Experimental Setup Description

I. SoC Board Setup and Emulated ETB Trace Collection

To emulate ETB data collection from the Xilinx Zynq-7000 SoC ZC702 evaluation kit board, we gather Register information from Xilinx SDK 2017.3 while running actual malware.

To prepare the computer to connect to the board, we completed the following steps:

1) Set the computer's IP address to connect to the evaluation board.
2) Disabled the computer's firewall at the Ethernet port to connect to the evaluation board (if not, the firewall may block communication between the board and computer). Antivirus programs on the host computer may also block malware from running through Xilinx SDK. If this happens, we have two options: either temporarily disable the antivirus program or reconfigure its settings to allow the malware file to run.
3) Once the board is physically connected to the computer using the JTAG and Ethernet connections, we established a serial connection between the computer and board to manipulate the board using the computer. For this connection, we used the serial connection option in Xilinx SDK with a baud rate of 115200, no flow control, 8 data bits, 1 stop bit, 5 second timeout, and no parity bit. The port number depends on the number associated with the USB port connected to JTAG.

To prepare the evaluation board to connect to the computer and to the Xilinx System Debugger, we completed the following steps:

1) Downloaded a ZC702-compatible Linux OS image to the SD Card. we used $xilinx-zc702-2017_34.9.0-xilinx-v2017.3$, which was provided by Xilinx and generated using PetaLinux.
2) Boot the board in SD mode as described in the ZC702 user manual [1].
3) Through the serial connection to the computer, set the evaluation board's IP address (the IP address is required when running programs using the System Debugger).
4) Compiled all desired programs in Xilinx SDK, and launched these programs on the board using Debug Configurations. Set the target in Debug Configurations to the board IP address, which was set in the previous step. To access register data at different points in the program's runtime, we added breakpoints to the program. The register values update at each breakpoint.
5) In cases where malware requires a server and a client, we ran the server on the host computer and the client on the SoC board to collect client data (assuming that we would be detecting an infected botnet client and not a server).
6) In cases where the server and client cannot connect because the IP addresses do not match, we hardcoded the required IP addresses for the connection.
7) In cases where the server cannot send attack commands because the embedded OS does not have a requested functionality, we hardcoded the attack command into the client program.

II. Gathering HPC Data and Matching to ETB Data

To gather HPC data, we ran the same test programs (malware and benign programs) on the SoC using the perf command. To associate regions of HPC data with regions from the malware source code where we added the breakpoints (see previous section), we defined perf trace events (See [2] to see how to add trace events). We added the trace events to record when different regions of the source code were reached to identify which subsection of code was reached when a certain HPC value was dumped (because perf will record the event along with the HPC data).

Here is one example of a bash script for running the perf command to generate all of the frequencies of HTB data for the mirai generic udp attack(please note that this is assuming the user has already defined the custom perf events, which can be identified here by the probe_ prefix):

```
for i in 625 1250 2500 5000 10000
do
commandName=$i"
    FrequencyperfHPCETBRegionsmiraiudpgeneric
    "; outputNameAlmost=$commandName$j
    ".txt";
perf record -F $i -e probe_mirai:
    killer_init,probe_mirai:attack_init
    ,probe_mirai:attack_parse,probe\
    _mirai:attack\_start,probe_mirai:
    attack_udp_generic,probe_mirai:
    ensure_single_instance,probe\_mirai
    :killer\_init,probe_mirai:main,
    probe_mirai:table_init,probe_mirai:
    unlock_tbl_if_nodebug,probe_mirai:
    attack_init__return,probe_mirai:
    attack_parse__return,probe_mirai:
    attack_start__return,probe_mirai:
    attack_udp_generic__return,
    probe_mirai:
```

```
    ensure_single_instance__return ,
    probe_mirai:killer_init__return ,
    probe_mirai:main__return ,
    probe_mirai:table_init__return ,
    probe_mirai:
    unlock_tbl_if_nodebug__return ,
    branch-instructions ,branch-misses ,
    cache-misses ,cache-references ,cpu-
    cycles ,instructions ,stalled-cycles-
    backend ,stalled-cycles-frontend ,L1-
    dcache-load-misses ,L1-dcache-loads ,
    L1-dcache-store-misses ,L1-dcache-
    stores ,L1-icache-load-misses ,branch
    -loads ,branch-load-misses ,dTLB-load
    -misses ,dTLB-store-misses ,iTLB-load
    -misses ./mirai.elf;
perf script > $outputNameAlmost;
done;
```

References

[1] Xilinx. *ZC702 Evaluation Board for the Zynq-7000 XC7Z020 SoC User Guide*.

[2] Perf Wiki. *Jolsa Features Togle Event*.