Pygmy Goat `libsophos.so` binary has its `INIT_ARRAY` section populated with a single entry pointing at a `main_constructor` function (named by the embedded debug symbols left inside the binary), which is guaranteed to execute before the actual functionality of the `sshd` binary.

The `main_constructor` function forks so as not to block the loading of the legitimate sshd process, and then immediately unsets the `LD_PRELOAD` environment variable for itself and all future child forks; although it is worth noting the original parent sshd process would still have the `LD_PRELOAD` variable in its environment at this point.

The malware checks the uptime of the host system to ensure it has been up for over 60 seconds, sleeping if not. It then attempts to acquire an exclusive lock on a single-instance pid file at ‘`/var/run/sshd.pid`’, to ensure it is the only instance of the malware currently executing. The malware forks again to launch a `crond` daemon through a statically compiled embedded BusyBox 1.33.1 to execute later cron tasks that the actor can deploy to the device through the malware.

Finally, the malware creates an ICMP raw socket to listen for all ICMP packets received by the device, as well as a Unix socket listening for connections to ‘`/tmp/.sshd.ipc`’

As Pygmy Goat is loaded into the `sshd` process with `LD_PRELOAD`, any symbols exported by the `libsophos.so` shared object will replace the functions of any symbols imported by `/bin/sshd`. Finding the intersection of the exports and imports of each reveals a single symbol; the `accept` function, effectively hooking any TCP connections made to the `sshd` daemon.

On being called, the hooked `accept` function uses `dlsym` to find and invoke the 'real' `accept` function. The function then does a non-consuming, non-blocking peek at the first `0x17` bytes which it repeats every 100 milliseconds for three seconds until either `0x17` bytes are seen, the connection drops, or the time elapses. If `0x17` bytes are seen, they are compared against a hardcoded string of bytes:

```
SSH-2.0-OpenSSH_5.3p1\r\n[1]
```

If these bytes are seen, the malware detects a backdoor SSH connection, establishing a connection to the `/tmp/.sshd.ipc` Unix socket created in the `main_constructor` function, which it uses to forward all data to and from the backdoor SSH connection.

The hooked accept function also unsets the `LD_PRELOAD` variable immediately when it is called.

Since the `accept` function is called in the parent `sshd` process which still had the `LD_PRELOAD` variable set, this hides the technique from casual forensics of the device as the environment variable is only set in the `sshd` process until the first time it accepts a TCP connection. That said, if the actor doesn't attempt to connect to a Pygmy Goat victim after it first boots, or the actor uses the ICMP wake up method, the `LD_PRELOAD` variable will never be unset in the parent `sshd` process.

---

[1] Although this is a legitimate OpenSSH protocol version, it was released in 2009 and presumably deemed unlikely to appear naturally by the malware developers in 2024.

Commands

Once a connection has been established with its C2, Pygmy Goat has a number of commands it can execute according to a command ID byte. Each command is detailed further in C2 Tasking.

Command ID	Description
`0x01`	Responds with the current date and time
`0x02`	Responds with system details
`0x03`	Spawns a `/bin/sh` shell
`0x04`	Spawns a `/bin/csh` shell
`0x05`	Spawns `crontab` to create new scheduled tasks
`0x06`	Starts a packet capture
`0x07`	Connects back to an EarthWorm Server (see EarthWorm Reverse Socks Proxy) to create a reverse SOCKS5 proxy

Communications

Pygmy Goat has two mechanisms that the actor can use to establish a C2 connection to the backdoor on demand:

- Port knock via ICMP raw socket.
- Response to the hooked SSH `accept`.

ICMP Port Knock

On receiving an ICMP packet of any type, Pygmy Goat will attempt to decrypt the first `0x10` bytes in the ICMP Data section with a hardcoded IV and key, using AES-256-CBC with null padding:

```
IV:  43 4a fc 1c 5d 9d 77 06 67 c1 c3 0e c1 37 47 bb

Key: 59 4b 6e 77 51 6a 6d 41 54 62 41 6e 52 6f 5a 6d
     30 66 47 37 55 5a 57 62 32 59 55 78 55 51 50 77
```

Once decrypted, the first four bytes are compared to a magic byte sequence to ensure the data is in fact a C2 control packet and not a legitimate ICMP packet:

```
ef 12 68 45
```

The next four bytes are treated as a big-endian IPv4 address, followed by two bytes of a big-endian TCP port number, with the remainder of the data being ignored.

All example data in this report has been generated in a virtual environment using the Pygmy Goat sample, and as such is indicative, rather than being procured from any victim or actor data.

Encrypted ICMP packet

```
00 00 cc a2 5a 9d 00 01 d7 00 9e 6c 17 c0 82 6b 95 f0 fa 5b 5b 4e
dd 8a
```

Echo (ping) reply	ICMP code	ICMP checksum
ICMP identifier	Sequence Number	Encrypted Data

Decrypted packet data

```
ef 12 68 45 c0 a8 06 01 1e 61 00 00 00 00 00 00
```

Magic validation bytes	IPv4 Address 192.168.6.1	TCP port 7777
AES padding		

Once the C2 IPv4 address and TCP port have been extracted, Pygmy Goat establishes a TLS connection with the server, verifying the certificate presented by the server against a Root CA certificate embedded inside the `libsophos.so` binary (see TLS Root CA Certificate). This is somewhat noteworthy as it means the actor can send the ICMP packets from a different device to the C2 connect-back server.

The Root CA Certificate claims to have been issued by FortiGate, Fortinet Ltd., another network device vendor.

SSH Accept Hook

Once the hooked `accept` function has identified the SSH version magic bytes and delegated the connection over the Unix socket at '`/tmp/.sshd.ipc`', Pygmy Goat continues to perform a fake SSH handshake with hardcoded data, reading a fixed number of bytes in response, although ignoring the contents:

Fake SSH Handshake (C2 -> Malware)

0x0000	53 53 48 2d 32 2e 30 2d 4f 70 65 6e 53 53 48 5f	SSH-2.0-OpenSSH_
0x0010	35 2e 33 70 31 0d 0a	5.3p1..

SSH Version Exchange (peeked at following initial accept)

Fake SSH Handshake (Malware -> C2)

0x0000	53 53 48 2d 32 2e 30 2d 44 38 70 6a 45 0d 0a	SSH-2.0-D8pjE..

SSH Version Exchange

Fake SSH Handshake (Malware -> C2)

0x0000	00 00 05 4c 0a 14 fd 8d cf 7b 16 6d de 60 6f f4	...L..ý.Ï{.mÞ`oô
0x0010	1c 19 89 c1 93 ee 00 00 00 80 63 75 72 76 65 32	...Á.î....curve2
0x0020	35 35 31 39 2d 73 68 61 32 35 36 40 6c 69 62 73	5519-sha256@libs
0x0030	73 68 2e 6f 72 67 2c 64 69 66 66 69 65 2d 68 65	sh.org,diffie-he
0x0040	6c 6c 6d 61 6e 2d 67 72 6f 75 70 2d 65 78 63 68	llman-group-exch
0x0050	61 6e 67 65 2d 73 68 61 32 35 36 2c 64 69 66 66	ange-sha256,diff
0x0060	69 65 2d 68 65 6c 6c 6d 61 6e 2d 67 72 6f 75 70	ie-hellman-group
0x0070	2d 65 78 63 68 61 6e 67 65 2d 73 68 61 31 2c 64	-exchange-sha1,d
0x0080	69 66 66 69 65 2d 68 65 6c 6c 6d 61 6e 2d 67 72	iffie-hellman-gr
0x0090	6f 75 70 31 34 2d 73 68 61 31 00 00 00 13 73 73	oup14-sha1....ss
0x00a0	68 2d 72 73 61 2c 73 73 68 2d 65 64 32 35 35 31	h-rsa,ssh-ed2551
0x00b0	39 00 00 00 bb 63 68 61 63 68 61 32 30 2d 70 6f	9...»chacha20-po
0x00c0	6c 79 31 33 30 35 40 6f 70 65 6e 73 73 68 2e 63	ly1305@openssh.c
0x00d0	6f 6d 2c 61 65 73 31 32 38 2d 63 74 72 2c 61 65	om,aes128-ctr,ae
0x00e0	73 31 39 32 2d 63 74 72 2c 61 65 73 32 35 36 2d	s192-ctr,aes256-
0x00f0	63 74 72 2c 61 72 63 66 6f 75 72 32 35 36 2c 61	ctr,arcfour256,a
0x0100	72 63 66 6f 75 72 31 32 38 2c 61 65 73 31 32 38	rcfour128,aes128
0x0110	2d 63 62 63 2c 33 64 65 73 2d 63 62 63 2c 62 6c	-cbc,3des-cbc,bl
0x0120	6f 77 66 69 73 68 2d 63 62 63 2c 63 61 73 74 31	owfish-cbc,cast1
0x0130	32 38 2d 63 62 63 2c 61 65 73 31 39 32 2d 63 62	28-cbc,aes192-cb
0x0140	63 2c 61 65 73 32 35 36 2d 63 62 63 2c 61 72 63	c,aes256-cbc,arc
0x0150	66 6f 75 72 2c 72 69 6a 6e 64 61 65 6c 2d 63 62	four,rijndael-cb
0x0160	63 40 6c 79 73 61 74 6f 72 2e 6c 69 75 2e 73 65	c@lysator.liu.se
0x0170	00 00 00 bb 63 68 61 63 68 61 32 30 2d 70 6f 6c	...»chacha20-pol
0x0180	79 31 33 30 35 40 6f 70 65 6e 73 73 68 2e 63 6f	y1305@openssh.co
0x0190	6d 2c 61 65 73 31 32 38 2d 63 74 72 2c 61 65 73	m,aes128-ctr,aes
0x01a0	31 39 32 2d 63 74 72 2c 61 65 73 32 35 36 2d 63	192-ctr,aes256-c
0x01b0	74 72 2c 61 72 63 66 6f 75 72 32 35 36 2c 61 72	tr,arcfour256,ar

```
0x01c0  63 66 6f 75 72 31 32 38 2c 61 65 73 31 32 38 2d  cfour128,aes128-
0x01d0  63 62 63 2c 33 64 65 73 2d 63 62 63 2c 62 6c 6f  cbc,3des-cbc,blo
0x01e0  77 66 69 73 68 2d 63 62 63 2c 63 61 73 74 31 32  wfish-cbc,cast12
0x01f0  38 2d 63 62 63 2c 61 65 73 31 39 32 2d 63 62 63  8-cbc,aes192-cbc
0x0200  2c 61 65 73 32 35 36 2d 63 62 63 2c 61 72 63 66  ,aes256-cbc,arcf
0x0210  6f 75 72 2c 72 69 6a 6e 64 61 65 6c 2d 63 62 63  our,rijndael-cbc
0x0220  40 6c 79 73 61 74 6f 72 2e 6c 69 75 2e 73 65 00  @lysator.liu.se.
0x0230  00 01 68 75 6d 61 63 2d 36 34 2d 65 74 6d 40 6f  ..humac-64-etm@o
0x0240  70 65 6e 73 73 68 2e 63 6f 6d 2c 75 6d 61 63 2d  penssh.com,umac-
0x0250  31 32 38 2d 65 74 6d 40 6f 70 65 6e 73 73 68 2e  128-etm@openssh.
0x0260  63 6f 6d 2c 68 6d 61 63 2d 73 68 61 32 2d 32 35  com,hmac-sha2-25
0x0270  36 2d 65 74 6d 40 6f 70 65 6e 73 73 68 2e 63 6f  6-etm@openssh.co
0x0280  6d 2c 68 6d 61 63 2d 73 68 61 32 2d 35 31 32 2d  m,hmac-sha2-512-
0x0290  65 74 6d 40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c  etm@openssh.com,
0x02a0  68 6d 61 63 2d 73 68 61 31 2d 65 74 6d 40 6f 70  hmac-sha1-etm@op
0x02b0  65 6e 73 73 68 2e 63 6f 6d 2c 75 6d 61 63 2d 36  enssh.com,umac-6
0x02c0  34 40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 75 6d  4@openssh.com,um
0x02d0  61 63 2d 31 32 38 40 6f 70 65 6e 73 73 68 2e 63  ac-128@openssh.c
0x02e0  6f 6d 2c 68 6d 61 63 2d 73 68 61 32 2d 32 35 36  om,hmac-sha2-256
0x02f0  2c 68 6d 61 63 2d 73 68 61 32 2d 35 31 32 2c 68  ,hmac-sha2-512,h
0x0300  6d 61 63 2d 73 68 61 31 2c 68 6d 61 63 2d 6d 64  mac-sha1,hmac-md
0x0310  35 2d 65 74 6d 40 6f 70 65 6e 73 73 68 2e 63 6f  5-etm@openssh.co
0x0320  6d 2c 68 6d 61 63 2d 72 69 70 65 6d 64 31 36 30  m,hmac-ripemd160
0x0330  2d 65 74 6d 40 6f 70 65 6e 73 73 68 2e 63 6f 6d  -etm@openssh.com
0x0340  2c 68 6d 61 63 2d 6d 64 35 2d 39 36 2d 65 74 6d  ,hmac-md5-96-etm
0x0350  40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68 6d 61  @openssh.com,hma
0x0360  63 2d 6d 64 35 2c 68 6d 61 63 2d 72 69 70 65 6d  c-md5,hmac-ripem
0x0370  64 31 36 30 2c 68 6d 61 63 2d 72 69 70 65 6d 64  d160,hmac-ripemd
0x0380  31 36 30 40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c  160@openssh.com,
0x0390  68 6d 61 63 2d 6d 64 35 2d 39 36 00 00 01 68 75  hmac-md5-96...hu
0x03a0  6d 61 63 2d 36 34 2d 65 74 6d 40 6f 70 65 6e 73  mac-64-etm@opens
0x03b0  73 68 2e 63 6f 6d 2c 75 6d 61 63 2d 31 32 38 2d  sh.com,umac-128-
0x03c0  65 74 6d 40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c  etm@openssh.com,
0x03d0  68 6d 61 63 2d 73 68 61 32 2d 32 35 36 2d 65 74  hmac-sha2-256-et
0x03e0  6d 40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68 6d  m@openssh.com,hm
0x03f0  61 63 2d 73 68 61 32 2d 35 31 32 2d 65 74 6d 40  ac-sha2-512-etm@
0x0400  6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68 6d 61 63  openssh.com,hmac
0x0410  2d 73 68 61 31 2d 65 74 6d 40 6f 70 65 6e 73 73  -sha1-etm@openss
0x0420  68 2e 63 6f 6d 2c 75 6d 61 63 2d 36 34 40 6f 70  h.com,umac-64@op
0x0430  65 6e 73 73 68 2e 63 6f 6d 2c 75 6d 61 63 2d 31  enssh.com,umac-1
0x0440  32 38 40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68  28@openssh.com,h
0x0450  6d 61 63 2d 73 68 61 32 2d 32 35 36 2c 68 6d 61  mac-sha2-256,hma
0x0460  63 2d 73 68 61 32 2d 35 31 32 2c 68 6d 61 63 2d  c-sha2-512,hmac-
0x0470  73 68 61 31 2c 68 6d 61 63 2d 6d 64 35 2d 65 74  sha1,hmac-md5-et
0x0480  6d 40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68 6d  m@openssh.com,hm
0x0490  61 63 2d 72 69 70 65 6d 64 31 36 30 2d 65 74 6d  ac-ripemd160-etm
0x04a0  40 6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68 6d 61  @openssh.com,hma
0x04b0  63 2d 6d 64 35 2d 39 36 2d 65 74 6d 40 6f 70 65  c-md5-96-etm@ope
0x04c0  6e 73 73 68 2e 63 6f 6d 2c 68 6d 61 63 2d 6d 64  nssh.com,hmac-md
0x04d0  35 2c 68 6d 61 63 2d 72 69 70 65 6d 64 31 36 30  5,hmac-ripemd160
0x04e0  2c 68 6d 61 63 2d 72 69 70 65 6d 64 31 36 30 40  ,hmac-ripemd160@
0x04f0  6f 70 65 6e 73 73 68 2e 63 6f 6d 2c 68 6d 61 63  openssh.com,hmac
0x0500  2d 6d 64 35 2d 39 36 00 00 00 15 6e 6f 6e 65 2c  -md5-96....none,
```

0x0510	7a 6c 69 62 40 6f 70 65 6e 73 73 68 2e 63 6f 6d	zlib@openssh.com
0x0520	00 00 00 15 6e 6f 6e 65 2c 7a 6c 69 62 40 6f 70	....none,zlib@op
0x0530	65 6e 73 73 68 2e 63 6f 6d 00 00 00 00 00 00 00	enssh.com.......
0x0540	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

SSH Key Exchange Init

Fake SSH Handshake (C2 -> Malware)

0x0000	?? * 0x490

SSH Key Exchange Init (Contents ignored by Pygmy Goat)

Fake SSH Handshake (C2 -> Malware)

0x0000	?? * 0x30

SSH Key Exchange

Fake SSH Handshake (Malware -> C2)

0x0000	00 00 00 bc 08 1f 00 00 00 33 00 00 00 0b 73 73	...¼.....3....ss
0x0010	68 2d 65 64 32 35 35 31 39 00 00 00 20 3d 5f 84	h-ed25519... =_.
0x0020	8d 6b d8 10 08 e4 91 22 7b 94 28 65 e0 e0 7a 76	.kØ..ä."{.(eààzv
0x0030	83 3d 74 de 60 bf b7 4b 39 21 1d 99 1e 00 00 00	.=tÞ`¿·K9!......
0x0040	20 52 91 ba 65 28 66 56 d4 cd 7d 06 c0 6d 06 e8	R.ºe(fVÔÍ}.Àm.è
0x0050	88 01 8f 3e 3a 9a 1f 3f 1c f4 84 e6 57 39 75 f8	...>:..?.ô.æW9uø
0x0060	1b 00 00 00 53 00 00 00 0b 73 73 68 2d 65 64 32	....S....ssh-ed2
0x0070	35 35 31 39 00 00 00 40 29 cc f0 cc 16 c5 46 6e	5519...@)ÌðÌ.ÅFn
0x0080	52 19 82 8e 86 65 42 8c 1f 1a d4 c3 a5 b1 cb fc	R....eB...ÔÃ¥±Ëü
0x0090	c0 26 6c 31 3c 5c 90 3a 24 7d e4 d3 57 6d da 8e	À&l1<\.:$}äÓWmÚ.
0x00a0	cb f4 66 d1 cb 81 4f 63 fd 4a fa 06 e4 7e 4c a0	ËôfÑË.OcýJú.ä~L
0x00b0	95 91 bd cb 97 a4 b3 0f 00 00 00 00 00 00 00 00	..½Ë.¤³.........
0x00c0	00 00 00 0c 0a 15 00 00 00 00 00 00 00 00 00 00	....

SSH Key Exchange

Fake SSH Handshake (C2 -> Malware)

0x0000	?? * 0x10

SSH New Keys

After the fake SSH handshake is complete, Pygmy Goat continues to establish a legitimate TLS handshake over the fake SSH TCP connection, following the same path as with the ICMP connect-back.

This is somewhat odd as TLS handshakes are typically initiated by the TCP client, whereas in this case Pygmy Goat is the TCP server but establishes the TLS handshake as though it was the client; i.e, sending the `Client Hello`, followed by the TCP client sending the `Server Hello`.

C2 Tasking

Regardless of which mechanism was used to establish the TLS connection to the C2 server, subsequent data sent from the server to Pygmy Goat follows the same code path.

Data sent over the C2 TLS channel in either direction consists of a command byte, an identifier byte, a subcommand byte, and a two-byte big-endian length, followed by an optional LZO1X compressed data section. The identifier byte is unused for simple request-response commands, and the value sent by the server is simply echoed back in the response, however for long running commands it is used to specify which instance of a previous command to interact with, e.g, to stop a previously started packet capture. The first command packet received by Pygmy Goat following the TLS handshake is expected to be another handshake containing a sequence of expected magic bytes:

All example command packets are with TLS stripped and the LZO1X data decompressed, to make the data legible. For small packets of data the LZO1X algorithm increases the length after 'compression' due to a header, hence the length being greater than the length of the uncompressed bytes in some cases. Unless otherwise stated as 'LE' (Little Endian), all numeric values are transmitted as Big Endian values.

Pygmy Goat Handshake (C2 -> Malware)

0x0000	63 00 01 00 0c 2c 62 45 42 33 3f 3d 6f	c....,bEB3?=o

Command (Handshake)	Identifier	Subcommand (Request)
Compressed Length	Magic Bytes	

Pygmy Goat Handshake (Malware -> C2)

0x0000	63 00 01 00 00	c....

Command (Handshake)	Identifier	Subcommand (Response)
Compressed Length		

Following the handshake, Pygmy Goat will start processing subsequent packets as commands.

0x01 DateTime

Pygmy Goat responds with the current date and time, formatted with the `ctime` C function.

DateTime request (C2 -> Malware)

0x0000	01 00 01 00 00	.....

Command (DateTime)	Identifier	Subcommand (Request)
Compressed Length		

DateTime response (Malware -> C2)

0x0000 0x0010 0x0020	01 00 06 00 23 54 75 65 20 4d 61 72 20 31 34 20 31 32 3a 33 37 3a 33 33 20 32 30 32 34 20 28 44 41 54 45 29	....#Tue Mar 14 12:37:33 2024 (D ATE)

Command (DateTime)	Identifier	Subcommand (Response)
Compressed Length	Formatted date time	

0x02 Details

Pygmy Goat responds with the victim system details from the `uname` C function.

Details request (C2 -> Malware)

0x0000	02 00 01 00 00	.....

Command (Details)	Identifier	Subcommand (Request)
Compressed Length		

Details response (Malware -> C2)

0x0000 0x0010 0x0020 0x0030 0x0040 0x0050 0x0060 0x0070 0x0080	02 00 06 00 8d 53 79 73 6e 61 6d 65 3a 20 20 4c 69 6e 75 78 0a 4e 6f 64 65 6e 61 6d 65 3a 20 75 62 75 6e 74 75 0a 52 65 6c 65 61 73 65 3a 20 20 34 2e 31 30 2e 30 2d 32 38 2d 67 65 6e 65 72 69 63 0a 56 65 72 73 69 6f 6e 3a 20 20 23 33 32 7e 31 36 2e 30 34 2e 32 2d 55 62 75 6e 74 75 20 53 4d 50 20 54 68 75 20 4a 75 6c 20 32 30 20 31 30 3a 31 39 3a 31 33 20 55 54 43 20 32 30 31 37 0a 4d 61 63 68 69 6e 65 3a 20 20 69 36 38 36	.....Sysname: L inux.Nodename: u buntu.Release: 4.10.0-28-generi c.Version: #32~ 16.04.2-Ubuntu S MP Thu Jul 20 10 :19:13 UTC 2017. Machine: i686

Command (Details)	Identifier	Subcommand (Response)
Compressed Length	System Details	

0x03 System Shell

Pygmy Goat forks a new `/bin/sh` process using code copied from *The Linux Programming Interface*[2], passing data to/from the socket to/from the shell child process. The identifier byte is used to support opening multiple shells at once.

Shell start request (C2 -> Malware)

0x0000	03 7b 01 00 08 00 04 00 00	.{.....

Command (System Shell)	Identifier (123)	Subcommand (Start)
Compressed Length	Buffer window size (1024, LE)	

Shell start response (Malware -> C2)

0x0000	03 7b 02 00 00	.{...

Command (System Shell)	Identifier	Subcommand (Started)
Compressed Length		

Shell output (Malware -> C2)

0x0000	03 7b 03 00 06 23 20	.{...#

Command (System Shell)	Identifier	Subcommand (IO)
Compressed Length	Shell output ('# ')	

Shell input (C2 -> Malware)

0x0000	03 7b 03 00 08 70 77 64 0a	.{...pwd.

Command (System Shell)	Identifier	Subcommand (IO)
Compressed Length	Shell input ('pwd\n')	

[2] https[:]//man7[.]org/tlpi/code/online/dist/pty/pty_fork.c.html

Shell output (Malware -> C2)

0x0000 0x0010 0x0020	03 7b 03 00 21 70 77 64 0d 0a 2f 68 6f 6d 65 2f 75 73 65 72 2f 6c 69 62 73 6f 70 68 6f 73 0d 0a 23 20	.{...pwd../home/ user/libsophos.. #

Command (System Shell)	Identifier	Subcommand (IO)
Compressed Length	Shell output ('pwd\r\n/home/user/libsophos\r\n# ')	

Shell stop request (C2 -> Malware)

0x0000	03 7b 05 00 00	.{...

Command (System Shell)	Identifier	Subcommand (Stop)
Compressed Length		

Shell stop response (Malware -> C2)

0x0000	03 7b 06 00 00	.{...

Command (System Shell)	Identifier	Subcommand (Stopped)
Compressed Length		

0x04 CLI Shell

Pygmy Goat forks a new `/bin/csh` process, and otherwise operates in the same manner as the System Shell command including the command packets and control flow, with the sole exception of the Command byte being `0x04` instead of `0x03`; as such, example packets have not been included.

0x05 Crontab

Pygmy Goat forks a new execution of `crontab` using the statically compiled embedded BusyBox instance, otherwise operating similarly to the previous two commands. This provides the actor with an interactive instance of `crontab`, enabling them to create scheduled tasks on the victim device to execute when the actor isn't actively interacting with the system.

Crontab CLI start request (C2 -> Malware)

0x0000	05 7b 01 00 08 00 04 00 00 00 00 00 00 2d 6c	.{.....-l

Command (Crontab CLI)	Identifier	Subcommand (Start)
Compressed Length	Buffer window size (LE)	Unused
Additional args to crontab (e.g, -l to list tasks, -e to edit crontab file)		

Crontab CLI start response (Malware -> C2)

0x0000	05 7b 02 00 00	.{...

Command (Crontab CLI)	Identifier	Subcommand (Started)
Compressed Length		

Crontab CLI output (Malware -> C2)

0x0000 0x0010 0x0020	05 7b 03 00 21 33 30 20 2a 20 2a 20 2a 20 2a 20 65 63 68 6f 20 70 77 6e 64 20 3e 20 2f 68 6f 6d 65 2f 75 73 65 72 2f 70 77 6e 64 0d 0a	.{..!30 * * * * echo pwnd > /hom e/user/pwnd..

Command (Crontab CLI)	Identifier	Subcommand (IO)
Compressed Length	CLI output	

Crontab CLI stopped (Malware -> C2)

0x0000	05 7b 06 00 00	.{...

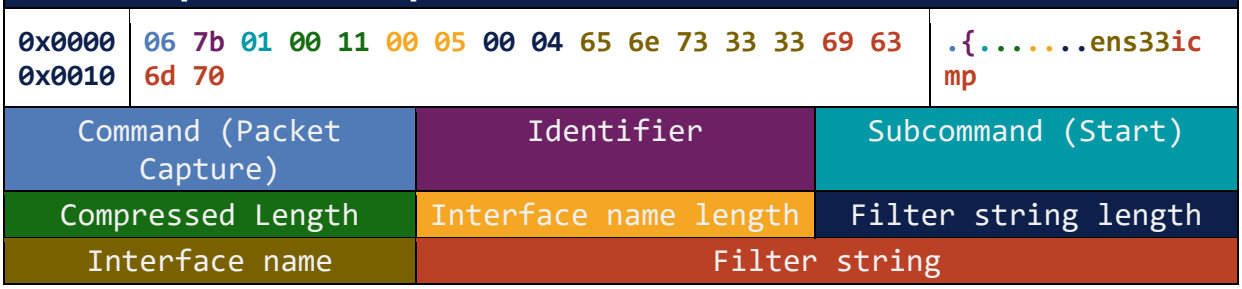
Command (Crontab CLI)	Identifier	Subcommand (Stopped)
Compressed Length		

Note, the 'Stopped' message from the malware is sent automatically when the child `crontab` process exits. In the above example, the C2 server sent the '-l' argument which causes crontab to list the current cron tasks and then exit; hence the 'IO' subcommand containing the standard output, followed by the Stopped 'subcommand'.

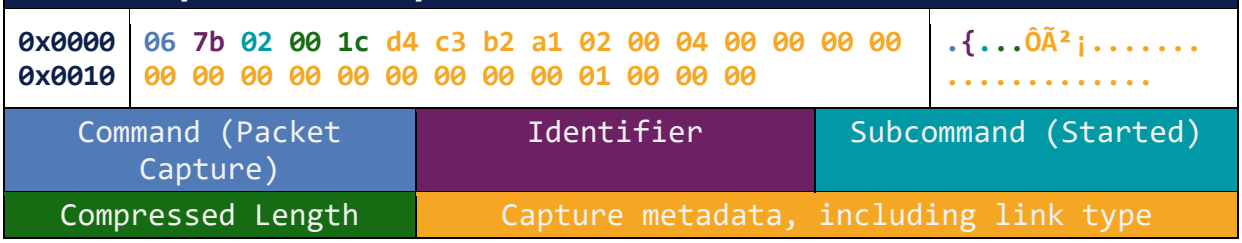
0x06 Packet Capture

Pygmy Goat uses the `libpcap` library to start capturing traffic on the specified interface according to the specified filter string until a subsequent command is sent to tell it to stop.

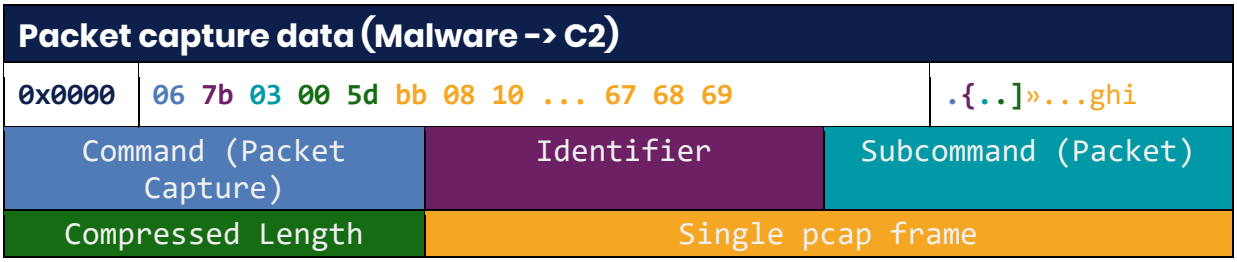
Packet capture start request (C2 -> Malware)



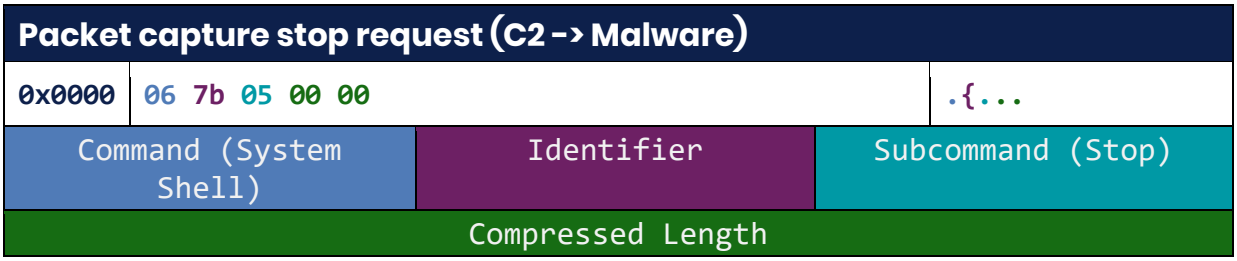
Packet capture start response (Malware -> C2)



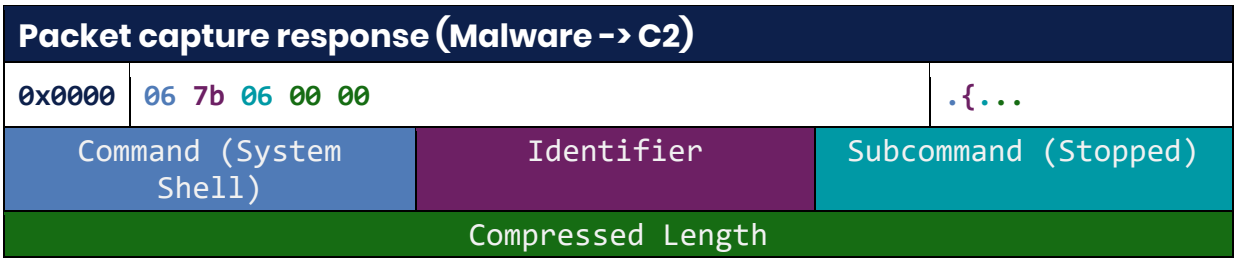
Packet capture data (Malware -> C2)



Packet capture stop request (C2 -> Malware)



Packet capture response (Malware -> C2)



0x07 EarthWorm Reverse Socks Proxy

Pygmy Goat reads a new hostname/IP address and TCP port from the compressed data, and then calls through to a copied version of the EarthWorm[3] source code, into the `create_rssocks_server` function[4], passing the address, port, and hardcoded timeout of 10 seconds.

EarthWorm is an open-source network tunnelling tool designed for offensive capabilities, in particular offering a reverse SOCKS5 proxy server to enable penetration through firewalls. While EarthWorm isn't directly malicious, it has been used by threat actors previously, as noted in reporting by Mandiant[5] and CrowdStrike[6]. The original developer has since taken down the source code from their github page, although it remains available in mirrored repositories[7].

---

[3] https[:]//rootkiter[.]com/EarthWorm/en/index.html

[4] https[:]//github[.]com/anhilo/xiaogongju/blob/master/rssocks_pro.c#L3

[5] https[:[//www.mandiant[.]com/resources/blog/pst-want-shell-proxyshell-exploiting-microsoft-exchange-servers

[6] https[:]//www.crowdstrike[.]com/blog/overwatch-insights-reviewing-a-new-intrusion-targeting-mac-systems/

[7] https[:]//github[.]com/anhilo/xiaogongju/

Reverse proxy start request (C2 -> Malware)

```
0x0000  07 7b 01 00 27 65 78 61 6d 70 6c 65 2e 63 6f 6d  .{...example.com
0x0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x00a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x00b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x00c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x00d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x00e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x00f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0100  00 00 00 00 00 22 b8                             .....",
```

Command (Proxy)	Identifier	Subcommand (Start)
Compressed Length	EarthWorm server address (256 bytes null padded)	EarthWorm server TCP port (8888)

Reverse proxy start response (Malware -> C2)

```
0x0000  07 7b 02 00 00                                   .{...
```

Command (Proxy)	Identifier	Subcommand (Started)
Compressed Length		

In a forked process, Pygmy Goat then establishes a new TCP connection to the address and port contained in the start request data.

Whilst the channel used to request that Pygmy Goat create a reverse SOCKS proxy is TLS-encrypted, the EarthWorm channel itself is not, and as such subsequent packet examples are raw TCP.

EarthWorm Client Request (Malware -> EarthWorm Server)

```
0x0000  01 01 00 00 00 00                                ......
```

Type (Handshake)	Subtype (Request)	Pool Number

EarthWorm Server Response (EarthWorm Server -> Malware)		
0x0000	01 02 00 00 00 00	.....
Type (Handshake)	Subtype (Response)	Pool Number

EarthWorm Assign Pool Number (EarthWorm Server -> Malware)		
0x0000	01 03 00 00 04 d2	.....Ò
Type (Handshake)	Subtype (Assign Pool Num)	Pool Number

Pool numbers are used so a single EarthWorm server can distinguish between multiple clients.

Pygmy Goat then establishes a new TCP connection to the EarthWorm server, this time using the Pool Number it's just been assigned, to request a remote tunnel.

EarthWorm Request Remote Tunnel (Malware -> EarthWorm Server)		
0x0000	01 04 00 00 04 d2	.....Ò
Type (Handshake)	Subtype (Tunnel Request)	Pool Number

EarthWorm Remote Tunnel Created (EarthWorm Server -> Malware)		
0x0000	01 05 00 00 04 d2	.....Ò
Type (Handshake)	Subtype (Tunnel Response)	Pool Number

At this point the EarthWorm server creates a new locally listening port, which a SOCKS5 client can connect into. All raw data sent through the locally listening port is tunnelled directly through to Pygmy Goat, which behaves as a SOCKS5 server, enabling the actor to send SOCKS traffic from their local end, through the firewall device and beyond.

Of note with EarthWorm, regardless of which authentication methods the SOCKS5 client states it supports, the EarthWorm SOCKS5 server running in Pygmy Goat will always choose 'No Authentication'. EarthWorm also only

supports the remote address being specified as IPv4, and finally, after establishing the remote connection to the IPv4 address and port specified by the SOCKS5 client, the server will respond stating that its local bind address and port is `\x41\x41\x41\x41` and `\x41\x41`:

Proxied Curl Request, Client Greeting (Curl -> EW Server -> Malware)		
0x0000	05 02 00 02	....
SOCKS Version	Num Auth	No auth & User/Pass

Proxied Curl Request, Server Choice (Malware -> EW Server -> Curl)		
0x0000	05 00	..
SOCKS Version	Auth choice (No auth)	

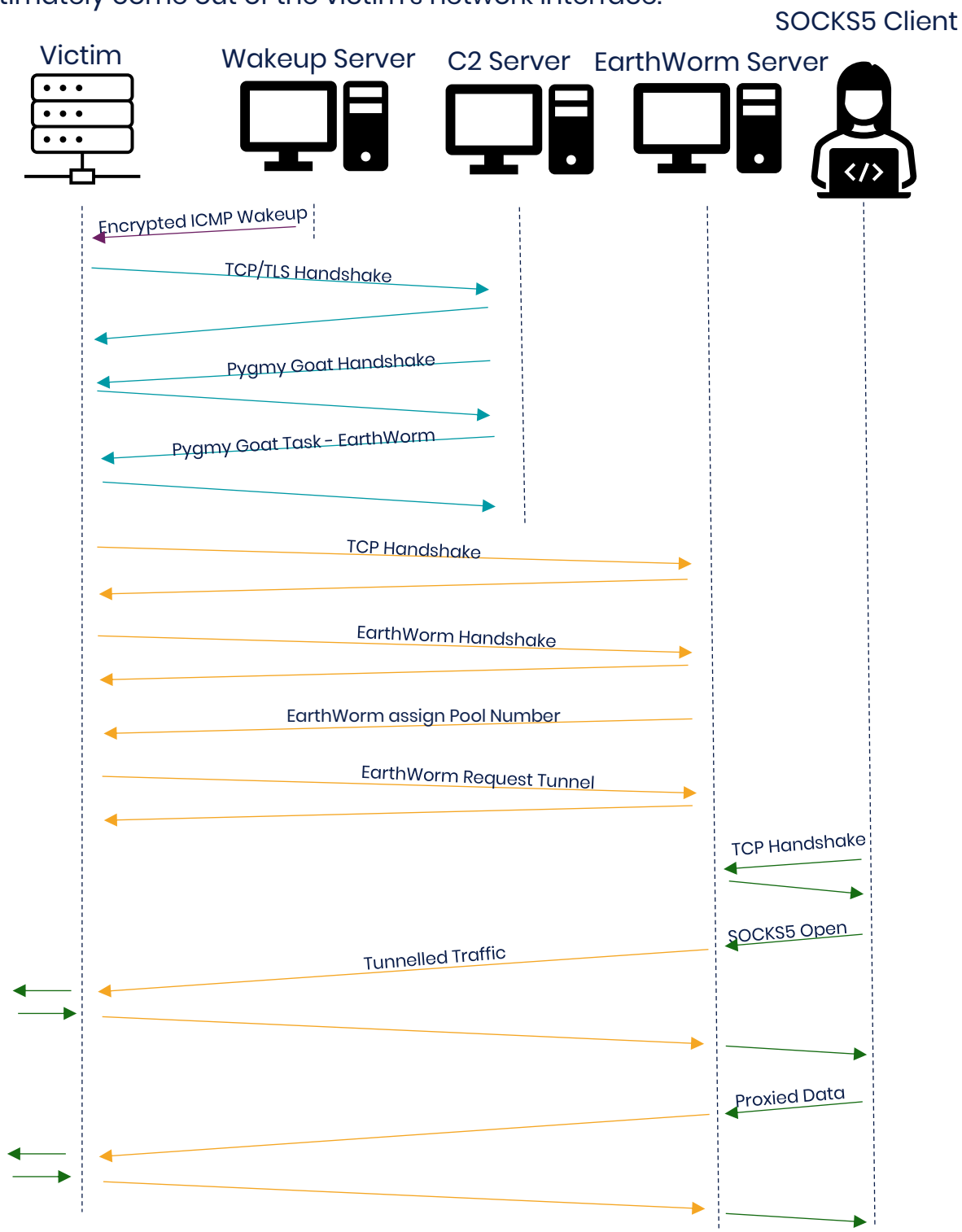
Proxied Curl Request, Client Connection Request (Curl -> EW Server -> Malware)		
0x0000	05 01 00 01 c0 a8 06 01 1a 0a	....À¨....
SOCKS Version	Establish TCP Connect	Reserved
Address Type (IPv4)	Packed IPv4 Address	TCP port

Proxied Curl Request, Server Response (Malware -> EW Server -> Curl)		
0x0000	05 00 00 01 41 41 41 41 41 41	.....AAAAAA
SOCKS Version	Request Granted	Reserved
Address Type (IPv4)	Packed IPv4 Bind Address	TCP bind port

TLP MARKING: TLP:CLEAR

Example Network Diagram

In the below example, the actor uses the ICMP wakeup method to establish a TLS connection to the IP and port inside the encrypted ICMP packet data. Pygmy Goat establishes a TCP & TLS connection to the C2 server, verifying the C2 server's TLS certificate against the embedded CA cert. The C2 server performs the Pygmy Goat handshake, followed by a task: to establish a reverse SOCKS5 EarthWorm connection with the IP and port inside the task data. After establishing the reverse tunnel, a SOCKS5 client can connect into the EarthWorm server on a newly opened port to send tunnelled data which will ultimately come out of the victim's network interface.



Conclusion

While not containing any novel techniques, Pygmy Goat is quite sophisticated in how it enables the actor to interact with it on demand, while blending in with normal network traffic. The code itself is clean, with short, well-structured functions aiding future extensibility, and errors are checked throughout, suggesting it was written by a competent developer or developers.

Although Pygmy Goat has so far only been seen on Sophos XG Firewalls, design choices suggest that it could be intended for use on an array of Linux devices rather than specifically targeted. The malware has multiple methods of comms wake-up, as well as two separate remote shells, `/bin/sh` and `/bin/csh`, which would likely be considered unnecessary effort if the malware had been developed for a specific device. Pygmy Goat does not rely on any device-specific external libraries and will run on a base Ubuntu distribution.

In particular, the embedded Root CA Certificate, which claims to have been issued by 'FortiGate, Fortinet Ltd.', as well as one of the IPC Unix sockets having the filename '`.fgmon_cli.ipc`', suggests that Pygmy Goat may have initially been intended to execute on FortiGate devices. Recent reporting from Mandiant shows attacks on FortiGate devices, with CASTLETAP having similar TTPs to Pygmy Goat, such as an encrypted ICMP packet containing C2 information being used to establish a reverse SSL connection[8].

[8] https[:]//www.mandiant[.]com/resources/blog/fortinet-malware-ecosystem

Detection

Indicators of compromise

Type	Description	Values
Path	Copied malware path	`/lib/libsophos.so`
Path	IPC Unix server socket	`/tmp/.sshd.ipc`
Path	IPC Unix client socket	`/tmp/.fgmon_cli.ipc`
Path	IPC Unix server socket from older sample	`/tmp/.goat.ipc`
Path	Single instance pid file	`/var/run/sshdd.pid`
Path	Single instance pid file from older sample	`/var/run/goat.pid`

Rules and signatures

Description	Pygmy Goat AES key built on the stack or in data
Precision	No false positives in VirusTotal retro hunt over the previous 12 months
Rule type	YARA

```
rule pygmy_goat_aes_key
{
	meta:
	  author = "NCSC"
	  description = "Pygmy Goat AES key built on the stack or in data"
	  date = "2024-10-22"
	  hash1 = "71f70d61af00542b2e9ad64abd2dda7e437536ff"

	strings:
		$dword_1 = { 59 4b 6e 77 }
		$dword_2 = { 51 6a 6d 41 }
		$dword_3 = { 54 62 41 6e }
		$dword_4 = { 52 6f 5a 6d }
		$dword_5 = { 30 66 47 37 }
		$dword_6 = { 55 5a 57 62 }
		$dword_7 = { 32 59 55 78 }
		$dword_8 = { 55 51 50 77 }

	condition:
		(uint32(0) == 0x464c457f) and all of them
}
```

TLP MARKING: TLP:CLEAR

Description	Pygmy Goat magic byte sequences used in C2 comms
Precision	No false positives in VirusTotal retro hunt over the previous 12 months
Rule type	YARA

```
rule pygmy_goat_magic_strings
{
      meta:
        author = "NCSC"
        description = "Pygmy Goat magic byte sequences used in C2 comms"
        date = "2024-10-22"
        hash1 = "71f70d61af00542b2e9ad64abd2dda7e437536ff"

      strings:
            $c2_magic_handshake = ",bEB3?=o"
            $fake_ssh_banner = "SSH-2.0-D8pjE"
            $fake_ed25519_key = { 29 cc f0 cc 16 c5 46 6e 52 19 82 8e 86
65 42 8c 1f 1a d4 c3 a5 b1 cb fc c0 26 6c 31 3c 5c 90 3a 24 7d e4 d3 57
6d da 8e cb f4 66 d1 cb 81 4f 63 fd 4a fa 06 e4 7e 4c a0 95 91 bd cb 97
a4 b3 0f }

      condition:
            (uint32(0) == 0x464c457f) and any of them
}
```

Description	EarthWorm pool num generation x86 assembly
Precision	Signature is looking for use of EarthWorm in an x86 ELF binary, which may include benign uses
Rule type	YARA

```
rule earthworm_id_generation_x86
{
	meta:
	  author = "NCSC"
	  description = "EarthWorm pool num generation x86 assembly"
	  date = "2024-10-22"
	  hash1 = "71f70d61af00542b2e9ad64abd2dda7e437536ff"

	strings:
		$chartoi = {
			8b 45 ??	// MOV		EAX,dword ptr [EBP + ??]
			c1 e0 07	// SHL		EAX,0x7
			89 c1		// MOV		ECX,EAX
			8b 55 ??	// MOV		EDX,dword ptr [EBP + ??]
			8b 45 ??	// MOV		EAX,dword ptr [EBP + ??]
			01 d0		// ADD		EAX,EDX
			0f b6 00	// MOVZX	EAX,byte ptr [EAX]
			0f be c0	// MOVSX	EAX,AL
			01 c8		// ADD		EAX,ECX
			89 45 ??	// MOV		dword ptr [EBP + ??],EAX
			83 6d ?? 01	// SUB		dword ptr [EBP + ??],0x1
		}

	condition:
		(uint32(0) == 0x464c457f) and all of them
}
```

Description	Pygmy Goat Fake SSH handshake
Precision	Unknown, presumed high due to fake SSH protocol version
Rule type	Snort v2

```
alert tcp any 22 -> any any (msg: "Pygmy Goat Fake SSH handshake";
content: "SSH-2.0-D8pjE"; offset: 0; depth: 13; classtype:trojan-
activity;)
```

TLP MARKING: TLP:CLEAR

Description	Pygmy Goat Fake SSH ed25519 key
Precision	Unknown, presumed high due to content length
Rule type	Snort v2

```
alert tcp any 22 -> any any (msg: "Pygmy Goat Fake SSH ed25519
key"; content:
"|29ccf0cc16c5466e5219828e8665428c1f1ad4c3a5b1cbfcc0266c313c5c903a
247de4d3576dda8ecbf466d1cb814f63fd4afa06e47e4ca09591bdcb97a4b30f|"
; offset: 120; depth: 64; classtype:trojan-activity;)
```

TLP MARKING: TLP:CLEAR

MITRE ATT&CK®

This report has been compiled with respect to the MITRE ATT&CK® framework, a globally accessible knowledge base of adversary tactics and techniques based on real-world observations.

Tactic	ID	Technique	Procedure
Persistence	T1574.006	Dynamic Linker Hijacking	Pygmy Goat uses the `LD_PRELOAD` environment variable to inject into `sshd`
Execution	T1059.004	Command and Scripting Interpreter: Unix Shell	Pygmy Goat can create a `/bin/sh` or `/bin/csh` remote shell
	T1053.003	Scheduled Task/Job: Cron	Pygmy Goat can create arbitrary cron tasks using crontab
	T1559	Inter-Process Communication	Pygmy Goat uses Unix sockets for inter-process communication between parent and forked child processes
Discovery	T1040	Network Sniffing	Pygmy Goat can use libpcap to sniff network traffic according to a BPF filter and exfiltrate it to the C2
Command and Control	T1205.001	Traffic Signaling: Port Knocking	Pygmy Goat listens on an ICMP raw socket for encrypted packets containing magic bytes and the C2 address to connect back to
	T1001.003	Data Obfuscation: Protocol Impersonation	Pygmy Goat responds to an SSH connection that sends magic bytes with a fake SSH handshake
	T1573.002	Encrypted Channel: Asymmetric Cryptography	Pygmy Goat encrypts C2 communications with TLS
	T1572	Protocol Tunneling	Pygmy Goat can create a reverse SOCKS5 proxy server to tunnel traffic through it
Collection	T1560.002	Archive Collected Data: Archive via Library	Pygmy Goat compresses its C2 communications with LZO1X
Exfiltration	T1041	Exfiltration Over C2 Channel	Pygmy Goat exfiltrates its result data over its C2 channel

Appendix

TLS Root CA Certificate

```
-----BEGIN CERTIFICATE-----
MIIC3zCCAccCFB8e5bk6nwcaR66tdgFt7kh7iw19MA0GCSqGSIb3DQEBCwUAMCwx
FjAUBgNVBAoMDUZvcnRpbmV0IEx0ZC4xEjAQBgNVBAMMCUZvcnRpR2F0ZTAeFw0y
MTA4MzEwMTU0NDJaFw0zMTA4MjkwMTU0NDJaMCwxFjAUBgNVBAoMDUZvcnRpbmV0
IEx0ZC4xEjAQBgNVBAMMCUZvcnRpR2F0ZTCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBAO0UTvHfYvBeIKqKYWV5xfoJW4hsHZbMHSWefuUiYSLDliBDWV8e
4hBdi6krF8YGGRkKlHPZcfTHzJQmYwBG2mAjEWiIQm3Z0aD4wJjnF/B0VAYDTG29
Vqj+PFU5UsckGeWqTomKOFwutazXiUWicGjzTJErhVj26AgXUPiKO5VHBdHR/xqW
xJ5ed4L0OOW4c/WYQjUReeiKw2iP3bAtrglXWInJexiWzA/FzsRTbwCUexGmXhwG
65QsW5t4beDCBHMJN/uP6Q7kIqtDe3/JL8osT0UmGTEcQBx1mkc7Nb6dMgabHPoM
NWbGLlxRKxb9+H0XGq+effuFMr9CUXb60PcCAwEAATANBgkqhkiG9w0BAQsFAAOC
AQEAHiZ0DxXlTEykWJxKDcKXhv02TO4C6jhDotr1xYiXha9s+o83h/Q9SFnCL7mP
1KKB/hRA6/CwXH/P9YUR0OnbPEsUJoQp3jbcXV5m/Xen/Zss+AIwCtLVy20ctPCn
svXbPp0lEf69fgByXmL1gB+03dk4QTsy96yrfIPCIXMn7Q3A5LZ2AMBSg/+YJ4xP
Il+oGhm90WUbr4PgMS+DqTHuf+ghxxHTgbRtLdCvGLA8fu6CcM8rwGae48aE/+gU
MaavuO9VUiW8eGdouyVZvGhutVpWWYABrslchLpZEF48pEFMk9ChLU9/17Qd1zgQ
Ug/Gkjn036B8ZfA3xdCpTd7ldA==
-----END CERTIFICATE-----
```

```
X509 Certificate:
Version: 1
Serial Number: 1f1ee5b93a9f071a47aead76016dee487b8b0d7d
Signature Algorithm:
    Algorithm ObjectId: 1.2.840.113549.1.1.11 sha256RSA
    Algorithm Parameters:
    05 00
Issuer:
    CN=FortiGate
    O=Fortinet Ltd.
  Name Hash(sha1): b87a11fc647eed1aed3543237cb1540d99ead580
  Name Hash(md5): d45eadb1d50562927512b7f545a02b65

 NotBefore: 8/31/2021 1:54 AM
 NotAfter: 8/29/2031 1:54 AM

Subject:
    CN=FortiGate
    O=Fortinet Ltd.
  Name Hash(sha1): b87a11fc647eed1aed3543237cb1540d99ead580
  Name Hash(md5): d45eadb1d50562927512b7f545a02b65

Public Key Algorithm:
    Algorithm ObjectId: 1.2.840.113549.1.1.1 RSA (RSA_SIGN)
    Algorithm Parameters:
    05 00
Public Key Length: 2048 bits
Public Key: UnusedBits = 0
    0000  30 82 01 0a 02 82 01 01  00 ed 14 4e f1 df 62 f0
    0010  5e 20 aa 8a 61 65 79 c5  fa 09 5b 88 6c 1d 96 cc
    0020  1d 25 9e 7e e5 22 61 22  c3 96 20 43 59 5f 1e e2
    0030  10 5d 8b a9 2b 17 c6 06  19 19 0a 94 73 d9 71 f4
    0040  c7 cc 94 26 63 00 46 da  60 23 11 68 88 42 6d d9
    0050  d1 a0 f8 c0 98 e7 17 f0  74 54 06 03 4c 6d bd 56
    0060  a8 fe 3c 55 39 52 c7 24  19 e5 aa 4e 89 8a 38 5c
    0070  2e b5 ac d7 89 45 a2 70  68 f3 4c 91 2b 85 58 f6
    0080  e8 08 17 50 f8 8a 3b 95  47 05 d1 d1 ff 1a 96 c4
    0090  9e 5e 77 82 f4 38 e5 b8  73 f5 98 42 35 11 79 e8
    00a0  8a c3 68 8f dd b0 2d ae  09 57 58 89 c9 7b 18 96
    00b0  cc 0f c5 ce c4 53 6f 00  94 7b 11 a6 5e 1c 06 eb
    00c0  94 2c 5b 9b 78 6d e0 c2  04 73 09 37 fb 8f e9 0e
    00d0  e4 22 ab 43 7b 7f c9 2f  ca 2c 4f 45 26 19 31 1c
    00e0  40 1c 75 9a 47 3b 35 be  9d 32 06 9b 1c fa 0c 35
    00f0  66 c6 2e 5c 51 2b 16 fd  f8 7d 17 1a af 9e 7d fb
    0100  85 32 bf 42 51 76 fa d0  f7 02 03 01 00 01
Certificate Extensions: 0
Signature Algorithm:
    Algorithm ObjectId: 1.2.840.113549.1.1.11 sha256RSA
    Algorithm Parameters:
    05 00
Signature: UnusedBits=0
    0000  74 e5 de 4d a9 d0 c5 37  f0 65 7c a0 df f4 39 92
    0010  c6 0f 52 10 38 d7 1d b4  d7 7f 4f 2d a1 d0 93 4c
    0020  41 a4 3c 5e 10 59 ba 84  5c c9 ae 01 80 59 56 5a
    0030  b5 6e 68 bc 59 25 bb 68  67 78 bc 25 52 55 ef b8
    0040  af a6 31 14 e8 ff 84 c6  e3 9e 66 c0 2b cf 70 82
    0050  ee 7e 3c b0 18 af d0 2d  6d b4 81 d3 11 c7 21 e8
    0060  7f ee 31 a9 83 2f 31 e0  83 af 1b 65 d1 bd 19 1a
    0070  a8 5f 22 4f 8c 27 98 ff  83 52 c0 00 76 b6 e4 c0
    0080  0d ed 27 73 21 c2 83 7c  ab ac f7 32 3b 41 38 d9
```

```
    0090  dd b4 1f 80 f5 62 5e 72  00 7e bd fe 11 25 9d 3e
    00a0  db f5 b2 a7 f0 b4 1c 6d  cb d5 d2 0a 30 02 f8 2c
    00b0  9b fd a7 77 fd 66 5e 5d  dc 36 de 29 84 26 14 4b
    00c0  3c db e9 d0 11 85 f5 cf  7f 5c b0 f0 eb 40 14 fe
    00d0  81 a2 d4 8f b9 2f c2 59  48 3d f4 87 37 8f fa 6c
    00e0  af 85 97 88 c5 f5 da a2  43 38 ea 02 ee 4c 36 fd
    00f0  86 97 c2 0d 4a 9c 58 a4  4c 4c e5 15 0f 74 26 1e
Signature matches Public Key
Root Certificate: Subject matches Issuer
Key Id Hash(rfc-sha1): 241a37a7ac3e26d8d703a8058ffe100dd1150193
Key Id Hash(sha1): d05ec61f560ec38990760bbb71339e09ebd3a4cc
Key Id Hash(bcrypt-sha1): 1febcf83a6f6e2598a5288a0e57742d1fc6e7620
Key Id Hash(bcrypt-sha256):
efbb9150e66eff1492404ca6bfb219dd656c640814e27cfb3e757ff94fe6aa5a
Key Id Hash(md5): eae7cc16a30ed5a98916f9f381a5bcb2
Key Id Hash(sha256):
8049bd8e86a6b5f382639b0739c78c5fd55780c72d3b5c9a6084e22981f9dc51
Key Id Hash(pin-sha256): frcO5XKYZ/rwLDKF6EeMNz4MYTQrkNTwd1VPrxMDwSo=
Key Id Hash(pin-sha256-hex):
7eb70ee5729867faf02c3285e8478c373e0c61342b90d4f077554faf1303c12a
Cert Hash(md5): e1b9842e7e0b9cf722bcc7d08c768486
Cert Hash(sha1): 8d453ff52947af1842a0231d74ffbb6faacf6167
Cert Hash(sha256):
f85280bd427aa2e9d714ea3bc11febf5a436cfc04fcbbe708c2592a88b6000a3
Signature Hash:
ef0ae22901ab9ab07f3b6e1f80ee41cd21deee957e81d7a48fac2517ae5ce87e
```

Disclaimer

This report draws on information derived from NCSC and industry sources. Any NCSC findings and recommendations made have not been provided with the intention of avoiding all risks and following the recommendations will not remove all such risk. Ownership of information risks remains with the relevant system owner at all times.

This information is exempt under the Freedom of Information Act 2000 (FOIA) and may be exempt under other UK information legislation.

Refer any FOIA queries to ncscinfoleg@ncsc.gov.uk.