**Thilina Ratnayake**

**COMP 8505**

**A3**

Table of Contents

Table of Contents

[Introduction 3](#_Toc433013844)

[Features: 3](#_Toc433013845)

[Practical Application 3](#_Toc433013846)

[Usage 4](#_Toc433013847)

[Requirements: 4](#_Toc433013848)

[Sending & Receiving Data 4](#_Toc433013849)

[Results 5](#_Toc433013850)

[Diagrams 5](#_Toc433013851)

[Code Listings 6](#_Toc433013852)

[Psuedocode 6](#_Toc433013853)

[Blackhat.py: 6](#_Toc433013854)

[Client.py: 6](#_Toc433013855)

[Tests 7](#_Toc433013856)

[Figures 8](#_Toc433013857)

# Introduction

Backsniffer is a covert communication suite for that allows communication between an attacker and a backdoor application on a target’s compromised machine.

Backsniffer contains two modules:

1. Blackhat.py – This is the module that sends commands to the target and waits for replies.
2. Client.py – The actual “Back-Door” that can be run on the client machine.

## Features:

Some features of the Backsniffer suite include:

* **Firewall Evasion:**  
  Commands are able to get through to the target machine even with a running firewall due to the use of raw-sockets to sniff for packets.
* **Process Masking:**The back-door module running on the client’s machine can camouflage itself by changing the name of it’s process. This allows it to remain invisible through usual detection methods such as running ‘ps aux’.
* **Authentication**By checking for a pre-determined TTL and destination port, there are two layers of authentication to ensure that the backdoor only picks up messages that are meant for it.
* **AES 256 Bit Encryption**All messages sent between the client and backdoor are encrypted using AES 256 bit encryption to mitigate any chance of easy discovery via packet captures.

## Practical Application

A typical scenario for this application would be uploading and executing the client.py application on a target machine via an entry vector of ones choice (social-engineering etc)

Once run, the backdoor will disguise its-self with a process name as set by the attacker so as to evade detection on any process lists. After it has masked its process name, it will listen on raw sockets for packets from the attackers, which match a specific signature. The signature is a combination of 2 facts, the IP header’s TTL and destination port it is being received on. As mentioned above, this ensures that messages get through any personal firewall that is running on the compromised system.

Packets then get decrypted using a pre-shared decryption key producing the command which is executed and sent back using the same encryption sequence. This allows an attacker to virtually have a remote shell on a compromised system.

# Usage

## Requirements:

1. Backsniffer requires that the following python libraries be installed
   1. PyCrypto
   2. Setproctitle
   3. Scapy
2. This can all be installed by running the shell script:
   1. sh startup.sh

## Sending & Receiving Data

1. Starting the backdoor.

On the target machine, enter command: **python client.py 80 71 012345689abcdef abcdefghijklmnop [KWorker2:0]**

The first two arguments that we are listening for packets that have the characteristics: (1) incoming to port 80 and (2) have a TTL of 71. This is the criteria to specify that the packets are from the attacker.

Secondly, we then enter in our pre-shared encryption key and initialization vectors. These two parameters allows us to decrypt and encrypt the messages between attacker and victim.

We instruct the backdoor to start-up with the process name “[KWorker2:0]”. This is because on our test systems, there are multiple kworker\*:\* processes running at any given time, and choosing the name specified will be able to easily mask the process. The first two letters have been highlighted to be able to recognize the process in order to kill it later.

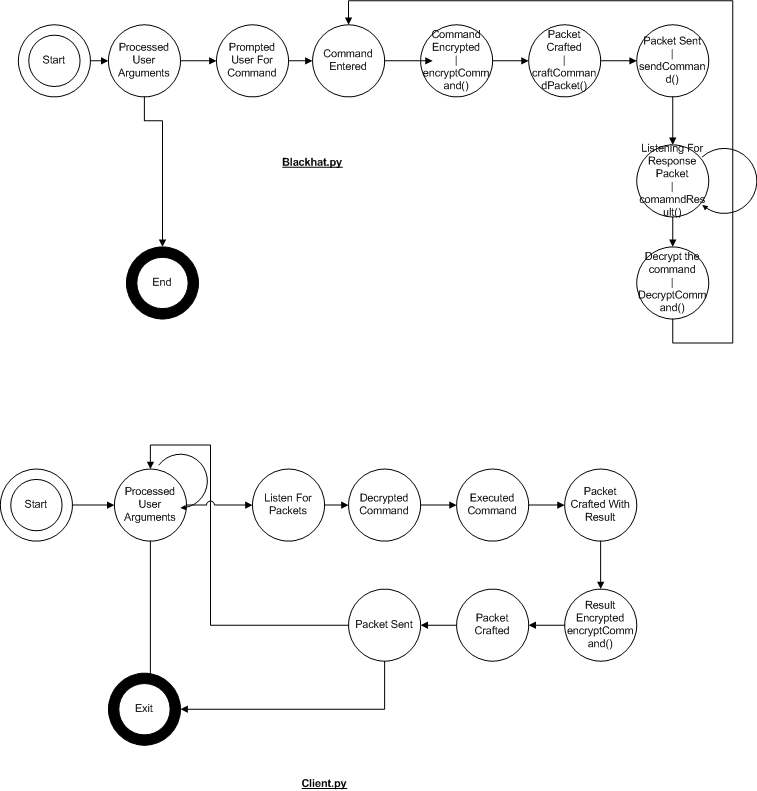
1. Starting the attacker’s shell.  
   On the attackers machine, enter command: **python blackhat.py 192.168.0.3 500 80 71 0123456789abcdef abcdefghijklmnop**After the connection info has been entered, the attacker can simply begin entering in commands as if he or she were utilizing a shell on the victims machine.

# Results

After running tests and experiments, the results show that Backsniffer is able to:

* Send encrypted messages between the two systems.
* Execute commands on the client machine and send the output back.
* Evade a firewall that is dropping all packets.

# Diagrams



# Code Listings

The Backsniffer application is split up into two modules:

1. Blackhat.py – This is the module that sends commands to the target and waits for replies.
2. Client.py – The actual “Back-Door” that can be run on the client machine.

# Psuedocode

### Blackhat.py:

1. Take command line arguments
2. Prompt use for message
3. Encrypt message
4. Craft packet to with source port, destination port, and ttl specified in step 1
5. Send Packet
6. Listen for a response from the client
7. Decrypt the received packet
8. Display the result

### Client.py:

1. Take command line arguments
2. Set process title to what is specified in 1
3. Listen on the port specified in 1
4. For each packet received on that packet, check it against the signature specified in 1 (ttl and src port)
   1. If a packet matches the signatures
      1. Decrypt the command from packet
      2. Run command
      3. Get output
      4. Craft a packet destined for the source that the incoming packet came from.
      5. Encrypt result
      6. Put result into packet
      7. Send packet
5. Listen for new packets

# Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Test / Resource(s) | Command | Expected | Result | Figs |
| 1 | Process name is masked. – Client.py | **python client.py 80 71 012345689abcdef abcdefghijklmnop [KWorker2:0]** | Ps aux | grep “client.py” or “python.py” should not show any processes.  Ps aux | grep “KWorker” should. | PASS | 1.1-1.3 |
| 2 | Command’s are received sent from the attacker to the client on the specified ports – Blackhat.py & Client.py | **python blackhat.py 192.168.0.3 500 80 71 0123456789abcdef abcdefghijklmnop** | Packet should be going from source 500 to port 80 | PASS | 2.1 |
| 3 | Commands sent/received from the attacker are encrypted. Client.py | Same as 2 | Data in packet payload should not be readable. | PASS | 3.1,3.2,3.3 |
| 4 | The right command is received and executed | Same as 2. | The outputs of the directory should be displayed. | PASS | 4.1 |
| 5 | The results are sent back and are encrypted | Same as 2 | The outputs of the response are encrypted | PASS | 4.5 |
| 6 | Commands that produce no output send back the placeholder text | Client: **python client.py 80 71 012345689abcdef abcdefghijklmnop [KWorker2:0]**  Server: **python blackhat.py 192.168.0.3 500 80 71 0123456789abcdef abcdefghijklmno**p | The commands executed should throw an error as they are decrypted/encrypted with the wrong key. | PASS | 6.1, 6.2 |

# Figures

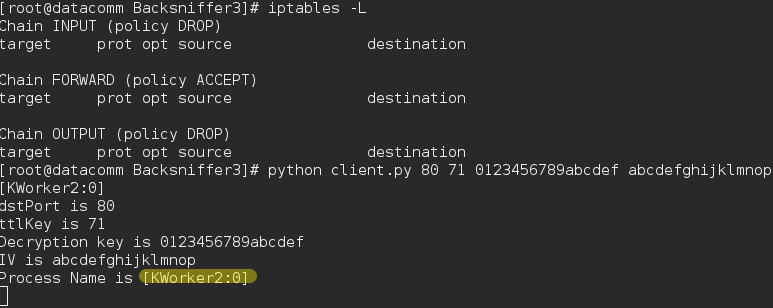


Fig 1.1 – Client output after backdoor has been started.

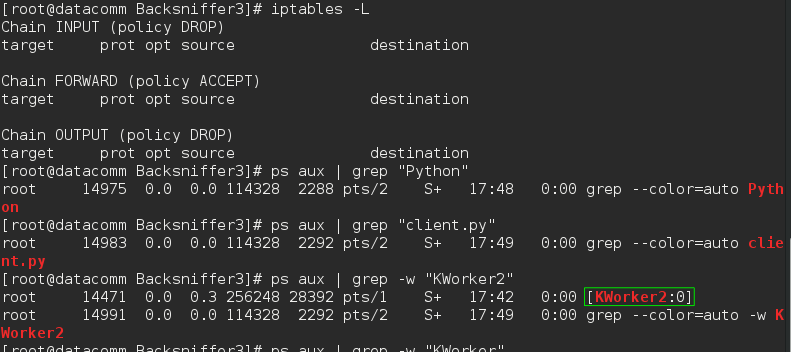


Fig 1.2- Client terminal showing that the process is not shown as client.py or as a python script. Client process is “KWorker2:0” as specified in the initial command.

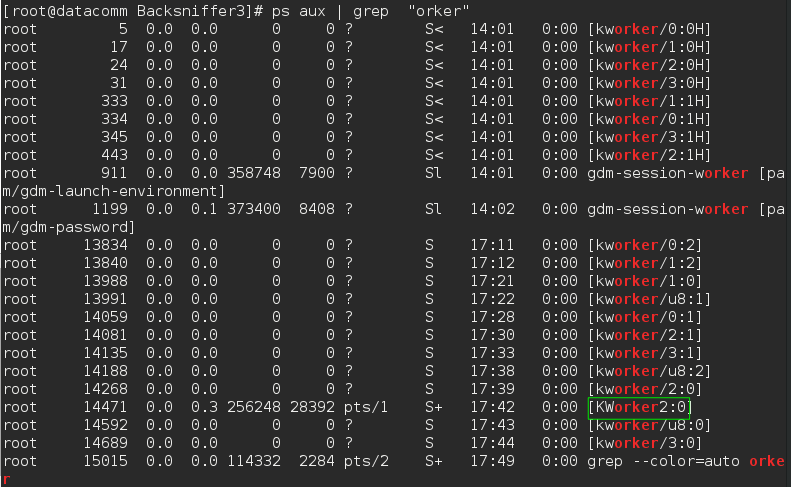


Fig 1.3 – Terminal displayed shows all other running KWorker threads. This puts into perspective what it would be like for a user or analyst viewing currently open processes.

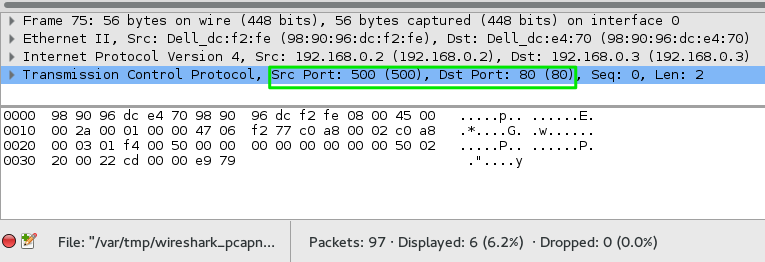


Fig 2.1 – The packets are being sent from the right source port to the right destination port

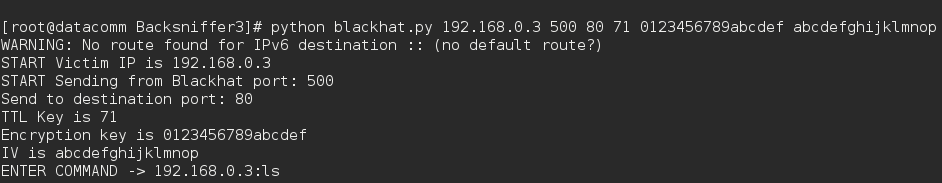


Fig 3.1 – Terminal Output showing the command that the attacker has sent.

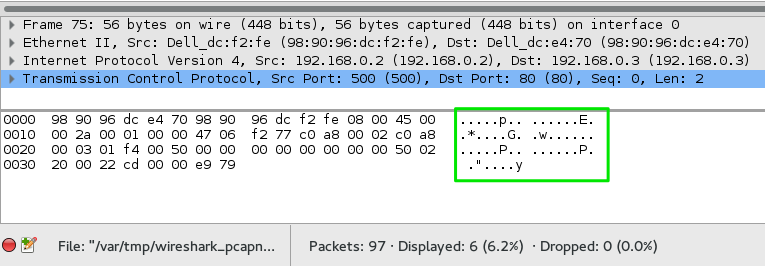
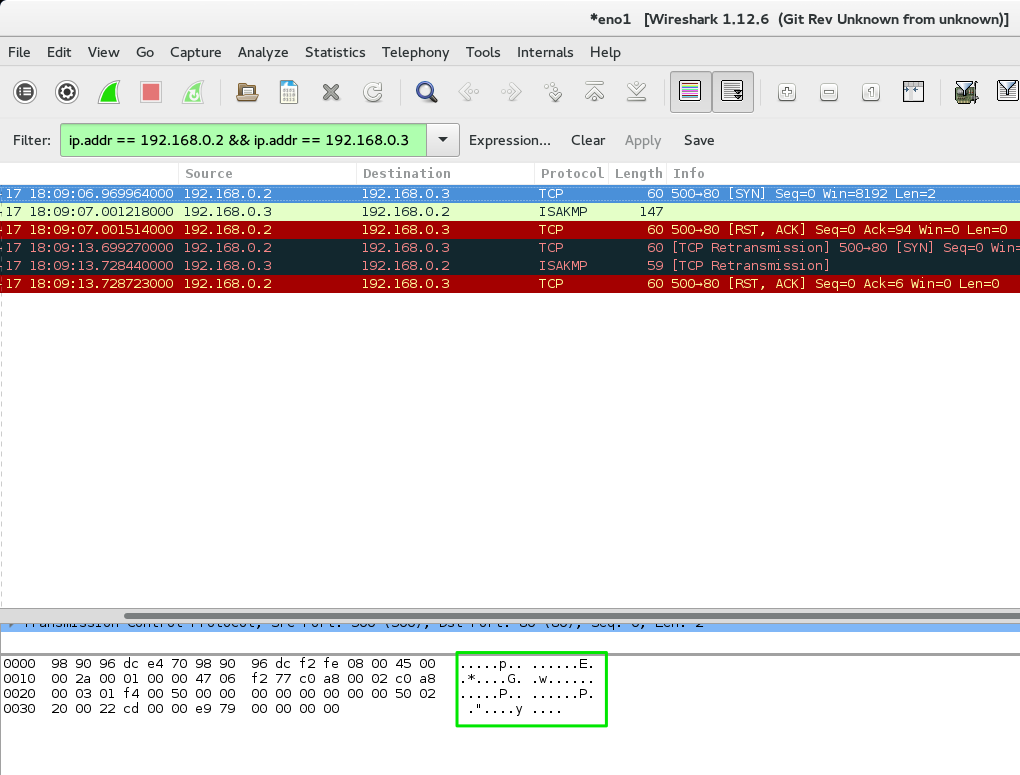


Fig 3.2 – The Wireshark capture at the attacker showing that the packet’s data is not in plain-text.

Fig 3.3 – The Wireshark capture at the client showing that the received packet is encrypted after being in transit.

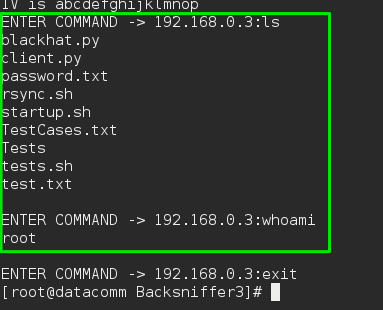


Fig 4.1 – Results are sent back.

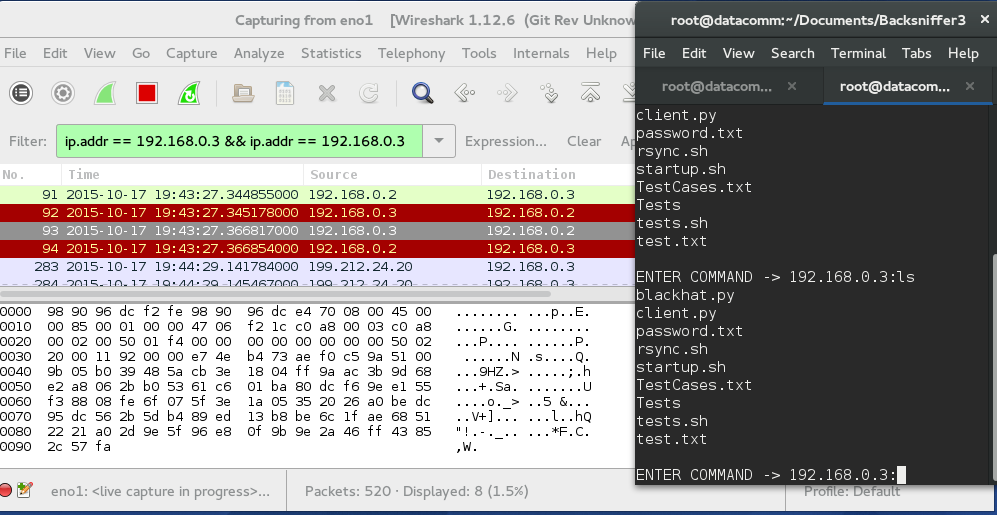


Fig 4.5 – The result coming back is encrypted

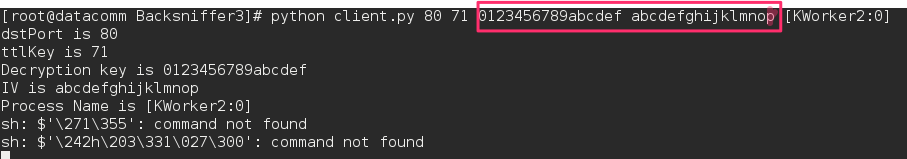


Fig 6.1 – Client: Backdoor start with different key.

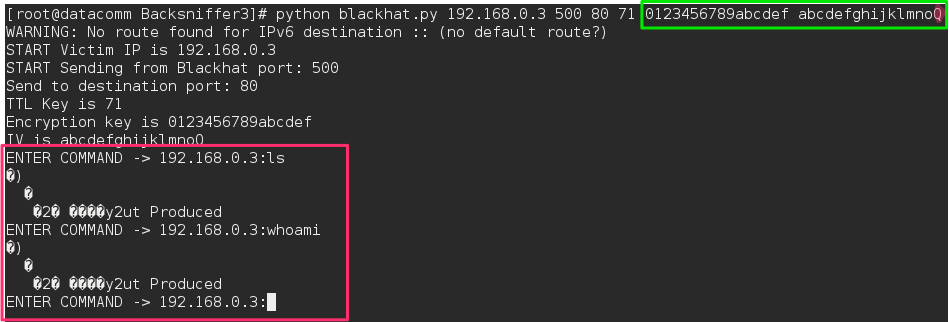


Fig 6.2 – Attacker: The result is wrong if the wrong encryption keys and decryption keys are specified