Project 3: Malware Analysis

CS 6262

Sections:

Windows Malware Analysis

Scenario:

You got a malware sample from the wild! Your task is to discover what the malware does by analyzing it.

How do you discover the malware's behaviors? There are multiple ways of analyzing it but we'll be focusing on two ways: Static Analysis and Dynamic Analysis.

Static Analysis:

- Manual Reverse Engineering
- Programming binary analysis

Dynamic Analysis:

- Network behavioral tracing
- Runtime system behavioral tracing (File/Process/Thread/Registry)
- Symbolic Execution
- Fuzzing

In our scenario, you are going to analyze the given malware with tools that we provide. These tools help you to analyze the malware with static and dynamic analysis.

Objective:

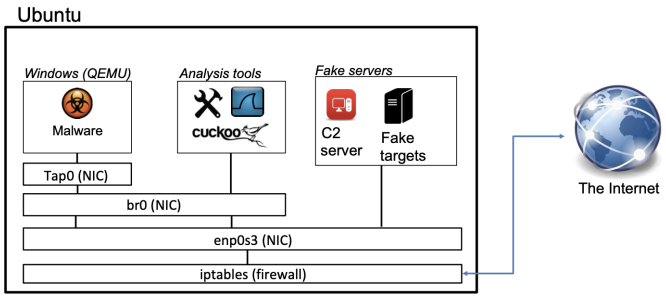
1. Find which server controls the malware (the command and control (C2) server)
2. Discover how the malware communicates with the command and control (C2) server
    a. URL and Payload
3. Discover what activities are done by the malware (Attack activities)

Requirement:

1. Make sure that no malware traffic goes out from virtual machine
2. The command and control server is dead, so YOU need to reconstruct it
    a. Use tools to reconstruct the server and then reveal hidden behaviors of the malware
3. Analyze network traffic on the host and figure out the list of available commands for the malware
4. Analyze network traffic and program trace of the host, and figure out what malware does
5. Write down your answer into **assignment-questionnaire.txt**

Project Structure:

- Make sure to install/update to the latest version of VirtualBox
    - https://www.virtualbox.org/wiki/Downloads
- Download the Virtual Machine (VM)
    - https://www.dropbox.com/s/dnk6acztw9ewp83/Project%203.zip?dl=0
    - Unarchive the file with 7zip and password is cs6262
- Network Configurations:
    - tap0:
        - Virtual network interface for Windows XP • IP Address: 192.168.133.101
    - br0
        - A network bridge between Windows XP and Ubuntu
            - IP Address: 192.168.133.1
    - enp0s3
        - A network that faces the Internet
            - IPAddress:10.0.2.15 (it varies with your VirtualBox settings)



- Open VirtualBox
    - Go to File → Import Appliance
    - Select the ova file and import it
    - For detailed information on how to import the VM, see:

    - Before starting, it might be useful to configure the settings, allocate more base memory, processors etc. to your VM, as per your device configurations for better performance.
- VM user credentials
    - **Username: analysis**
    - **Password: analysis**

NOTE: VM Setup

- For **M Series Mac Users**:
    - Please install the latest version of UTM (https://mac.getutm.app/) and follow the instructions in the link to import and set up the VM.
- **Warning**:
    - Due to different CPU architecture, running those windows programs based on X86 are very slow on a M Series Mac. It's your own choice to get a X86 machine by yourself or keep using the M Series Mac. If you really have difficulty finding a X86 machine, please contact TAs on Ed Discussion ASAP, we can provide alternative solutions such as cloud VMs to you.

- In the Virtual Machine:
    - Files
        - init.py
            - This initializes the project environment
                - Type your Georgia Tech username (your Canvas Login Name) after running this •

```
$./init.py
```

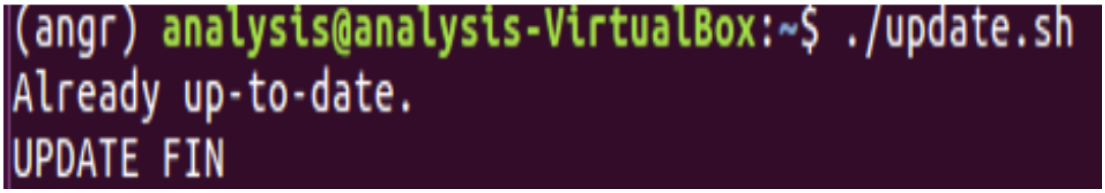
        - update.sh
            - This script updates the VM if any further update has been made by TAs
            - **Note**:
                - Please run this script when you start the project! (If it says that you're already updated when you run it, that's fine)
                - If you have already completed stage 1 before running update.sh, you do NOT need to redo stage 1 – but you will need to run update.sh to complete stage 2
        - archive.sh
            - This will archive the answer sheet for submission (create a zip file)
    - Directories:
        - vm
            - A directory that stores the Windows XP virtual machine (runs with QEMU)
            - We use the given VM for both Cuckoo and a testbed.

        - shared
            - A shared directory between the Ubuntu host and Windows guest (XP is running on a VM within your project VM). You can copy/move files to or from this directory.
        - report
            - The answer sheet for project questionnaire
        - setup
            - Required files for setting up the machine. You don't need to modify, nor use the files in this directory.
        - Tools
            - network
                - Configure your network firewall rules (iptables) by editing iptables-rules.
                - You can allow/disallow/redirect the traffic from the malware
                - '`./reset`' command in this directory will apply the changes
            - cfg-generation (CFG stands for Control-Flow Graph)
                - An analysis tool that helps you to find interesting functions of malicious activity
                - You need to edit score.h to generate the control-flow graph
                - Use `xdot` to open the generated CFG.
            - Sym-exec
                - A symbolic executor (based on angr : https://github.com/angr)
                    - Helps you to figure out the commands that malware expects
                - Use cfg-generation tool to figure out the address of the function of interests
            - c2-command
                - A simplified tool for C2 server reconstruction
                - You can write down command in the *.txt file as a line
                - It will randomly choose command at a time to send to the malware
    - Malware:
        - stage1.exe – stage 1 malware
            - It will download the stage 2 malware if this malware receives the correct command
        - stage2.exe – stage 2 malware
            - It will download the stage 3 malware if this malware receives the correct command
        - payload.exe – the linux malware attack payload
            - Analyze the dynamic instruction trace
            - Write a script to detect where the C&C communication happens – Find the loop entry point and function sequence in the loop
            - Add constraint to symbolic execution to limit the loop to one

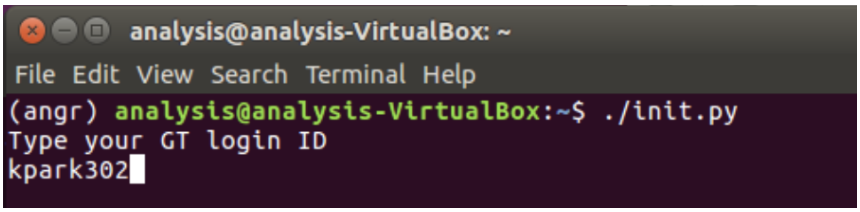
- Find the feasible attacks within a given set of possible attacks.

Tutorials:

- stage1.exe malware
  - Update the project 3 before begin
    - Open the terminal (`Ctrl-Alt-T`, or choose terminal from the menu)
    - Run `./update.sh`
      - It will update any necessary files that are required for this project.



  - Initializing the project
    - Open the terminal (`Ctrl-Alt-T`, or choose terminal from the menu)
    - Run `./init.py`
      - Type your Georgia Tech username (the login name used for Canvas)
      - This will download the stage1 malware (stage1.exe) into the ~/shared directory



  - **Note**:
    - These are malware samples hosted under the Georgia Tech Network
      - It is likely that security measures would kick in and encrypt these files
        - That is all the malware samples you will be downloading during this project
    - **IMPORTANT**
      - After each download, make sure to check the type of file
      - In the linux VM, execute

```
$ file <path-to-exe>
```

      - If the result of that is an archive of some sort then execute:

```
unzip <path-to-exe>
```

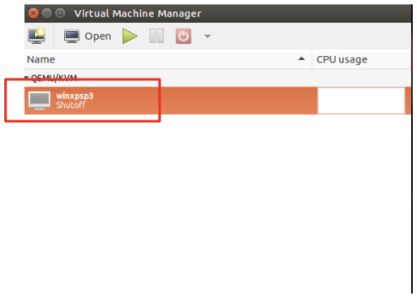
    - Password: infected
- For stage1 and stage2, the file format should be

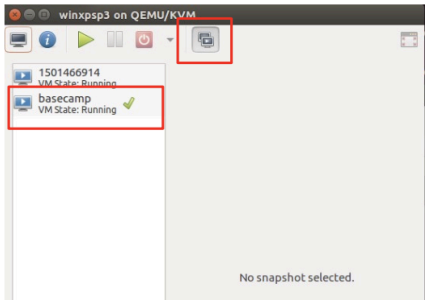
```
(angr) analysis@analysis-VirtualBox:~/shared$ file stage1.exe
stage1.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

- For stage3, the file format should be

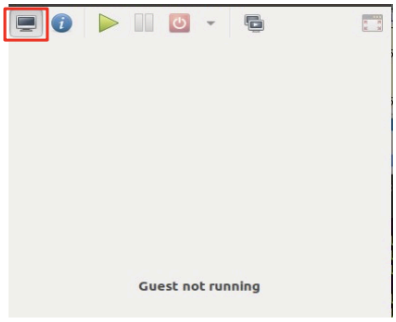
```
(angr) analysis@analysis-VirtualBox:~/shared$ file payload
payload: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, not stripped
```

- Secure Experiment Environment
    - We need a secure experiment environment to execute the malware
    - Why?
        - Insecure analysis environment could damage your system
        - You may not want:
            - Encrypting your file during a ransomware analysis
            - Infecting machines in your corporate network during a worm analysis
            - Creating a tons of infected bot client in your network during a bot/trojan analysis
    - The solution:
        - Contain malware in a virtual environment
            - Virtual Machine
            - Virtual Network
                - Conservative rules(allow network traffic only if it is secure)
    - We provide a Win XP VM as a testbed!
- Run Win XP VM
    - Run Windows XP Virtual Machine with virt-manager
    - Open a terminal
    - Type "`virt-manager`" and double click "`winxpsp3`"
    - Click the icon with the two monitors and click on "basecamp"



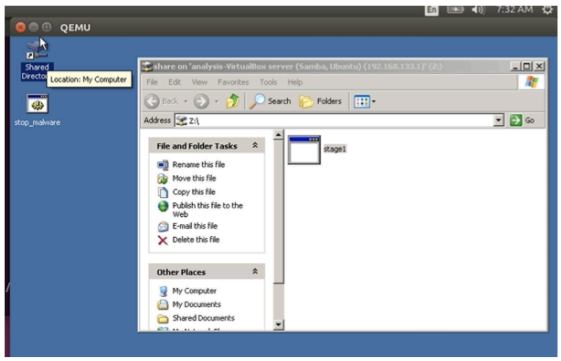


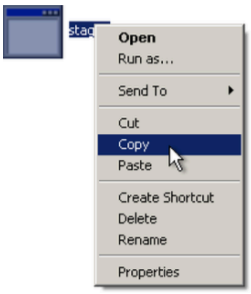
- Right click on basecamp, and click "Start snapshot." Click Yes if prompted.
- Once, virt-manager successfully calls the snapshot, click Show the graphical console.
  - Click on the Windows Start Menu and Turn off Computer.
  - Then select Restart

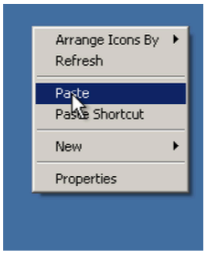


- **DO NOT MODIFY OR DELETE THE GIVEN SNAPSHOTS!**
  - The given snapshots are your backups for your analysis.
  - If something bad happens on your testbed, always revert back to the basecamp snapshot.

- Copy from Shared Directory
  - Go to the shared directory by clicking its icon (in Windows XP)
    - Copy stage1.exe into Desktop
    - If you execute it in the shared directory, the error message will pop up. Please copy the file to Desktop.







- Run the malware
  - Now we will run the malware
    - Execute stage1.exe (double click the icon)
    - It will say “Executing Stage 1 Malware”. Then, click OK.
      - You should click OK on each dialog to dismiss it
        - Otherwise, malware execution will be blocked
  - If you want to halt the malware that is running…
    - Execute stop_malware in the temp directory.
      - This will stop the currently running malware.
      - Please halt first before you execute another malware file.

- Network Behavioral Analysis……..
  - To analyze network behaviors, you need
    - Wireshark (https://www.wireshark.org/)
      - Network Protocol Analyzer
  - Cuckoo (https://cuckoosandbox.org/)
    - Capturing & Recording inbound/outbound network packets
- Observing Network Behavior
  - By capturing and recording network packets through the tools
    - Reveal C&C protocol
    - Attack Source & Destination
  - **But, malware will not do anything. Why?**
    - The C2 server is dead!
    - Therefore, the malware (C2 client) will never unfold its behaviors.
    - Question?
      - If we know C&C dialog of malware, can we build a fake C2 server in order to unfold the malware behaviors?
      - **Answer**: Hack Yeah! That is your job for this project!
- Wireshark
  - Let's check it through network monitoring
    - Everything has been already installed.
    - Open Wireshark, capture the traffic for the network bridge (Make sure to run with root privileges)
    - IP address = 192.168.133.1
    - Reference: https://www.wireshark.org/docs/
    - Get yourself familiarized with Linux commands and how to employ Wireshark.
    - Other references:
      - https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html
      - https://www.varonis.com/blog/how-to-use-wireshark
- Redirect Network Connection
  - From WireShark, we can notice that the malware tries to connect to the host at 128.61.240.66, but it fails
  - Let's make it redirect to our fake C2 server
    - Go to `~/tools/network`
    - Edit iptables_rules to redirect the traffic to 128.61.240.66 to 192.168.133.1 (fake host)
  - Whenever you edit iptables_rules, always run reset.
    - (type "`./reset`" from the `~/tools/network directory`)
  - **IMPORTANT**! If you shut down your project VM, be sure to run reset again the next time you start it up.

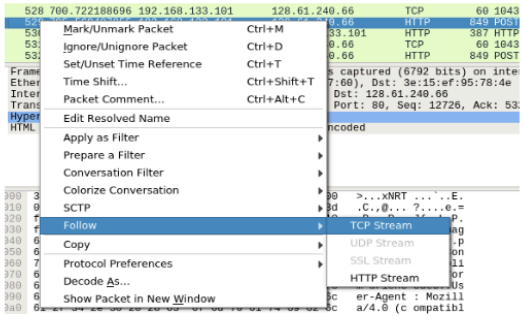
```
(angr) analysis@analysis-VirtualBox:~/tools/network$ cd ~
(angr) analysis@analysis-VirtualBox:~$ cd tools/network/
(angr) analysis@analysis-VirtualBox:~/tools/network$ ls
iptables-rules  reset
(angr) analysis@analysis-VirtualBox:~/tools/network$ vim iptables-rules
```

analysis@analysis-VirtualBox: ~/tools/network

```
1 #!/bin/sh
2 echo "Importing iptables rules"
3
4 # Do not forget to run ./reset after updating this.
5
6 # Uncomment following lines (or add rules) to adjust filtering rules
7 #sudo iptables -t nat -A PREROUTING -p tcp -s 192.168.133.101 -d 143.215.206.97
  --dport 80 -j DNAT --to 192.168.133.1:80
8 sudo iptables -t nat -A PREROUTING -p tcp -s 192.168.133.101 -d 128.61.240.66 --
  dport 80 -j DNAT --to 192.168.133.1:80
```

- Reading C2 Traffic
  - Observing C2 traffic
    - In WireShark, we can notice that now the malware can communicate with our fake C2 server

```
526 700.559453229 192.168.133.101   128.61.240.66     HTTP   849 POST /images/logo/header.php HTTP/1.1  (application/x-www-form-urlencoded)
527 700.560517510 128.61.240.66     192.168.133.101   HTTP   387 HTTP/1.1 200 OK  (text/html)
528 700.722188696 192.168.133.101   128.61.240.66     TCP     60 1043 → 80 [ACK] Seq=12726 Ack=5329 Win=65202 Len=0
529 705.569407055 192.168.133.101   128.61.240.66     HTTP   849 POST /images/logo/header.php HTTP/1.1  (application/x-www-form-urlencoded)
530 705.570186210 128.61.240.66     192.168.133.101   HTTP   387 HTTP/1.1 200 OK  (text/html)
```




```
POST /images/logo/header.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
Host: netscan.gtisc.gatech.edu
Content-Length: 509
Cache-Control: no-cache
Cookie: response=%24updatE%3Aexecute-command

page=ZjllYWNiZWUtNTJjZS00ZmVkLTkzMDAtOTUxOGVhYWQ5MjV1&unm=dA&cnm=R1QtODVGM0E5NDgwNjQ1&query=
V2luZG93cyBYUA==&spec=MzIgQml0&opt=MA&view=W1N5c3RlbSBQcm9jZXNzXQpTeXN0ZW0Kc21zcy5leGUKY3Nyc
3MuZXhlCndpbmxvZ29uLmV4ZQpzZXJ2aWNlcy5leGUKbHNhc3MuZXhlCnN2Y2hvc3QuZXhlCnN2Y
2hvc3QuZXhlCnN2Y2hvc3QuZXhlCnN2Y2hvc3QuZXhlCnNwb29sc3YuZXhlCnN2Y2hvc3QuZXhlCnd1YXVjbHQuZXhlC
mV4cGxvcmVyLmV4ZQphbGcuZXhlCndzY250ZnkuZXhlCmN0Zm1vbi5leGUKd21pcHJ2c2UuZXhlCmlleHBsb3JlLmV4Z
QppZXhwbG9yZS5leGUK&var=TWFjaGluZXM=&val=dW1pa2Y=HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Sat, 17 Feb 2018 04:28:58 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: response=None%3Aexecute-command; expires=Sun, 18-Feb-2018 04:28:58 GMT; Max-
Age=86400; path=/; domain=netscan.gtisc.gatech.edu

4
None
0
```

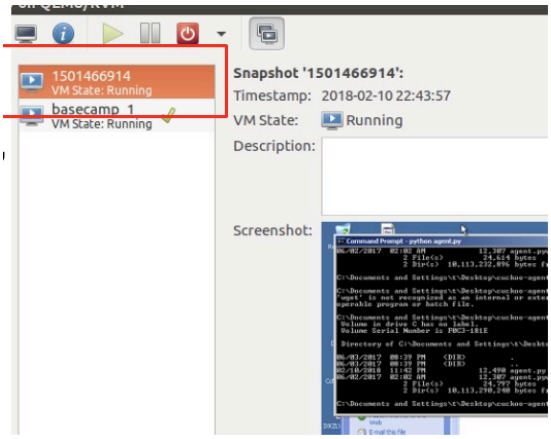
      - But there will not be further execution, because the command is wrong
    - You can see the contents of the traffic by right-clicking on the line, then clicking Follow – TCP Stream

```
1 0.000000000   192.168.133.101   128.61.240.66     TCP    62 1047 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
2 2.927100732   192.168.133.101   128.61.240.66     TCP    62 [TCP Retransmission] 1047 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
3 9.005808865   192.168.133.101   128.61.240.66     TCP    62 [TCP Retransmission] 1047 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
```

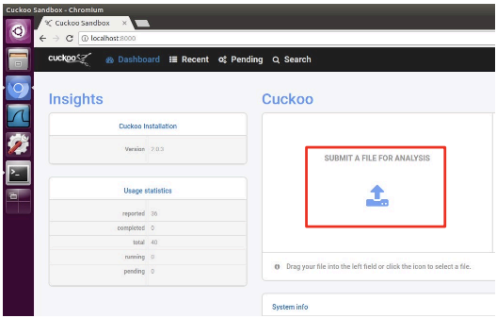
- Cuckoo
  - Let's take a look at cuckoo. Cuckoo is NOT necessarily required to complete this project, but it is a useful tool to help you understand what your malware is doing, and therefore how you might want to modify your score.h file later in the project.
  - **Note**! You can't run the testbed VM and cuckoo simultaneously.
  - **Always turn off** the testbed VM, and follow the steps below to execute Cuckoo
  - Open two terminals.
  - '`$workon cuckoo`' (Set virtualenv as cuckoo for both terminal1 and terminal2)

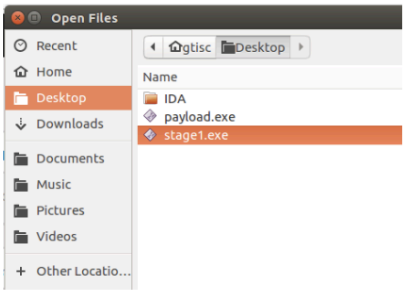
- Open one terminal in debug mode, with command: '`$cuckoo -d`'
- Open other cuckoo terminal for the webserver, with command: '`$cuckoo web`'

- Reference: [Malware Analysis using Cuckoo Sandbox]
- If you get an error when running cuckoo web because port 8000 is already in use, run "`sudo fuser -k 8000/tcp`" and try again.



- The Cuckoo uses a snapshot of the given testbed VM.
- The snapshot is 1501466914
- • DO NOT TOUCH the snapshot!



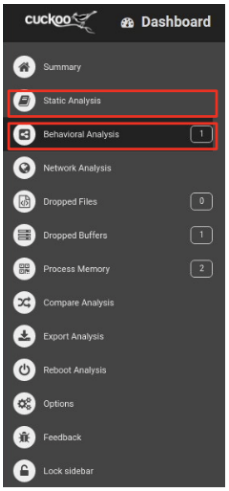


- Upload a file to Cuckoo
  - To open the cuckoo web server, type the following URL into Chromium
    - http://localhost:8000
  - To upload a file, click the red box and choose a file.

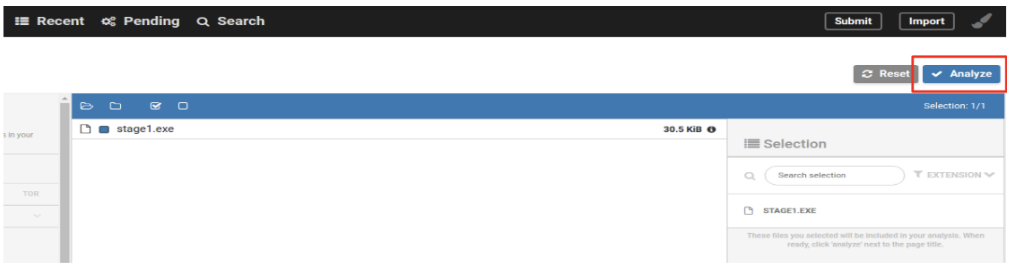
  - Once you click the Analyze button, it will take some time to run the malware.

- Analysis with Cuckoo
    - Once you click the Analyze button, it will take some time to run the malware.

- Figuring Out the List of Commands
    - The malware does not exhibit its behavior because we did not send the correct command through our fake C2 server
    - We will use
        - File/Registry/Process tracing analysis to guess the malware behavior.
        - control-flow graph (CFG) analysis and symbolic execution to figure out the list of the correct commands
    - The purpose of tracing analysis is to draw a big picture of the malware
        - What kinds of System call/API does the malware use?
        - Does the malware create/read/write a file? How about a registry?
    - The purpose of CFG analysis is to find the exact logic that involves the interpretation of the command and the execution of malicious behavior
    - Then, symbolic execution finds the command that drives the malware into that execution path



- Tracing Analysis on Cuckoo
    - On the side bar, there are useful menus for tracing analysis.
        - We are focusing on:
            - Static Analysis
                - API/System Call.



- Behavioral Analysis

        - Trace behaviors in time sequence.
- Static Analysis on Cuckoo
    - Static Analysis
        - Information about the malware.
        - Win32 PE format information
            - Windows binary uses the PE format
            - Complicated structure
            - Sections includes
                - .text
                - Strings, etc.
                - .data
                - .idata
                - .reloc
    - More information: Malware researcher's handbook (demystifying PE file)
    - Interestingly three DLL(Dynamic Link Libraries) files are imported.
    - In WININET.dll, we can see that the malware uses http protocol.
    - In ADVAPI32.dll, we can check if the malware touches registry files
    - In Kernel32.dll, we can check the malware waiting signal, also sleep.

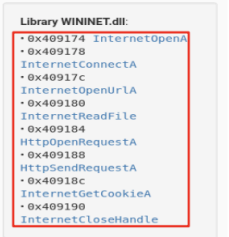
- Behavior Analysis on Cuckoo
    - Tracing a behavior(file/process/thread/registry/network) in time sequence.
    - Useful to figure out cause-and-effect in process/file/network.
    - Malware creates a new file and runs the process, then writes it to memory.

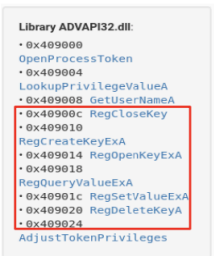
- Cuckoo analysis result
    - Based on our analysis with Cuckoo, we can determine if…
        - The malware uses HTTP protocol to communicate
            - Communicate with whom? C&C?
            - Web server access? For checking if the C2 server is active?
            - Commands through http protocol? Cookies?
        - The malware touches(create/write/read) a file/registry/process
            - This might be a dropper? Or does it download a binary from the C2 server?
            - What is the purpose of creating processes? Modifying the registry?

Sections

Name	Virtual Address
.text	0x00001000
.data	0x00006000
.idata	0x00009000
.reloc	0x0000a000

- Control Flow Graph Analysis
    - Based on the pre-information that we collected from the previous step, we are going to perform CFG analysis & symbolic

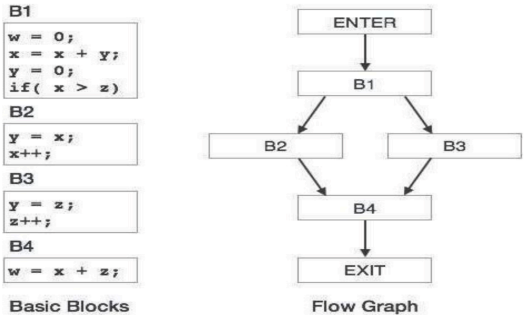




execution analysis

- CFG:
  - graph representation of computation and control flow in the program
  - Nodes are basic blocks
  - Edges represent possible flow of control from the end of one block to the beginning of the other.

- But, in malware analysis, we are analyzing CFG at the instruction level.
- We provide a tool for you that helps to find command interpretation logic and malicious logic
  - We list the functions of system calls the malware uses internally
  - If you provide the score (how malicious it is, or how likely the malicious logic is to use such a function) for the functions, then the tool will find where the malicious logic is, based on its score
    - Example: if you set StrCmpNIA to have a score of 10, then the function that calls StrCmpNIA 5 times within itself will have the score 50.
    - A higher score implies that more functions related to the malicious activity are used within the malware.
  - Your job is to write the score value per each function



- More info: http://www.cs.cornell.edu/courses/cs412/2008sp/lectures/lec24.pdf
- From our network analysis, we know that the malware uses an Internet connection to 128.61.240.66
- From our cuckoo-based analysis, we know that the malware uses the HTTP protocol.
- Moreover, it uses some particular functions to communicate and stay in touch with the command and control server.

- Modify the score values for these particular functions in order to generate a better CFG – for proper analysis.
- Find the file to be edited – score.h.
- Path: /tools/cfg-generation/score.h
- Build control flow graph
  - By executing ./generate.py stage1, the tool gives you the CFG
    - This finds the function with higher score
      - Implies that this calls high score functions on its execution
  - For stage2
    - Use 'stage2' as argument
- **Note**: your graph and its memory addresses will vary from this example
- The function entry is at the address of 405190
  - And, there is a function (marked as sub) of score 12
    - At the address 40525a (marked in red)
    - Use the block_address, not the call sub_address
  - This implies that
    - sub_4050c0 calls some internet related functions.
    - We need to find out what this command is
      - Run from 405190 to 40525a

- Finding Command
  - Finding Commands with Symbolic Execution
    - We want to find a command that drives malware from 405190 to 40525a
      - Let's do symbolic execution to figure that out
  - What is symbolic execution?
    - Rather than executing the program with some input, symbolic execution treats the input data as a symbolic variable, then tries to calculate expressions for the input along the execution.
    - Path explosion
    - Modeling statements and environments
    - Constraint solving
  - Symbolic Execution Engine: Klee, Angr, Mayhem, etc. • Loading a binary into the analysis program
  - • Translating a binary into an intermediate representation (IR). • Translating that IR into a semantic representation
  - • Performing the actual analysis with symbolic execution.

Example1 – Symbolic Execution

Code Example

```c
int main() {
    int i, j;
    printf("Give me two integers\n");
    scanf("%d %d", &i, &j);
    if(i+5<j) {
        if(i%2 == 0) {
            if(j%3 == 0) {
                printf("Correct!\n");
                return 0;
            }
        }
    }
    printf("Incorrect!\n");
    return 1;
}
```

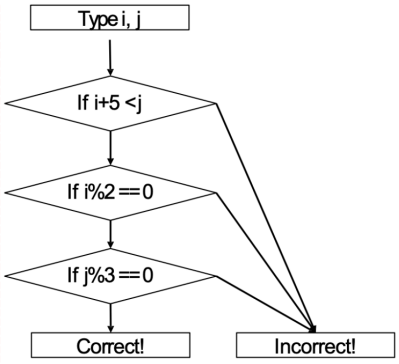
Expressions

i+5 < j

i+5 < j; i%2==0

i+5 < j; i%2==0; j%3 == 0

Solve the expression
i = 2
j > 7, but multiple of 3 so
j=9



i=2, j=9 will lead the program to print “Correct!”

- In this example, ONLY i=2, j=9 conditions will lead the program to print “Correct!”
- Symbolic execution is available to solve the expression in order to reach a target, in this case ”Correct”.
- Let’s apply it into Malware Command & Control logic. A C&C bot(malware) is expecting inputs(solve the expressions) to trigger behaviors(targets).

Example2 – Symbolic Execution

Code Example

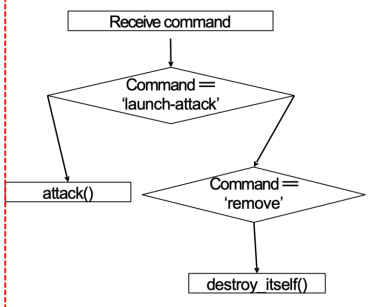
```c
int main() {
    char command[512];
    // receive the command
    recv(socket, command, 512, 0);

    // compare the input with 'launch-attack'
    if(strcmp(command, "launch-attack") == 0) {
        attack();
    }
    else if (strcmp(command, "remove") == 0) {
        // when the command is 'remove'
        destroy_itself();
    }
}
```

Expressions

Command == ‘launch-attack’

Command == ‘remove’



This executes attack() on command ‘launch-attack’, and destroy_itself() on ‘remove’ command

- In this example, ONLY ‘launch-attack’ and ‘remove’ commands(inputs) triggers attack() and destroy_itself().

- Symbolic execution is able to find ”launch-attack” as an input to trigger attack(), which is a malicious behavior.
- Plus, ”remove” will lead to destroy_itself(), which is another behavior.
- Our job in this project with Symbolic execution is to find inputs, and then feed the inputs to trigger behaviors.

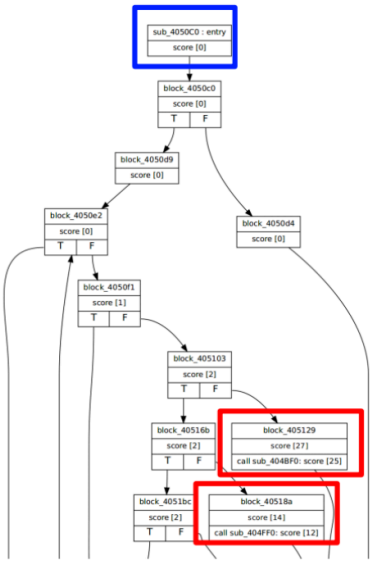
- Finding Commands with Angr
  - We prepared a symbolic executor and a solver for you
    - Your job is to find the starting point of the function which interprets the command, and find the end point where malware actually executes some function that does malicious operations
      - Use a Control-flow Graph (CFG) analysis tool!
    - The symbolic executor is called angr (http://angr.io/index.html)
  - We prepared a symbolic executor and a solver for you.
  - How do you run it?
    - Go to `~/tools/sym-exec`
    - Run it like

```
python ./sym_exec.py [program_path] [start_address] [end_address]
```

  - Replace the (above) start and end addresses from your CFG graph.

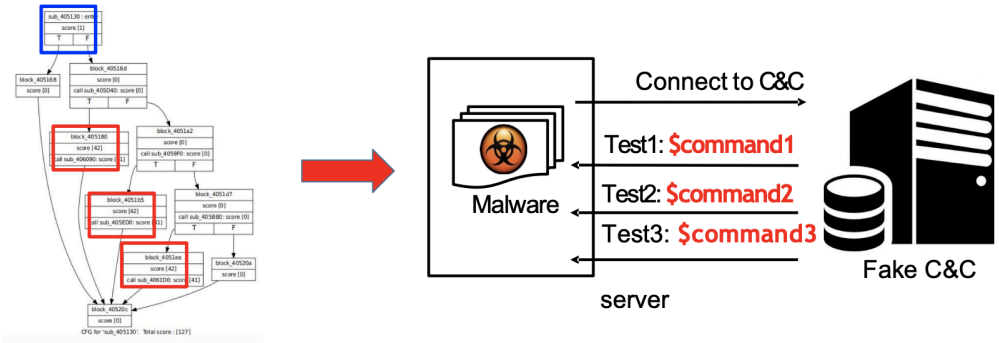
```
python ./sym_exec.py ~/shared/stage1.exe 4050c0 40518a
```

  - The command will be printed at the end (if found)



- Reconstructing C2 server
  - After CFG analysis + symbolic execution, reconstruct the C2 server
  - The tool for reconstructing the C2 server is already on the VM
  - It runs nginx and php script
    - This will look like ~/tools/c2-command/stage*-command.txt

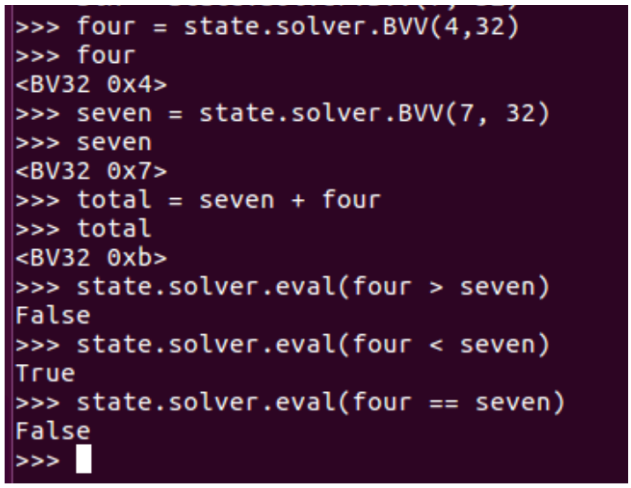
- Your job is to add your commands to the relevant *.txt file
    - The command that leads the execution from 405190 to 40525a is "`$insert`" (note: the name of the command you see may vary)
    - Then, type "`$insert`" and save the file.
    - **Important**: be sure to put the '$' character before your commands, even if stage* - command.txt says that it's optional
    - The order of commands in the file does not matter – they'll run in a random order
        - **Note**: This means that if you want to run only a particular command, you'll need to remove, or comment out the other commands in your file



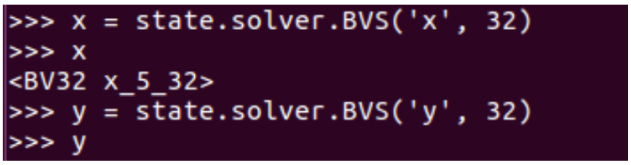
- angr
    - *SimState*
        - angr – SimState
        - While angr perform symbolic execution, it stores the current state of the program in the SimState objects.
        - SimState is a structure that contains the program's memory, register and other information.
        - SimState provides interaction with memory and registers. For example, state.regs offers read, write accesses with the name of each registers such as state.regs.eip, state.regs.rbx, state.regs.ebx, state.regs.ebh
        - Creating an empty 64 bit SimState

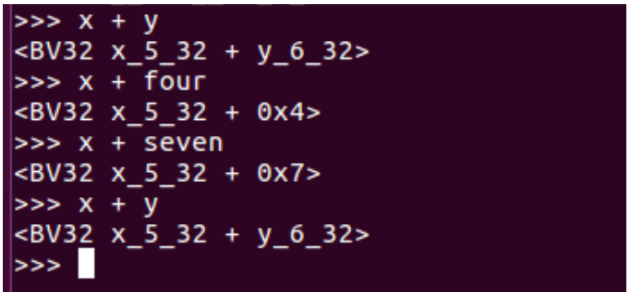
```
(angr) analysis@analysis-VirtualBox:~/report$ python
Python 2.7.12 (default, Jul  1 2016, 15:12:24)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import angr
WARNING | 2019-02-22 16:23:44,860 | angr.analyses.disassembly_utils | Your
>>> state = angr.SimState(arch='AMD64')
WARNING | 2019-02-22 16:24:02,100 | angr.sim_state | SimState defaulting to
>>> state
<SimState @ <BV64 reg_rip_0_64{UNINITIALIZED}>>
>>>
```

- *Bitvectors*
  - Since, we are dealing with binary files, we don't deal with regular integers.
  - In binary program, everything becomes bits and sequence of bits.
  - A bitvector is a sequence of bits used to perform integer arithmetic for symbolic execution.
  - Creating some 32 bit bitvector values
  - state.solver.BVV(4,32) will create 32 bit length bitvector with value 4
  - We can perform arithmetic operations or comparisons using the bitvectors



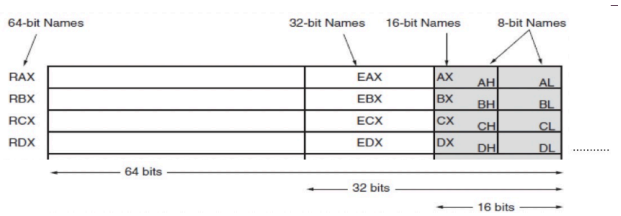
- *Symbolic Bitvectors*
  - state.solver.BVS('x', 32) will create a symbolic variable named x with 32 bit length
  - Angr allows us to perform arithmetic operation or comparisons using them.





- *Registers*
  - State provides access to the registers through state.regs.register_name where register_name could be rcx, ecx, cx, ch and cl. Same applies to the other registers.
  - Look at the types of registers -- they are bit vectors
  - Look at the length of registers examined below.
    - They are all symbolic bitvector because they are not initizlized yet.
  - For cl, ch, cx and ecx they are all part of rcx.
  - You can compare the length and the location of cl, ch, cx, ecx and rcx in angr with the actual architecture depicted below.

```
>>> state.regs.rcx
<BV64 reg_rcx_4_64{UNINITIALIZED}>
>>> state.regs.ecx
<BV32 Reverse(Reverse(reg_rcx_4_64)[63:32])>
>>> state.regs.cx
<BV16 Reverse(Reverse(reg_rcx_4_64)[63:48])>
>>> state.regs.ch
<BV8 reg_rcx_4_64[15:8]>
>>> state.regs.cl
<BV8 reg_rcx_4_64[7:0]>
>>>
```



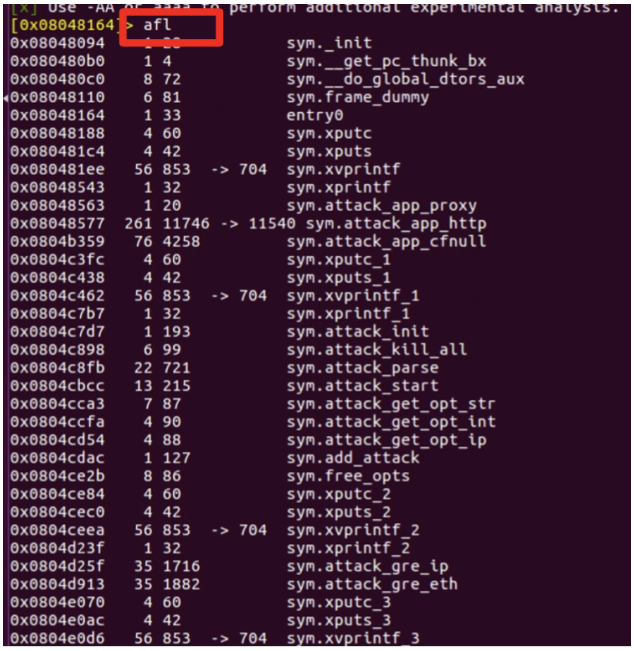
- *Constraints*
  - In a CFG, a line like if (x > 10) creates a branch. Please look at the Symbolic Execution Concepts tutorial.
  - Assuming x is a symbolic variable, this will create a <Bool x_5_32 > 4> when the True branch is taken for the successor state
  - For the false branch,negation of a <Boolx_5_32>4> will be created.
  - Adding a constraint to a SimState
    - Cl register equals to 11
    - state.add_constraints(state.regs.cl == 11)
    - state.add_constraints(state.regs.cl == state.solver.BVV(0xb, 8) since state.solver.BVV(0xb, 8) equals to 11
    - You can see their effect is the same for SimState in the example below.

```
>>> state.add_constraints(state.regs.cl == state.solver.BVV(0xb, 8))
>>> state.se.constraints
[<Bool reg_rcx_7_8 == 11>]
>>> state.add_constraints(state.regs.cl == 11)
>>> state.se.constraints
[<Bool reg_rcx_7_8 == 11>]
>>> state.add_constraints(state.regs.cl >= 13)
>>> state.se.constraints
[<Bool reg_rcx_7_8 == 11>, <Bool reg_rcx_7_8 >= 13>]
>>>
```

- *Radare2*
  - Launch radare2 with $ r2 ~/shared/payload.exe
  - Then type aaa which will analyze all (functions + bbs)

```
(angr) analysis@analysis-VirtualBox:~$ r2 ~/shared/payload
Warning: Cannot initialize dynamic strings
 -- This is just an existentialist experiment.
[0x08048164]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
[x] Type matching analysis for all functions (afta)
[x] Use -AA or aaaa to perform additional experimental analysis.
```

- `afl` list all functions



- `afl` lists all the functions which are hard to analyze.
- `afl~name` grep the list of functions with given name
- `afl~attack` will list all the functions having attack

- You can use linux commands while inside the r2 console such as grep.
- On the right side, you can see all the functions having the attack vector (afl~send)
- Using those api calls, this linux malware performs DDoS attacks based on the commands they receive from C&C server.
- The example below shows how to find all the attack vectors calling `sym.send/sym.sendto`
- Now, we have to iterate all the attack functions on the right. For example, the example below shows three attack functions, and only one of them is called. Our focus is the call sym.attack_????? functions.

- Let's analyze the example below.
- `axt sym.attack_app_http` has only one reference which is a push instruction. This is not the attack function we are interested in.
- `axt sym_attack_app_cfnull` has no reference at all. This is not the attack function we need to explore.
- `axt sym_attack_`???? Is one of the functions listed on the right example, and have call `sym.attack_`????? Instruction. That is the function we need to explore more to determine the target address for the symbolic execution.
- **You need to find 2 attack functions**.

    - After finding the attack function, we can determine the target address.
        - First, step into the function using `s sym.attack_????.`
        - Second, `pdf | grep sym.send` or `pdf | grep sym.sendto` to determine the instruction address
        - Third, `s address_for_call_sym.send(to)` to point to the instruction which is call sym.send or sym.sendto
        - Lastly, print 2 instructions starting with the call sym.send/sym.sendto instruction
    - The address of the instruction which is the successor of call sym.send(to) is the target address for the symbolic execution.

    - For more information :
        - https://github.com/radare/radare2
        - https://www.radare.org/get/THC2018.pdf
- Other Tools:
    - You don't have to use Radare2.
    - Here some of the tools you may want to use
        - objdump
        - IDA-Pro (Disassembly tool with GUI) (Free version)
            - https://www.hex-rays.com/products/ida/support/download_freeware.shtml
        - Cutter (GUI for the radare2)
            - https://www.radare.org/cutter/
            - https://github.com/radareorg/cutter

After stage1.exe

- If you find all of the commands for stage1.exe malware, the malware will download stage2.exe by updating itself.
- Now you've found the commands from running sym-exec.py
- Add those commands to stage1-commands.txt. Remember to put `$<command>`.

- Start up the windows VM again, then copy stage1.exe to the desktop. Then double click on it and continue.
- Note if stage1 fails to download stage2, your firewall might be blocking it
    - This is actual malware so some IDS have signatures that match it.

- For stage2.exe, please follow the same steps in the tutorial
    - Check its network access with Wireshark
    - Redirect network traffic to if required (if the connection fails)
    - Try to identify malicious functions by editing score.h and using the cfg-generation tool
    - Discover the list of commands using the symbolic execution tool
    - Fill the commands in ~/tools/c2-command/stage2-command.txt
    - Run it as mentioned before.

Linux Malware Analysis

- Stage2.exe will download stage3 malware, which is payload.exe.
    - This is **Linux Malware**.
- We need to handle the linux malware differently unlike windows malware, and will use different tools and methods to analyze this malware

Linux Malware Tools

- First copy the linux malware into a shared folder. The tools which you will use are installed inside the Linux host.
- ~/tools/sym-exec/linux_sym_exe.py
    - for linux malware symbolic execution
    - python linux_sym_exec.py path_to_linux_mw start target
    - To make it work, you need to modify two **linux_sym_exec.py** functions
        - targs_len_before and opts_len_before
- ~/tools/dynamicanalysis/
    - instrace.linux.log : the dynamic instruction trace for the linux malware
    - detect_loop.py : you have to modify this file to find the loop in the given trace
    - Usage: python detect_loop.py
- Run ‘python linux_sym_exec.py path_to_linux start target’.
- It won’t be able to find any input because of path explosion. You need to add constraints to make symbolic execution targeted
- Follow the steps in assignment- questionnaire.txt and find the inputs.
- Analyze the dynamic instruction trace and locate the C&C communication

Android Malware Analysis

- Manifest Analysis
  - Identifying suspicious components
- Static Analysis
  - Search for C&C commands and trigger conditions
  - Vet the app for any anti-analysis techniques that need to be removed.
- Dynamic analysis
  - Leverage the information found via static analysis to trigger the malicious behavior.

Manifest Analysis

- Identify suspicious components
  - Broadcast receivers registering for suspicious actions.
  - Background services
- Narrow the scope of analysis
  - Malicious apps are repackaged in benign apps with thousands of classes.

```
<receiver android:name="com.android.SMSReceiver">
    <intent-filter android:priority="10000">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
        <action android:name="android.provider.Telephony.SMS_SENT" />
    </intent-filter>
</receiver>
```

Broadcast receiver from CoinPirate's malware family.

Static Analysis

- Search for C&C commands and trigger conditions

```
private int checkMessage(String body) {
    String message1 = getResources().getString(C0197R.string.message_1);
    String messsage2 = getResources().getString(C0197R.string.message_2);
    String message3 = getResources().getString(C0197R.string.message_3);
    if (body.equals(message1)) {
        return 1;
    }
    if (body.equals(messsage2)) {
        return 2;
    }
    if (body.equals(message3)) {
        return 3;
    }
    return 0;
}
```

```
<string name="message_1">hey whats up?</string>
<string name="message_2">Is this Tim?</string>
<string name="message_3">Oh, sorry wrong number! =(</string>
```

- Identifying Anti-analysis techniques

```
public static boolean checkID(Context context) {
    TelephonyManager telephonyManager = (TelephonyManager) context.getApplicationContext().getSystemService("phone");
    if (telephonyManager == null || telephonyManager.getDeviceId().equals(context.getResources().getString(C0197R.string.avalue))) {
        return false;
    }
    return true;
}
```

Scenario

- Analyzing Android Malware
  - You have received a malware sample **sms.apk**.
  - You need to identify communication with the C&C server
  - Identify anti-analysis techniques being used by the app.
  - Identify commands that trigger any malicious behavior.

Project Structure

- Android emulator
  - An emulator for Android 4.4 is pre-installed
    - Run ‘run-emulator’
      - This will start the Android emulator (this takes along time, especially the first time you start it)
  - Jadx
    - Disassembles apk files into Java source code.
- Apktool
  - Disassembles apk file into Smali.
  - Rebuilds apk files.
- Write-up (~/Android/MaliciousMessenger/writeup.pdf)
  - Detailed guide on how to complete the Android section of the lab.
- Android App
  - ~/Android/MaliciousMessenger/tutorialApps
    - Emu-check.apk
      - A tutorial example (Shown as ‘My application’ in the emulator)
  - CoinPirate.apk
    - Another tutorial example
- ~/Android/MaliciousMessenger/sms.apk
  - Target app to analyze to answer the questionnaire
- **READ ~/Android/MaliciousMessenger/writeup.pdf**

Android Cheatsheet

Android Cheatsheet (thanks to Joey Allen)

•**Start Emulator**`~/bin/run-emulator`

•**Add Contact**
The sleeps are needed to allow a slow emulator time to process
```
adb shell "am start -a android.intent.action.INSERT -t vnd.android.cursor.dir/contact -e name 'GatechID'" sleep 1 adb shell input keyevent 4 sleep 1
adb shell input keyevent 4
```

•**Android Log** `adb logcat`Reference: abd-logcat

•**Filtered log**
The adb tool has no way to filter by app, fortunately there's a script that'll do just that. colors!

Get the script and make it executable (review it before running something off the internet :|)
```
wget -O ~/bin/pidcat.py https://raw.githubusercontent.com/JakeWharton/pidcat/master/pidcat.py chmod +x ~/bin/pidcat.py
```
Monitor the malware log`~/bin/pidcat.py com.smsmessenger`Reference: pidcat

•**Decompile APK**
Note: Omitting the -r,--no-res option allows it to decode the resources as well as the smali code.
```
apktool decode ~/Android/MaliciousMessenger/sms.apk --output ~/Android/MaliciousMessenger/sms
```

•**Build Modified APK**
```
apktool build ~/Android/MaliciousMessenger/sms --output ~/Android/MaliciousMessenger/sms_modded.apk
```

•**Sign Modified APK**`~/bin/signer.py ~/Android/MaliciousMessenger/sms_modded.apk`

•**Uninstall APK**`adb uninstall com.smsmessenger`

•**Install Modified APK**`adb install ~/Android/MaliciousMessenger/sms_modded.apk`

•**Launch the app**
The app will not be active until you run it at least once after re-installation 😡, spent a bunch of time banging my head against the wall until i figured this one out.`adb shell monkey -p com.smsmessenger -c android.intent.category.LAUNCHER 1`

•**Send an SMS**
Use single quotes or you'll need to escape the message contents.
Note: I didn't test with emojis!
```
adb emu sms send 8675309 '♫ Jenny Ive called your number...♫'
```

•**Enable Ubuntu Workspaces**
This is a personal preference but it makes it easier to separate the work into different contexts
```
gsettings set org.compiz.core:/org/compiz/profiles/unity-lowgfx/plugins/core/ hsize 2 gsettings set org.compiz.core:/org/compiz/profiles/unity-lowgfx/plugins/core/ vsize 2
```
Or if you prefer using the mouse change the settings: Start -> Appearance -> Behavior -> Enable Workspaces

Tips for assignment-questionnaire.txt

- Please use the **latest version** of VirtualBox when you import the VM. Please do not modify anything related to network settings in the VM.
- Domain name
    - On the questionnaire sheet, there are entries for writing domain names. Please follow the following rules on getting answers for those questions.
    - You should write FQDN, which means, if the full domain name is canof.gtisc.gatech.edu then write canof.gtisc.gatech.edu, not just gatech.edu or gtisc.gatech.edu
    - For the others (connections check, DDoS, sending info, etc.), you should get the exact domain name that the malware uses. For example, the IP address 130.207.188.35 belongs to both coe.gatech.edu and web-plesk5.gatech.edu.
    - Because there are multiple mappings, you cannot be sure about which domain that the malware used by just using nslookup. In this case, please go through the other way of getting domain names from DNS Packets in Wireshark.
    - All Domains should be based on Wireshark DNS packets
        - e.g., get it from a DNS query packet or redirect HTTP traffic into a local VM and examine the Host header.
    - If you get see the log in the Wireshark, You will find DNS query(Standard query) and DNS response(Standard query response)
    - In Domain Name System section, there is Query section, like below
    - Queries:
        - x.y.z: type A, class IN.
    - Answers:
        - x.y.z: type CNAME, class IN, cname a.b.c
    - You should use x.y.z
- URL
    - For all URLs, you do not have to specify the protocol (http:// or https://, etc.).
    - However, if HTTP traffic is like the following:
        - POST /a/b/c/d?asdf=1234 HTTP/1.1 Host: www.zzz.com
    - Then please write this as
        - www.zzz.com/a/b/c/d?asdf=1234
- Writing commands in *.txt files under c2-command directory
    - There are pre-installed PHP scripts in the VM locally that read the *.txt file for each stage,
        - These scripts send the command to the malware after reading them from the TXT files.
        - One caveat of these scripts is that they are written to send the commands in random order (i.e., if there are commands a, b, c, then the script will randomly choose one command and send it to the malware).
        - So if you want to test ONE command at a time, then please write only that command in the TXT file.
            - Ex. If you just want to run the command $uninstall, then please write only that command in stage1- command.txt.

- linux_sym_exec and detect_loop for linux malware
    - You could use free IDA-Pro, objdump or radare2 for this task to find out called attack functions, and the target addresses.
    - Look for some angr examples on the github, which adds constraints to the state.
    - For the loop detection, focus on function sequence that called repetitive
- Correct command but malware is not working?
    - Note that some commands for stage 2 are different per each student, by having 4 digit hexadecimal numbers at the end of the command.
        - Ex. a command for stage 2 is formatted like $COMMANDa1b4
        - (NOTE: three commands in stage 2 have the 4 digit hexadecimal tail.
        - All commands in stage 3 have the 4 digit hexadecimal tail on the command.
    - However, there could be a case that only gets the front part of the command like
        - $COMMAND
        - If the endpoint address of symbolic execution is not correctly set. In such a case, please set the correct end point that you can get the entire command.
- Cuckoo
    - In the VM, we provide cuckoo, which is a dynamic malware analysis framework.
        - It is very convenient and easy to use.
        - While you are running cuckoo, you might meet some warnings and errors "critical time blah blah~" and "YARA signature.... blah blah". **Please ignore them**.
        - Because you are executing malware in the QEMU Windows VM, the framework needs to set a time.
            - Cuckoo will check if the malware is terminated or not.
            - However, the three malware you will meet are never going to be terminated (intentionally, modified by me for educational purposes.)
            - So, please ignore "critical time blah blah~, terminating.
        - In our case, the malware is never going to unfold even though you give an infinite time to be executing the malware unless you feed the right inputs (The malware expects C2 commands.)
    - IPtable Setting
        - If you check /home/analysis/.cuckoo/conf/kvm.conf, you will find how we set the QEMU windows host VM.
        - You will find the IP of the host VM is "192.168.133.101".
        - If you want to see network behaviors in Cuckoo, you want to forward the IP in /home/analysis/tools/network/iptables- rules.
        - For example, open iptables-rules, you want to add

```
sudo iptables -t nat -A PREROUTING -p tcp -s 192.168.133.101 -d
[DEST-IP] --dport 80 -j DNAT --to 192.168.133.1:80
```

Miscellaneous VM Performance Tips

Part 1 : Windows Malware / Generic VM Issues

- **Try lowering your screen resolution**
- **Save often!**
- Avoid using a resource heavy IDE like IntelliJ, Eclipse etc. Lightweight alternatives include gedit, vim, emacs, Sublime Text, Visual Studio Code, nano, etc
- Most importantly, do / run only 1 task at a time. That means:
    - Run the Windows VM only when:
        - Sending commands to malware
        - Analyzing network traffic via Wireshark
        - Once done with those tasks, turn off the Windows VM.
    - Avoid running the windows VM when:
        - Running cuckoo analysis
        - Generating CFGs
        - Running Symbolic Execution - This is quite resource intensive, avoid doing other stuff to get this done quickly. (TIP: If this seems to be taking infinite memory/time, you're mostly trying to reach an unreachable / invalid address! check your addresses!)
    - Try running the VM at a lower resolution (recommend at-least 1280x800, for legibility) - If you have a very high resolution on your host machine. You can do this in 2 ways:
        - VirtualBox Menu - View > Virtual Screen 1 > Resize to a x b
        - Ubuntu Menu - Type "Displays" > Change it there
    - Restart after a task / stage. This is mostly a last resort but restarting the VM after finishing a task/stage made everything feel really smooth, instead of trying to free memory etc. Just be sure to run ./reset in ~/tools/networks after each VM restart!

Part 2: Android

- Some of the above stuff applies here (VM Settings, resolution, etc).
- Restarting after working on Part 1, helps a lot.
- If you still really feel your android emulator is slow you can add the following flags to the emulator command flags in ~/bin/run-emulator

```
-memory 2048 -gpu swiftshader
```

- You can experiment with RAM allocation and CPU usage based on your machine – but keep in mind that the project VM has only been tested at 4 GB and with 2 or 3 CPUs.

Extra Tips

- Once you successfully complete the stage1 part, and the stage2 file is downloaded on the Windows Vm, you can move it to the shared folder, for better handling. Verify the file type as mentioned in the write-up before, and handle it in the same manner as stage1.
- For stage2, do not forget to update the 'iptables_rules' files, and run './reset' after it.
- General tips – If your device frequently lags, or takes a long time to execute, reboot your device.
    - Fewer resource allocation could result in some issues, you could try to reinstall the VM image (deleting the previously stored state), and even Virtual-box as a last resort.
- Do NOT change the base snapshots.
- Ensure you have set up no firewalls.
- Some particular MAC users might be unable to unzip the project3.zip to obtain the .ova file, in which case login into DropBox as a user, instead of a guest. Verify the file properties afterwards.
- For all users – a partial file download will result into errors. Verify once before execution.
- Moreover, if you have a problem with your current device, (it's too old or cannot allocate proper resources for a smoother experience), please contact us beforehand so we can arrange for an alternative, we cannot provide one in the last few days.
- One alternative to try on your own: Amazon EC2. Set-up an OVA file on an EC2 instance, initially converting the file to a format supported by EC2, i.e. VMDK, VHD, or RAW formats. They may differ as per the instance type chosen by you. Next Step would be to upload the converted file to a S3 bucket, create a containers.json file for handling, start the instance and import it manually. Virtualbox or other operating software would need to be installed as well, and then import it and execute.

Submission

Required files

- Zip the following files and upload report.zip to Canvas
    - Running ~/archive.sh will automatically zip all of the files
        - ~/report/assignment-questionnaire.txt
        - Stage1.exe, stage2.exe, payload.exe (linux malware)
        - ~/tools/network/iptables_rules
        - ~/tools/cfg-generation/score.h
- Running ~/archive.sh will create report.zip automatically.
    - Please check the content of your zip file before submitting it to Canvas
- Submit only '**assignment-questionnaire.txt**' to **Gradescope**, the **report.zip** to **Canvas** (under Project3 Assignment).
  If you did not submit report.zip on time, a **5-point deduction** will be applied to your total score.

Questionnaire

- To get credit for the project, you have to answer the questionnaire, found at **on canvas**
- Please **strictly follow the format** or the example answer for each question in assignment-questionnaire.txt. TAs use an **autograder** for your submission.
- Windows Part
    - Read **assignment-questionnaire.txt**
    - Carefully read the questions, and answer them in **assignment-questionnaire.txt**
    - For each stage, there are 4-6 questions regarding the behavior of the malware.
- Android Part
    - **READ ~/Android/MaliciousMessenger/writeup.pdf**
    - Carefully read the writeup, answer in **assignment-questionnaire.txt**
    - Make sure you overwrite ANSWER_HERE

Rubric

- The value for each max score is within its particular section
    - Windows has 110 possible points
    - Android has 100.
    - As each section is worth an equal amount of your overall P2 grade, we normalized the Windows score by dividing by 1.1 (and rounded up), then averaged it with the Android score to get your final grade. So effectively, each point in the table above is worth half a point of your final project grade (slightly less for Windows).
- If the Partial Credit column is blank, there is no partial credit for the question. “Ratio” refers to Levenshtein ratio, it’s a metric of similarity between strings

Question Number	Subquestion	Max Score*	Partial Credit**	Penalty
1.1 (Stage 1)	IP	1		
	URL	1	0.5 pts if ration > 0.9	
1.2	Process Name	2	1 pt if ratio > 0.8	
1.3	Command 1	4		
	Command 2	4		
	Command 3	4		
1.4	Command	2	1 pt if correct answer in student answer/vice versa	
1.5	IP	1		
	Domain	1	0.5 pts if ratio > 0.9	
2.1 (Stage 2)	IP	2		
	URL	2	1 pt if ratio > 0.9	
2.2	Process Name	2	1 pt if ratio >0.8	
2.3	Command 1	3		
	Command 2	3		
	Command 3	3		
	Command 4	3		
2.4	Command	2	1 pt if correct answer in student answer/vice versa	
2.5	Path	6	5 pts if student answer in correct answer, 4 pts if ratio > 0.8	
2.6	Command	4		
3.1 (Linux Malware)	Function Name 1	4		
	Function Name 2	4		
3.2	Constraint	4		
3.3	Constraint	4		
3.4	Command/Target 1	10		
	Command/Target 2	10		
3.5	Function Addresses	24	No partial credit per address, you get a fraction of the max point per correct address	If you provide more addresses than the correct answer, a -4 pt penalty will be applied
4.5.1 (Android)	Question 0	5		
4.5.2	Question 1	10		
4.5.3	Question 2	20		
4.5.4	Question 3	20		
4.6.1	Question 4	15		
4.6.2	Question 5	30		