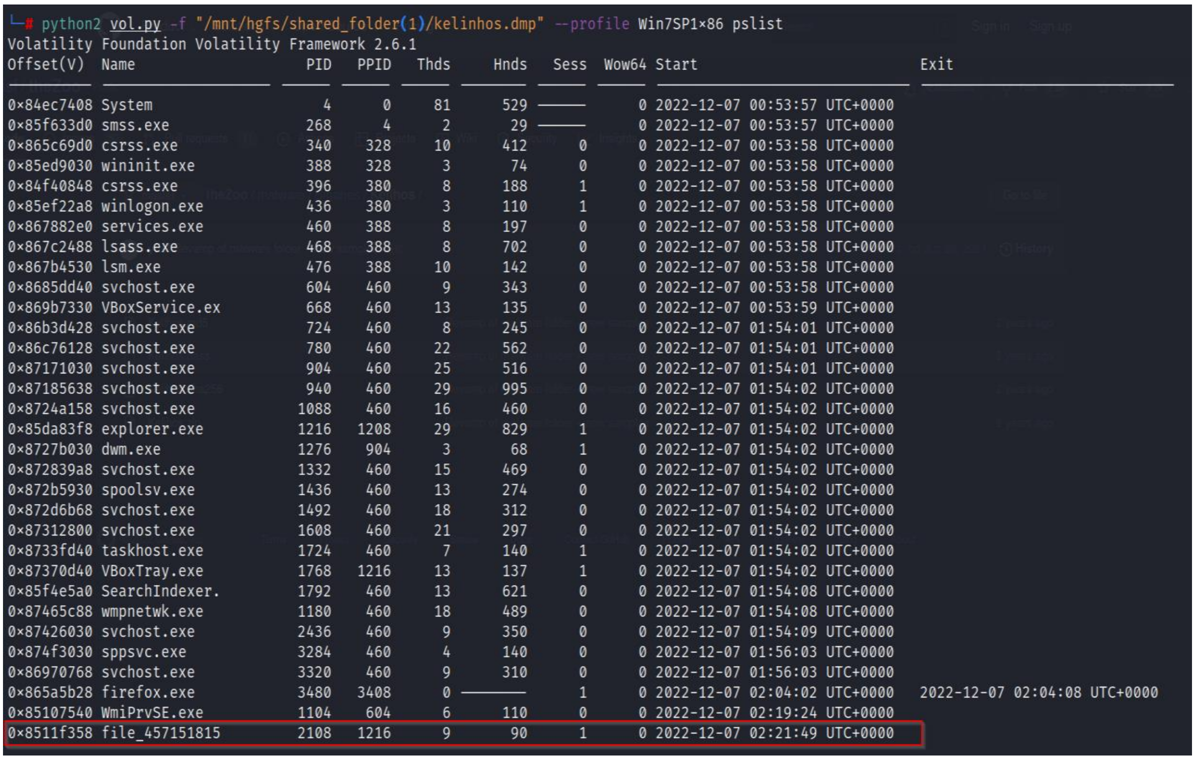
Kelihos Analysis

We chose the malware ***kelihos*** from the malware database for analysis.

On running forecast with extended logging to see the outputs of the headers being found in the binary we found the presence of remote internet connections and the socket opening in the system, which indicated a remote Command and Control (C & C) server capability.

To confirm this consideration, we had to run the malware and perform some analysis on a Windows 7 and Windows XP dump to confirm the presence of a remote connection.

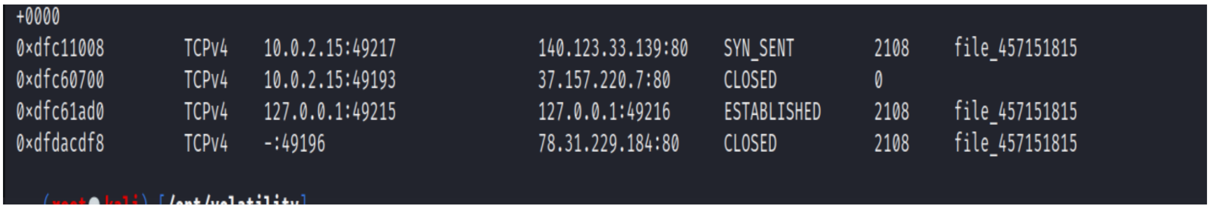
Windows 7 Analysis:



```
└─# python2 vol.py -f "/mnt/hgfs/shared_folder(1)/kelinhos.dmp" --profile Win7SP1×86 pslist
Volatility Foundation Volatility Framework 2.6.1
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------------------------------
0×84ec7408 System                    4      0     81      529 ------      0 2022-12-07 00:53:57 UTC+0000
0×85f633d0 smss.exe                268      4      2       29 ------      0 2022-12-07 00:53:57 UTC+0000
0×865c69d0 csrss.exe               340    328     10      412      0      0 2022-12-07 00:53:58 UTC+0000
0×85ed9030 wininit.exe             388    328      3       74      0      0 2022-12-07 00:53:58 UTC+0000
0×84f40848 csrss.exe               396    380      8      188      1      0 2022-12-07 00:53:58 UTC+0000
0×85ef22a8 winlogon.exe            436    380      3      110      1      0 2022-12-07 00:53:58 UTC+0000
0×867882e0 services.exe            460    388      8      197      0      0 2022-12-07 00:53:58 UTC+0000
0×867c2488 lsass.exe               468    388      8      702      0      0 2022-12-07 00:53:58 UTC+0000
0×867b4530 lsm.exe                 476    388     10      142      0      0 2022-12-07 00:53:58 UTC+0000
0×8685dd40 svchost.exe             604    460      9      343      0      0 2022-12-07 00:53:58 UTC+0000
0×869b7330 VBoxService.ex          668    460     13      135      0      0 2022-12-07 00:53:59 UTC+0000
0×86b3d428 svchost.exe             724    460      8      245      0      0 2022-12-07 01:54:01 UTC+0000
0×86c76128 svchost.exe             780    460     22      562      0      0 2022-12-07 01:54:01 UTC+0000
0×87171030 svchost.exe             904    460     25      516      0      0 2022-12-07 01:54:01 UTC+0000
0×87185638 svchost.exe             940    460     29      995      0      0 2022-12-07 01:54:02 UTC+0000
0×8724a158 svchost.exe            1088    460     16      460      0      0 2022-12-07 01:54:02 UTC+0000
0×85da83f8 explorer.exe           1216   1208     29      829      1      0 2022-12-07 01:54:02 UTC+0000
0×8727b030 dwm.exe                1276    904      3       68      1      0 2022-12-07 01:54:02 UTC+0000
0×872839a8 svchost.exe            1332    460     15      469      0      0 2022-12-07 01:54:02 UTC+0000
0×872b5930 spoolsv.exe            1436    460     13      274      0      0 2022-12-07 01:54:02 UTC+0000
0×872d6b68 svchost.exe            1492    460     18      312      0      0 2022-12-07 01:54:02 UTC+0000
0×87312800 svchost.exe            1608    460     21      297      0      0 2022-12-07 01:54:02 UTC+0000
0×8733fd40 taskhost.exe           1724    460      7      140      1      0 2022-12-07 01:54:02 UTC+0000
0×87370d40 VBoxTray.exe           1768   1216     13      137      1      0 2022-12-07 01:54:02 UTC+0000
0×85f4e5a0 SearchIndexer.         1792    460     13      621      0      0 2022-12-07 01:54:08 UTC+0000
0×87465c88 wmpnetwk.exe           1180    460     18      489      0      0 2022-12-07 01:54:08 UTC+0000
0×87426030 svchost.exe            2436    460      9      350      0      0 2022-12-07 01:54:09 UTC+0000
0×874f3030 sppsvc.exe             3284    460      4      140      0      0 2022-12-07 01:56:03 UTC+0000
0×86970768 svchost.exe            3320    460      9      310      0      0 2022-12-07 01:56:03 UTC+0000
0×865a5b28 firefox.exe            3480   3408      0 --------      1      0 2022-12-07 02:04:02 UTC+0000   2022-12-07 02:04:08 UTC+0000
0×85107540 WmiPrvSE.exe           1104    604      6      110      0      0 2022-12-07 02:19:24 UTC+0000
0×8511f358 file_457151815         2108   1216      9       90      1      0 2022-12-07 02:21:49 UTC+0000
```

The last process was run as a part of kelihos binary and has the **PID of 2108.**

From here on performing a netscan in the dump file may help us getting the network connections.



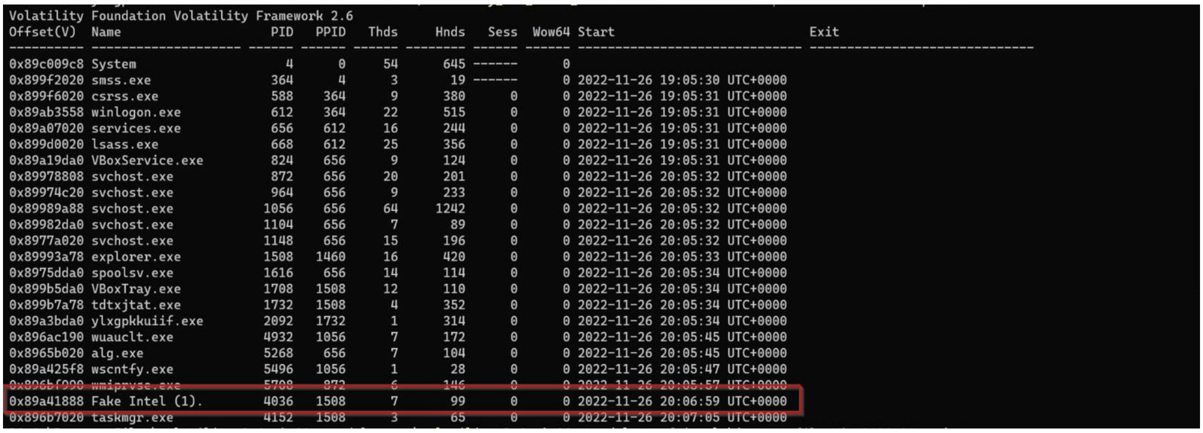
```
+0000
0×dfc11008    TCPv4    10.0.2.15:49217        140.123.33.139:80    SYN_SENT       2108    file_457151815
0×dfc60700    TCPv4    10.0.2.15:49193        37.157.220.7:80      CLOSED         0
0×dfc61ad0    TCPv4    127.0.0.1:49215        127.0.0.1:49216      ESTABLISHED    2108    file_457151815
0×dfdacdf8    TCPv4    -:49196                78.31.229.184:80     CLOSED         2108    file_457151815

   (root㉿kali)-[/opt/volatility]
```

On analysing this we do see that a SYN Packet was sent to a foreign address 140.123.33.139 at port 80 assuming that it is a web server.

And hence code improvements to forecast to analyse such socket presence can help in detection of kelihos.

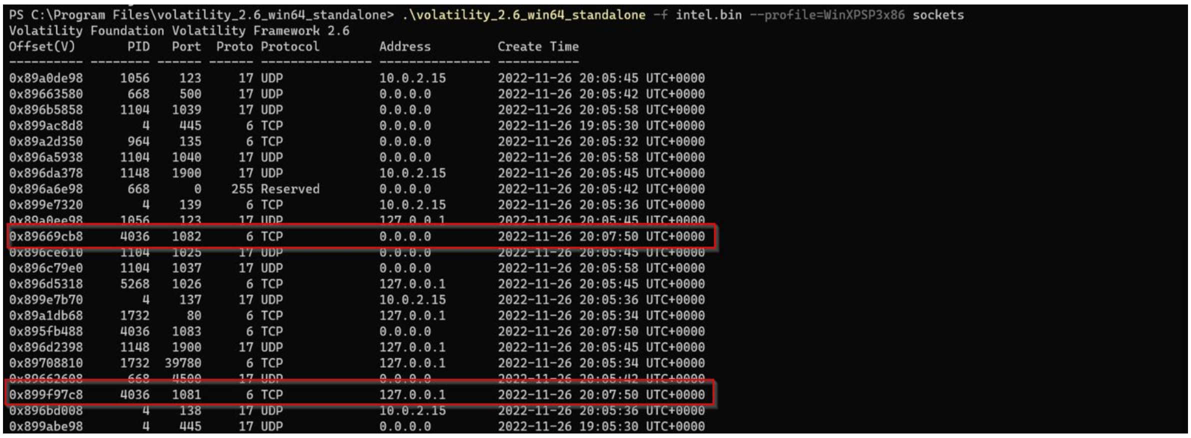
Windows XP Analysis:

Again, since we used volatility for analysis and it has specific and more specific commands for checking network connections as such, we performed the same dump file analysis for Kelihos by running it in a Windows XP system.



```
Volatility Foundation Volatility Framework 2.6
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------------------------------
0x89c009c8 System                    4      0     54      645 ------      0
0x899f2020 smss.exe                364      4      3       19 ------      0 2022-11-26 19:05:30 UTC+0000
0x899f6020 csrss.exe               588    364      9      380      0      0 2022-11-26 19:05:31 UTC+0000
0x89ab3558 winlogon.exe            612    364     22      515      0      0 2022-11-26 19:05:31 UTC+0000
0x89a07020 services.exe            656    612     16      244      0      0 2022-11-26 19:05:31 UTC+0000
0x899d0020 lsass.exe               668    612     25      356      0      0 2022-11-26 19:05:31 UTC+0000
0x89a19da0 VBoxService.exe         824    656      9      124      0      0 2022-11-26 19:05:31 UTC+0000
0x89978808 svchost.exe             872    656     20      201      0      0 2022-11-26 20:05:32 UTC+0000
0x89974c20 svchost.exe             964    656      9      233      0      0 2022-11-26 20:05:32 UTC+0000
0x89989a88 svchost.exe            1056    656     64     1242      0      0 2022-11-26 20:05:32 UTC+0000
0x89982da0 svchost.exe            1104    656      7       89      0      0 2022-11-26 20:05:32 UTC+0000
0x8977a020 svchost.exe            1148    656     15      196      0      0 2022-11-26 20:05:32 UTC+0000
0x89993a78 explorer.exe           1508   1460     16      420      0      0 2022-11-26 20:05:33 UTC+0000
0x8975dda0 spoolsv.exe            1616    656     14      114      0      0 2022-11-26 20:05:34 UTC+0000
0x899b5da0 VBoxTray.exe           1708   1508     12      110      0      0 2022-11-26 20:05:34 UTC+0000
0x899b7a78 tdtxjtat.exe           1732   1508      4      352      0      0 2022-11-26 20:05:34 UTC+0000
0x89a3bda0 ylxgpkkuiif.exe        2092   1732      1      314      0      0 2022-11-26 20:05:34 UTC+0000
0x896ac190 wuauclt.exe            4932   1056      7      172      0      0 2022-11-26 20:05:45 UTC+0000
0x8965b020 alg.exe                5268    656      7      104      0      0 2022-11-26 20:05:45 UTC+0000
0x89a425f8 wscntfy.exe            5496   1056      1       28      0      0 2022-11-26 20:05:47 UTC+0000
0x896bf990 wmiprvse.exe           5708    872      6      146      0      0 2022-11-26 20:05:57 UTC+0000
0x89a41888 Fake Intel (1).        4036   1508      7       99      0      0 2022-11-26 20:06:59 UTC+0000
0x896b7020 taskmgr.exe            4152   1508      3       65      0      0 2022-11-26 20:07:05 UTC+0000
```

We note the PID of the binary **<u>to be 4036</u>**

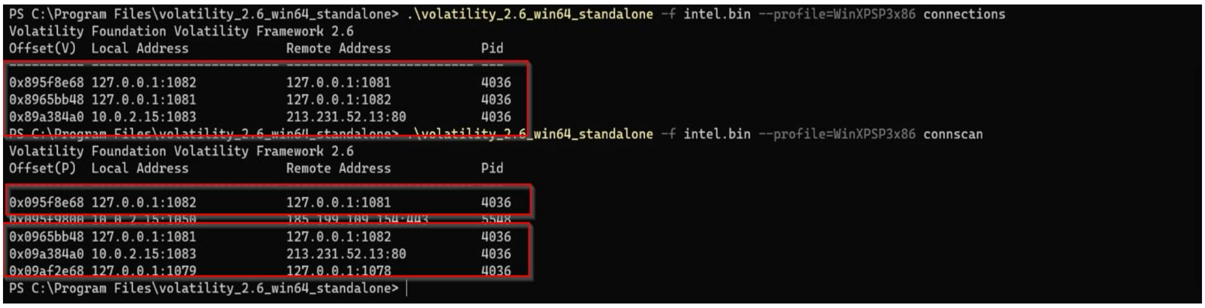


```
PS C:\Program Files\volatility_2.6_win64_standalone> .\volatility_2.6_win64_standalone -f intel.bin --profile=WinXPSP3x86 sockets
Volatility Foundation Volatility Framework 2.6
Offset(V)       PID   Port  Proto Protocol        Address         Create Time
---------- -------- ------ ------ --------------- --------------- -----------
0x89a0de98     1056    123     17 UDP             10.0.2.15       2022-11-26 20:05:45 UTC+0000
0x89663580      668    500     17 UDP             0.0.0.0         2022-11-26 20:05:42 UTC+0000
0x896b5858     1104   1039     17 UDP             0.0.0.0         2022-11-26 20:05:58 UTC+0000
0x899ac8d8        4    445      6 TCP             0.0.0.0         2022-11-26 19:05:30 UTC+0000
0x89a2d350      964    135      6 TCP             0.0.0.0         2022-11-26 20:05:32 UTC+0000
0x896a5938     1104   1040     17 UDP             0.0.0.0         2022-11-26 20:05:58 UTC+0000
0x896da378     1148   1900     17 UDP             10.0.2.15       2022-11-26 20:05:45 UTC+0000
0x896a6e98      668      0    255 Reserved        0.0.0.0         2022-11-26 20:05:42 UTC+0000
0x899e7320        4    139      6 TCP             10.0.2.15       2022-11-26 20:05:36 UTC+0000
0x89a0ee98     1056    123     17 UDP             127.0.0.1       2022-11-26 20:05:45 UTC+0000
0x89669cb8     4036   1082      6 TCP             0.0.0.0         2022-11-26 20:07:50 UTC+0000
0x896ce610     1104   1025     17 UDP             0.0.0.0         2022-11-26 20:05:45 UTC+0000
0x896c79e0     1104   1037     17 UDP             0.0.0.0         2022-11-26 20:05:58 UTC+0000
0x896d5318     5268   1026      6 TCP             127.0.0.1       2022-11-26 20:05:45 UTC+0000
0x899e7b70        4    137     17 UDP             10.0.2.15       2022-11-26 20:05:36 UTC+0000
0x89a1db68     1732     80      6 TCP             127.0.0.1       2022-11-26 20:05:34 UTC+0000
0x895fb488     4036   1083      6 TCP             0.0.0.0         2022-11-26 20:07:50 UTC+0000
0x896d2398     1148   1900     17 UDP             127.0.0.1       2022-11-26 20:05:45 UTC+0000
0x89708810     1732  39780      6 TCP             127.0.0.1       2022-11-26 20:05:34 UTC+0000
0x89662608      668   4500     17 UDP             0.0.0.0         2022-11-26 20:05:42 UTC+0000
0x899f97c8     4036   1081      6 TCP             127.0.0.1       2022-11-26 20:07:50 UTC+0000
0x896bd008        4    138     17 UDP             10.0.2.15       2022-11-26 20:05:36 UTC+0000
0x899abe98        4    445     17 UDP             0.0.0.0         2022-11-26 19:05:30 UTC+0000
```

We check for the availability of the sockets and if there are any opened by the PID 4036 that's the binary we are analysing.

We see **<u>open sockets on port 1081 and 1082</u>**.

And we do want to see if there any remote connections that existed in this case.

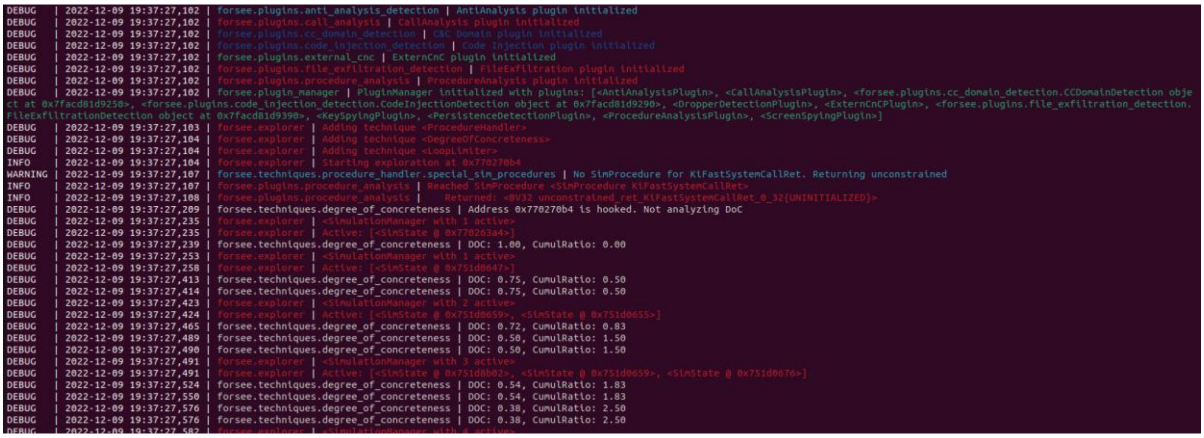


```
PS C:\Program Files\volatility_2.6_win64_standalone> .\volatility_2.6_win64_standalone -f intel.bin --profile=WinXPSP3x86 connections
Volatility Foundation Volatility Framework 2.6
Offset(V)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x895f8e68 127.0.0.1:1082            127.0.0.1:1081            4036
0x8965bb48 127.0.0.1:1081            127.0.0.1:1082            4036
0x89a384a0 10.0.2.15:1083            213.231.52.13:80          4036
PS C:\Program Files\volatility_2.6_win64_standalone> .\volatility_2.6_win64_standalone -f intel.bin --profile=WinXPSP3x86 connscan
Volatility Foundation Volatility Framework 2.6
Offset(P)  Local Address             Remote Address            Pid

0x095f8e68 127.0.0.1:1082            127.0.0.1:1081            4036
0x095f9800 10.0.2.15:1050            185.199.109.154:443       5548
0x0965bb48 127.0.0.1:1081            127.0.0.1:1082            4036
0x09a384a0 10.0.2.15:1083            213.231.52.13:80          4036
0x09af2e68 127.0.0.1:1079            127.0.0.1:1078            4036
PS C:\Program Files\volatility_2.6_win64_standalone>
```

And that is confirmed through the outputs of the **<u>connections and connscan</u>**.

Forecast Analysis:

In this part, the malware was run on a 32-bit Windows 7 virtual machine, and the malware dump file was created by task manager of Windows system (and named as "FakeIntel.DMP"). After setting up Forecast and creating virtual environment in an Ubuntu 18.04 virtual machine, we can ask Forecast to analyse the dump file and then get (part of) the result seen in pages below.



```
DEBUG   | 2022-12-09 19:37:27,102 | forsee.plugins.anti_analysis_detection | AntiAnalysis plugin initialized
DEBUG   | 2022-12-09 19:37:27,102 | forsee.plugins.call_analysis | CallAnalysis plugin initialized
DEBUG   | 2022-12-09 19:37:27,102 | forsee.plugins.cc_domain_detection | C&C Domain plugin initialized
DEBUG   | 2022-12-09 19:37:27,102 | forsee.plugins.code_injection_detection | Code Injection plugin initialized
DEBUG   | 2022-12-09 19:37:27,102 | forsee.plugins.external_cnc | ExternCnC plugin initialized
DEBUG   | 2022-12-09 19:37:27,102 | forsee.plugins.file_exfiltration_detection | FileExfiltration plugin initialized
DEBUG   | 2022-12-09 19:37:27,102 | forsee.plugins.procedure_analysis | ProcedureAnalysis plugin initialized
DEBUG   | 2022-12-09 19:37:27,102 | forsee.plugin_manager | PluginManager initialized with plugins: [<AntiAnalysisPlugin>, <CallAnalysisPlugin>, <forsee.plugins.cc_domain_detection.CCDomainDetection obje
ct at 0x7facd81d9250>, <forsee.plugins.code_injection_detection.CodeInjectionDetection object at 0x7facd81d9290>, <DropperDetectionPlugin>, <ExternCnCPlugin>, <forsee.plugins.file_exfiltration_detection.
FileExfiltrationDetection object at 0x7facd81d9390>, <KeySpyingPlugin>, <PersistenceDetectionPlugin>, <ProcedureAnalysisPlugin>, <ScreenSpyingPlugin>]
DEBUG   | 2022-12-09 19:37:27,103 | forsee.explorer | Adding technique <ProcedureHandler>
DEBUG   | 2022-12-09 19:37:27,104 | forsee.explorer | Adding technique <DegreeOfConcreteness>
DEBUG   | 2022-12-09 19:37:27,104 | forsee.explorer | Adding technique <LoopLimiter>
INFO    | 2022-12-09 19:37:27,104 | forsee.explorer | Starting exploration at 0x770270b4
WARNING | 2022-12-09 19:37:27,107 | forsee.techniques.procedure_handler.special_sim_procedures | No SimProcedure for KiFastSystemCallRet. Returning unconstrained
INFO    | 2022-12-09 19:37:27,107 | forsee.plugins.procedure_analysis | Reached SimProcedure <SimProcedure KiFastSystemCallRet>
INFO    | 2022-12-09 19:37:27,108 | forsee.plugins.procedure_analysis |     Returned: <BV32 unconstrained_ret_KiFastSystemCallRet_0_32{UNINITIALIZED}>
DEBUG   | 2022-12-09 19:37:27,209 | forsee.techniques.degree_of_concreteness | Address 0x770270b4 is hooked. Not analyzing DoC
DEBUG   | 2022-12-09 19:37:27,235 | forsee.explorer | <SimulationManager with 1 active>
DEBUG   | 2022-12-09 19:37:27,235 | forsee.explorer | Active: [<SimState @ 0x770263a4>]
DEBUG   | 2022-12-09 19:37:27,239 | forsee.techniques.degree_of_concreteness | DOC: 1.00, CumulRatio: 0.00
DEBUG   | 2022-12-09 19:37:27,253 | forsee.explorer | <SimulationManager with 1 active>
DEBUG   | 2022-12-09 19:37:27,258 | forsee.explorer | Active: [<SimState @ 0x751d0647>]
DEBUG   | 2022-12-09 19:37:27,413 | forsee.techniques.degree_of_concreteness | DOC: 0.75, CumulRatio: 0.50
DEBUG   | 2022-12-09 19:37:27,414 | forsee.techniques.degree_of_concreteness | DOC: 0.75, CumulRatio: 0.50
DEBUG   | 2022-12-09 19:37:27,423 | forsee.explorer | <SimulationManager with 2 active>
DEBUG   | 2022-12-09 19:37:27,424 | forsee.explorer | Active: [<SimState @ 0x751d0659>, <SimState @ 0x751d0655>]
DEBUG   | 2022-12-09 19:37:27,465 | forsee.techniques.degree_of_concreteness | DOC: 0.72, CumulRatio: 0.83
DEBUG   | 2022-12-09 19:37:27,489 | forsee.techniques.degree_of_concreteness | DOC: 0.50, CumulRatio: 1.50
DEBUG   | 2022-12-09 19:37:27,490 | forsee.techniques.degree_of_concreteness | DOC: 0.50, CumulRatio: 1.50
DEBUG   | 2022-12-09 19:37:27,491 | forsee.explorer | <SimulationManager with 3 active>
DEBUG   | 2022-12-09 19:37:27,491 | forsee.explorer | Active: [<SimState @ 0x751d8b02>, <SimState @ 0x751d0659>, <SimState @ 0x751d0676>]
DEBUG   | 2022-12-09 19:37:27,524 | forsee.techniques.degree_of_concreteness | DOC: 0.54, CumulRatio: 1.83
DEBUG   | 2022-12-09 19:37:27,550 | forsee.techniques.degree_of_concreteness | DOC: 0.54, CumulRatio: 1.83
DEBUG   | 2022-12-09 19:37:27,576 | forsee.techniques.degree_of_concreteness | DOC: 0.38, CumulRatio: 2.50
DEBUG   | 2022-12-09 19:37:27,576 | forsee.techniques.degree_of_concreteness | DOC: 0.38, CumulRatio: 2.50
DEBUG   | 2022-12-09 19:37:27,582 | forsee.explorer | <SimulationManager with 4 active>
```

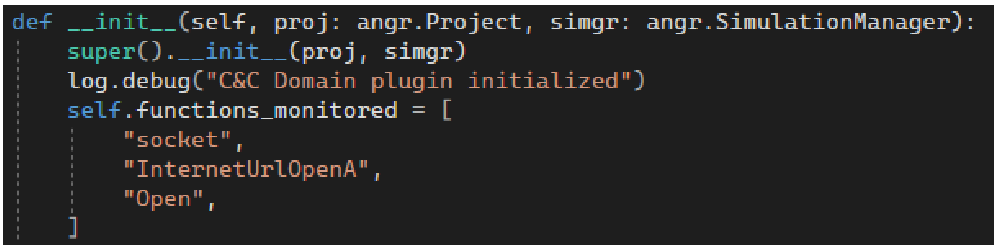
The full result is too long to be fully captured in this report. Unfortunately, Forecast failed to detected the socket connection, which should have triggered Forecast "C&C Domain" plugin to warn the user that C&C domain is detected.

Help Forecast Perform Better:

We can do some modifications to Forecast code to make Forecast perform better in socket detection (as well as some other malware function detection). Before beginning code modification, let's take a quick look at how Forecast is supposed to do when detecting sockets established by the malware.

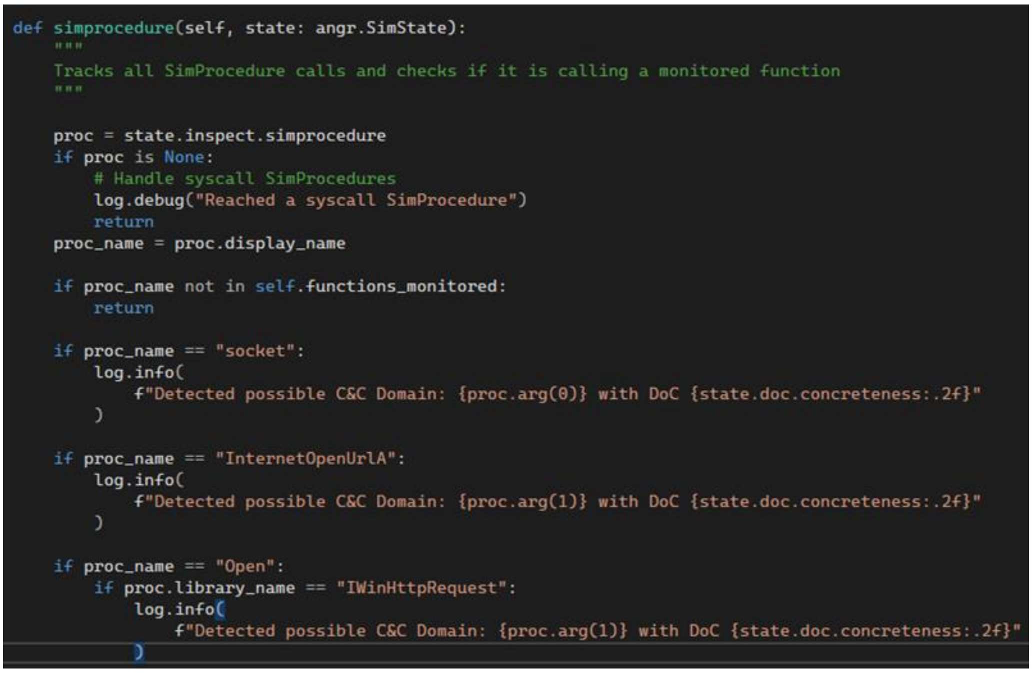
(1) How does Plugin works

For most of the plugins in Forecast directory, it has a list of "functions monitored", which contains a list of function name that related to the malware activity this plugin is taking responsibility for. For instance, function "`socket`", "`InternetUrlOpenA`" and "`Open`" are three functions that being monitored by plugin "cc_domain_dectection". It can be seen in class `CCDomainDetection` in `Forecast\forsee\plugins\cc_domain_detection.py`



```
def __init__(self, proj: angr.Project, simgr: angr.SimulationManager):
    super().__init__(proj, simgr)
    log.debug("C&C Domain plugin initialized")
    self.functions_monitored = [
        "socket",
        "InternetUrlOpenA",
        "Open",
    ]
```

Under the same class `CCDomainDetection`, there's a function named `simprocedure` (the code can be seen in the figure below), which acts as an intermedia of detected malware function and the log information presented to the user. Basically, what the function does is taking a variable that represents a function detected by Forecast underlying tools (for instance, angr in this case), and checking whether this function is a member of the function list that it is monitoring. If so, a corresponding waring log information will be presented to the user, and if not, nothing would be displayed on the command window and the program continues to explore the dump file.



```
def simprocedure(self, state: angr.SimState):
    """
    Tracks all SimProcedure calls and checks if it is calling a monitored function
    """

    proc = state.inspect.simprocedure
    if proc is None:
        # Handle syscall SimProcedures
        log.debug("Reached a syscall SimProcedure")
        return
    proc_name = proc.display_name

    if proc_name not in self.functions_monitored:
        return

    if proc_name == "socket":
        log.info(
            f"Detected possible C&C Domain: {proc.arg(0)} with DoC {state.doc.concreteness:.2f}"
        )

    if proc_name == "InternetOpenUrlA":
        log.info(
            f"Detected possible C&C Domain: {proc.arg(1)} with DoC {state.doc.concreteness:.2f}"
        )

    if proc_name == "Open":
        if proc.library_name == "IWinHttpRequest":
            log.info(
                f"Detected possible C&C Domain: {proc.arg(1)} with DoC {state.doc.concreteness:.2f}"
            )
```

In a larger scale, while exploring the dump file, as soon as an underlying tool detected a malware function, it sends the function name to every plugin that works with this mechanism. If one of the plugins finds that the function being presented is one of the functions that it is monitoring, a serious of if statements in the plugin code would lead to a corresponding waring log on the command window.

(2) Why does this happen

Considering the misfunction of Forecast in this situation, it might be caused by either the three reasons:

(i) The plugin isn't initialized.

(ii) The plugin doesn't get its desired function, so that nothing is shown out.

(iii) The function is presented properly, it's the plugin itself that doesn't perform properly.

As we can see at the very beginning of Forecast outputted result, the log

```
forsee.plugins.cc_domian_detection | C&C Domain plugin initialized
```

indicates that C&C domain detection plugin is initialized, which means the first assumption is a false statement.

To check the other two assumptions, I collected all functions that being presented to these plugins, they are

```
KiFastSystemCallRet
```

```
TranslateMessage
```

```
DispatchMessageA
```

```
GetMessageA
```

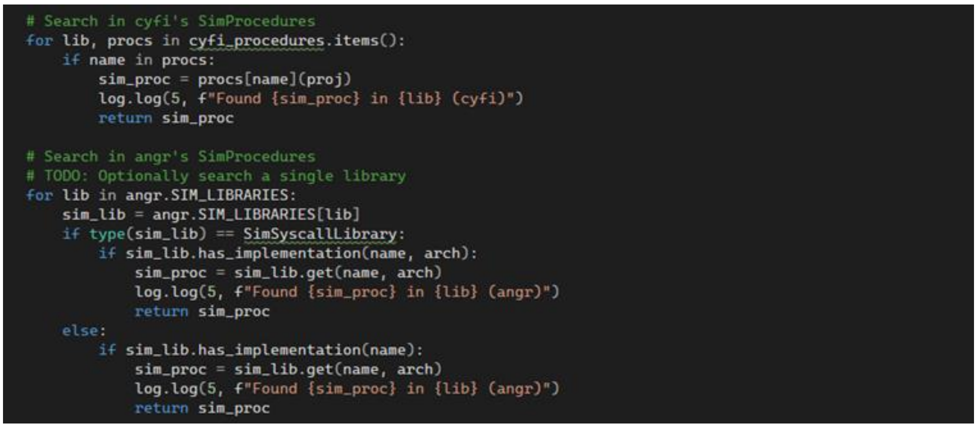
and some of these functions are presented multiple times. The plugin never receives a function named "socket", no wonder why it fails to warn the user that C&C domain has been detection. To make sure that the plugin can output what is supposed to say when a function is in the list is presented properly, I added the code

```
proc_name = "socket"
```

just before the if statements, which means the variable "proc_name" that originally represents the function presented from the underlying tool, is changed to a fixed string "socket", and the modified value will be brought to the following if statements. In this case, I got the desired output from the plugin, which means assumption (ii) is the problem we are going to solve, and it could be the entry point to make Forecast perform better.

Does Forecast fail to detect socket function while exploring the dump file? The misfunction of C&C domain detection plugin may seem to be hard to alleviate if this is the case. Luckily, the situation is much better than the worst case, because we can find another source of function list derived from the dump file in somewhere else.

There's a function `find_sim_procedure` in class `ExportManager` in the file `\Forecast\forsee\techniques\procedure_handler\procedure_handler.py`, which contains the code below



```
# Search in cyfi's SimProcedures
for lib, procs in cyfi_procedures.items():
    if name in procs:
        sim_proc = procs[name](proj)
        log.log(5, f"Found {sim_proc} in {lib} (cyfi)")
        return sim_proc

# Search in angr's SimProcedures
# TODO: Optionally search a single library
for lib in angr.SIM_LIBRARIES:
    sim_lib = angr.SIM_LIBRARIES[lib]
    if type(sim_lib) == SimSyscallLibrary:
        if sim_lib.has_implementation(name, arch):
            sim_proc = sim_lib.get(name, arch)
            log.log(5, f"Found {sim_proc} in {lib} (angr)")
            return sim_proc
    else:
        if sim_lib.has_implementation(name):
            sim_proc = sim_lib.get(name, arch)
            log.log(5, f"Found {sim_proc} in {lib} (angr)")
            return sim_proc
```

By back tracing the code, we can find that the variable `name` is a name of a specific function Forecast detects while exploring the dump file. The log is set not to display so that user can't see it in the command window.

By changing the first parameter in the log function from 5 to 50

`log.log(50, f"Found {sim_proc} in {lib} (cyfi)")`

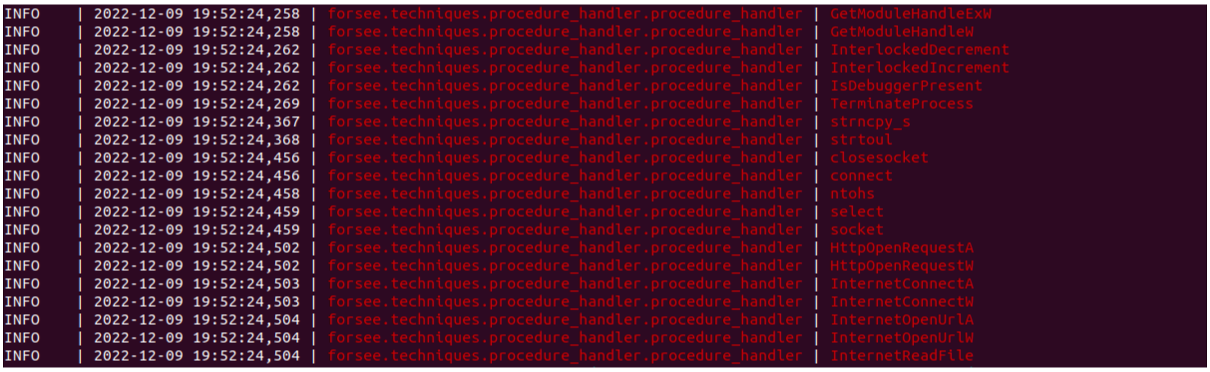
`log.log(50, f"Found {sim_proc} in {lib} (angr)")`

`log.log(50, f"Found {sim_proc} in {lib} (angr)")`

we can change the log information to highest priority and can see what it has got from the Forecast output information. It turns out that there are a number of functions that Forecast does detect, but aren't sent to plugins to process. We can add the code

`log.info(name)`

in each if statement to display all functions the program has got. The figure below shows part of the output of procedure_handler.py after code modification.



```
INFO     | 2022-12-09 19:52:24,258 | forsee.techniques.procedure_handler.procedure_handler | GetModuleHandleExW
INFO     | 2022-12-09 19:52:24,258 | forsee.techniques.procedure_handler.procedure_handler | GetModuleHandleW
INFO     | 2022-12-09 19:52:24,262 | forsee.techniques.procedure_handler.procedure_handler | InterlockedDecrement
INFO     | 2022-12-09 19:52:24,262 | forsee.techniques.procedure_handler.procedure_handler | InterlockedIncrement
INFO     | 2022-12-09 19:52:24,262 | forsee.techniques.procedure_handler.procedure_handler | IsDebuggerPresent
INFO     | 2022-12-09 19:52:24,269 | forsee.techniques.procedure_handler.procedure_handler | TerminateProcess
INFO     | 2022-12-09 19:52:24,367 | forsee.techniques.procedure_handler.procedure_handler | strncpy_s
INFO     | 2022-12-09 19:52:24,368 | forsee.techniques.procedure_handler.procedure_handler | strtoul
INFO     | 2022-12-09 19:52:24,456 | forsee.techniques.procedure_handler.procedure_handler | closesocket
INFO     | 2022-12-09 19:52:24,456 | forsee.techniques.procedure_handler.procedure_handler | connect
INFO     | 2022-12-09 19:52:24,458 | forsee.techniques.procedure_handler.procedure_handler | ntohs
INFO     | 2022-12-09 19:52:24,459 | forsee.techniques.procedure_handler.procedure_handler | select
INFO     | 2022-12-09 19:52:24,459 | forsee.techniques.procedure_handler.procedure_handler | socket
INFO     | 2022-12-09 19:52:24,502 | forsee.techniques.procedure_handler.procedure_handler | HttpOpenRequestA
INFO     | 2022-12-09 19:52:24,502 | forsee.techniques.procedure_handler.procedure_handler | HttpOpenRequestW
INFO     | 2022-12-09 19:52:24,503 | forsee.techniques.procedure_handler.procedure_handler | InternetConnectA
INFO     | 2022-12-09 19:52:24,503 | forsee.techniques.procedure_handler.procedure_handler | InternetConnectW
INFO     | 2022-12-09 19:52:24,504 | forsee.techniques.procedure_handler.procedure_handler | InternetOpenUrlA
INFO     | 2022-12-09 19:52:24,504 | forsee.techniques.procedure_handler.procedure_handler | InternetOpenUrlW
INFO     | 2022-12-09 19:52:24,504 | forsee.techniques.procedure_handler.procedure_handler | InternetReadFile
```

We can see that the function socket is detected, this source of function detection seems to be a better one compared to the one that the plugins are using. If we could add this source of function to all the plugins, it might lead to an improvement of performance on malware detection and analysis.

(3) Help Forecast perform better

To begin with, we need to create a container to store the list of function that listed in `procedure_handler.py`.

I created a new class `FunctionList` in a new file `function_detected.py` under directory `/Forecast/forsee/techniques` to do the job. A python dictionary is defined in the class to collect all the function names `procedure_handler.py` has got. Also, an add function is defined in the class to add new detected function in the dictionary, and the add process will not be executed if the same function already exists in the dictionary.

```
   class FunctionList:
       dic = {'fucntion_name': 'function_name'}
       def add(the_name: str, the_list :dict = dic):
           if the_name in the_list.keys():
               return
           else:
               the_list[the_name] = the_name
               return
```

When we need this class, simply add

`from forsee.techniques.procedure_handler.function_detected import FunctionList`
at the top of the file to import this class to the file we want.

Then it’s time to feed all the plugin with this new source we have just modified. We are not going to change the source from previous one to this dictionary. Instead, we are going to add an additional source to each plugin to help it performs better.

Let’s take the plugin `cc_domain_detection` for example. Basically, within the function `simprocedure`, after comparing `function_monitored` with the original source and executing all the if statements to display corresponding result, we are going to make it compared to our new defined collection after dealing with the old source, and execute the set of if statements for information display again. This could be done by iteratively assigning the value of `proc_name`, which initially represents the function name provided by the default source, to function names collected in our collection, and then executing all the if statement to determine the output.

In order to avoid code repetition, it’s better to define a new function taking charge of displaying information based on what we have got. When it comes to determine what should be displayed on the output, just pass the function name to the display function and it will make the decision.

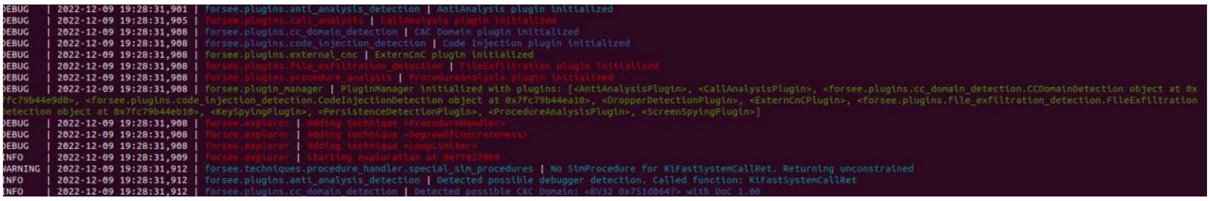
```
def saySomething(self, proc_name: str, state: angr.SimState):
        proc = state.inspect.simprocedure
        if proc_name not in self.functions_monitored:
            return

        if proc_name == "socket":
            log.info(
                f"Detected possible C&C Domain: {proc.arg(0)} with DoC
               {state.doc.concreteness:.2f}"
            )

        if proc_name == "InternetOpenUrlA":
            log.info(
                f"Detected possible C&C Domain: {proc.arg(1)} with DoC
               {state.doc.concreteness:.2f}"
            )
```

```
    if proc_name == "Open":
        if proc.library_name == "IWinHttpRequest":
            log.info(
                f"Detected possible C&C Domain: {proc.arg(1)} with DoC
              {state.doc.concreteness:.2f}"
            )
```

And the function `simprocedure` becomes

```
def simprocedure(self, state: angr.SimState):
    #Tracks all SimProcedure calls and checks if it is calling a monitored function
    proc = state.inspect.simprocedure
    if proc is None:
        # Handle syscall SimProcedures
        log.debug("Reached a syscall SimProcedure")
        return
    proc_name = proc.display_name
    self.saySomething(proc_name, state)
    for function, typ in FunctionList.dic.items():
        self.saySomething(typ, state)
```

For all other plugins that work with a list of `function_monitored`, we can do similar modification to add a new source for comparation. After all the modification are done, we can execute `run_minidump.py` with the same dump file sample.DMP, and see if there's any improvement on its performance.



```
DEBUG   | 2022-12-09 19:28:31,901 | forsee.plugins.anti_analysis_detection | AntiAnalysis plugin initialized
DEBUG   | 2022-12-09 19:28:31,905 | forsee.plugins.call_analysis | CallAnalysis plugin initialized
DEBUG   | 2022-12-09 19:28:31,908 | forsee.plugins.cc_domain_detection | C&C Domain plugin initialized
DEBUG   | 2022-12-09 19:28:31,908 | forsee.plugins.code_injection_detection | Code Injection plugin initialized
DEBUG   | 2022-12-09 19:28:31,908 | forsee.plugins.external_cnc | ExternCnC plugin initialized
DEBUG   | 2022-12-09 19:28:31,908 | forsee.plugins.file_exfiltration_detection | FileExfiltration plugin initialized
DEBUG   | 2022-12-09 19:28:31,908 | forsee.plugins.procedure_analysis | ProcedureAnalysis plugin initialized
DEBUG   | 2022-12-09 19:28:31,908 | forsee.plugin_manager | PluginManager initialized with plugins: [<AntiAnalysisPlugin>, <CallAnalysisPlugin>, <forsee.plugins.cc_domain_detection.CCDomainDetection object at 0x
7fc79b44e9d0>, <forsee.plugins.code_injection_detection.CodeInjectionDetection object at 0x7fc79b44ea10>, <DropperDetectionPlugin>, <ExternCnCPlugin>, <forsee.plugins.file_exfiltration_detection.FileExfiltration
Detection object at 0x7fc79b44eb10>, <KeySpyingPlugin>, <PersistenceDetectionPlugin>, <ProcedureAnalysisPlugin>, <ScreenSpyingPlugin>]
DEBUG   | 2022-12-09 19:28:31,908 | forsee.explorer | Adding technique <ProcedureHandler>
DEBUG   | 2022-12-09 19:28:31,908 | forsee.explorer | Adding technique <DegreeOfConcreteness>
DEBUG   | 2022-12-09 19:28:31,908 | forsee.explorer | Adding technique <LoopLimiter>
INFO    | 2022-12-09 19:28:31,909 | forsee.explorer | Starting exploration at 0x770270b4
WARNING | 2022-12-09 19:28:31,912 | forsee.techniques.procedure_handler.special_sim_procedures | No SimProcedure for KiFastSystemCallRet. Returning unconstrained
INFO    | 2022-12-09 19:28:31,912 | forsee.plugins.anti_analysis_detection | Detected possible debugger detection. Called function: KiFastSystemCallRet
INFO    | 2022-12-09 19:28:31,912 | forsee.plugins.cc_domain_detection | Detected possible C&C Domain: <BV32 0x751d0647> with DoC 1.00
```

Luckily, there're new findings listed in Forecast output after modification. The function `socket` is detected, and it triggers a warning message. Meanwhile, the plugin `anti_analysis detection` also finds a function being provided by the new source is among its monitoring list, and another waring message comes out unsurprisingly.

What needs to be pointed out is that, the code modification above is just adding new analyzing approach in the program, it doesn't remove any existed functionality in Forecast code.