COMP 8505 Final Project

Covert Comm Application

Preliminary Design Documentation

Ian Lee & Luke Tao

Table of Contents

Introduction

This design documentation is meant to give an overview of the structure and the timelines involved in the final project. All the constraints specified in the final project documentation will be observed, with the exception of port knocking.

Our objective is to bring together the stealth and backdoor concepts that we learned in class and integrate them into a covert comm application. In addition, this project is to help us learn how that such a powerful application can help us infiltrate access to a network undetected, as well as exfiltrate sensitive data from a network.

To give an overview of how the backdoor server should work, the disguised backdoor will listen for client packets that contain encrypted, shell-based commands on a particular port. Once a client packet has been captured, the server will decrypt and authenticate the payload, parse the command, and two things can happen: 1) The server will send the command results back to the client encrypted or 2) The server sends back encrypted contents of a file. Either way, it goes through a separate covert channel in order to avoid detection. Furthermore, the packets will be sent in separate intervals specified by the user configuration file.

As for the client, after entering the correct password, it will simply send either a UNIX shell command or a server-defined command that only the server will recognize.

You will find that our design specifications contain multiple features that we plan to implement into our application. Furthermore, the diagrams and pseudo code contained in the report gives a general idea of how our client and server should work, as well as assigning different tasks with associated deadlines.

Design Specifications

Configuration Files

The client and server will have their own respective configuration files. Note that we may use **libconfig** for configuration file processing. Here are the following parameters for the client and server:

Client:

- Target Host
- Target IP
- Protocol Used (TCP, UDP, ICMP, default is TCP). The protocol MUST match the server's protocol capture filter.

Server:

- Target directory or file to monitor
- Host to send packet (For relay purposes, default will be back to client)
- Port to send packet (For relay purposes, default will be on a fixed port value)
- Interval to send packets covertly (Default is arbitrary intervals)
- Daemon mode (True or False)
- Port to listen for incoming packets
- Protocol used (TCP, UDP, ICMP, default is TCP). The protocol MUST match the client's protocol capture filter.

Covert Channel Payloads

Decrypted payloads will be in the general format of:

```
Authentication(password)␣packetType␣[arguments]␣data
```

Server → Client Packet Types

Packet Type Value	Packet Type	Arguments
0	Command Output	N/A
1	File Transfer	`mode filename`

File Transfer Modes:

- **0:** Create - First Packet of File
  - On client, create empty file of `filename` and append data to it.
  - If `filename` exists already, rename existing file to `filename.#`
- **1:** Append - Subsequent Packets of file.
  - On client, open file of `filename` and append data to it.

Example Payload: [Password 1 0 test.txt Helloworld]
This packet is a file transfer type (indicated by 1) and creating a file (indicated by 0) called “test.txt” with the contents “Helloworld”. This payload will be sent back to the client through the covert channel.

Client → Server Packet Types

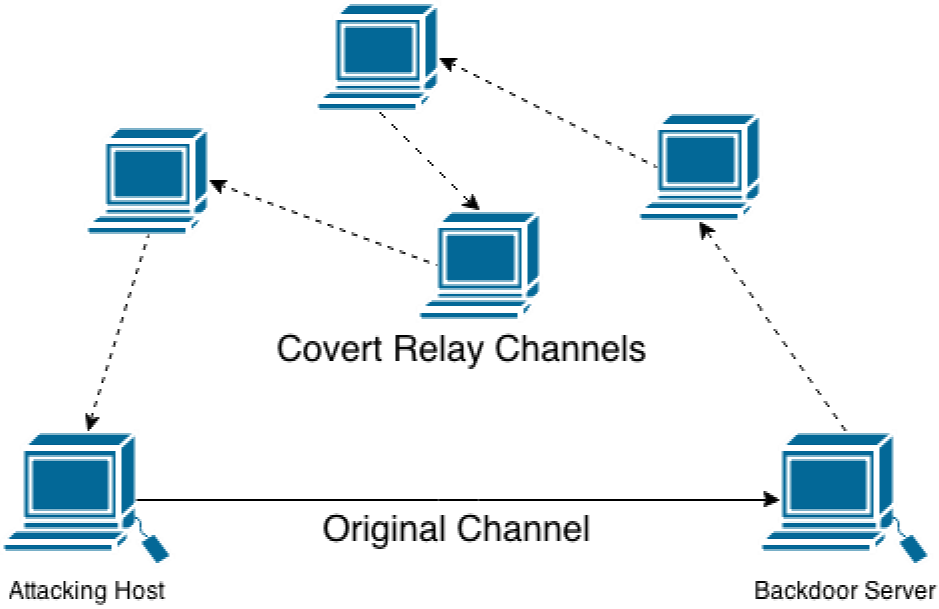
Packet Type Value	Packet Type	Arguments
0	Shell Command	N/A

Here, the client sends a shell command regardless if it’s UNIX-type or server-defined type as the server will determine this.

Port Knocking

The use of libpcap and per packet authentication make port knocking unneeded. For the time being, port knocking is somewhat useless when the server is covertly listening to incoming packets, regardless if the firewall is blocking or not. The server can check all packets against the libpcap filter rules, then capture and authenticate any and all packets that match the rules. For this reason, there’s no need to add an extra layer of complex security.

Covert Relay Channels



The above diagram shows how covert relay channels work once we’re able to implement the other things in mind. The original channel route is a one-way communication where the client simply sends shell commands, while the covert relay channels forwards the encrypted results of the commands or file contents back to the client. Essentially, we are making our own version of Netcat Relays where it’s difficult to trace back the origin of the attack or in this case, the covert channel sender if done properly and it allows us to redirect our data through ports permitted by the firewall.

Server-Defined File Commands

In addition to processing UNIX commands, the client will be able to send server-defined commands that will be able to control the server when to send the file through inotify. In other words, if the client sends an append command, the server will decrypt the payload, find the filename and append the data to that specific file. Assuming that the server was monitoring for modifications of a file, the server will send the file covertly back to the client.

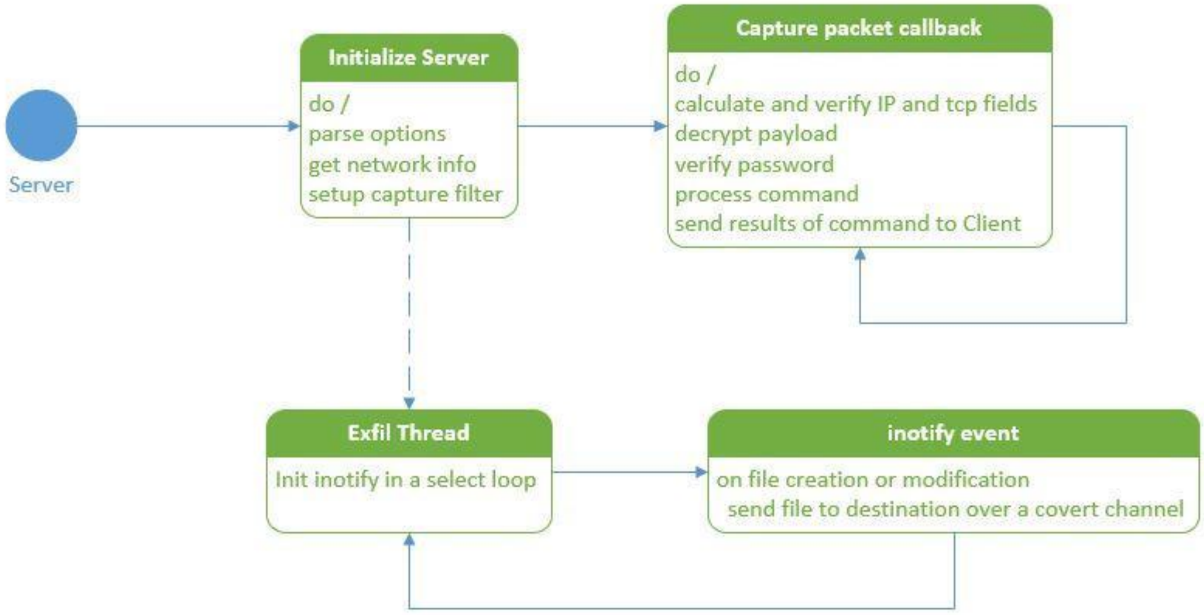
ISAAC Cipher

The encryption cipher that we will use for this project is the ISAAC cipher, where it's known to be a cryptographically secure pseudo-random number generator (CSPRNG) and stream cipher. This cipher is known to be very fast, especially when optimized, and portable to most architectures in almost all programming and scripting languages.

Protocols Used

Aside from the TCP protocol, the user will be able to use the UDP and ICMP protocol. Nothing else will change in terms of payload other than the protocol headers themselves. However, the backdoor server and the client must use the same protocol.

Server State Transition Diagram


Server
Initialize Server
do /
parse options
get network info
setup capture filter
Capture packet callback
do /
calculate and verify IP and tcp fields
decrypt payload
verify password
process command
send results of command to Client
Exfil Thread
Init inotify in a select loop
inotify event
on file creation or modification
send file to destination over a covert channel

Server Pseudo Code

Main Server Initialization function

```
{

    Find a capture device (lookupdev or listalldevs)
    Get netmask and IP

    Print capture info
    Set filter expression
    Use pcap loop to callback

    Clean up stuff
}
```

Mask Process (Server)

```
{
    Set process name passed in and return
}
```

Parse Options (Server)

```
{
    Set struct options

    Set defaults if user doesn’t specify them
    While parsing
    {
        Set Daemon to true if user wants it
        Display help if user requests it
    }
}
```

Print Usage (Server)

```
{
    Print the Following Options:
    Running as daemon with -d
    Masking process as (name)
}
```

Main Server

```
{
    Check to see if user is root otherwise exit
    Parse options

    Print Settings if required from -h
    Daemonize process if required from -d
    Mask process (function)
    Start Exfil thread
    Server Initialization
}
```

Callback function for packet Handling

```
{
    Calculate IP header offset
    Verify the IP header length
    Watch for packets defined by filter
    Calculate tcp/udp/icmp header offset
    Verify TCP/UDP/ICMP header length

    Calculate the payload offset and size
    Decrypt the payload

    Grab password and command field
    If incorrect password, return
    If we’re the server and the password is correct
        execute the send command then send the results back to the client
    Elseif is Client and password is correct
        if filetransfer
            if mode is create
                if file exists
                    rename(backup) old file
                create empty file
            open file and append payload data to file
        else is command output
            print command results to stdout
}
```

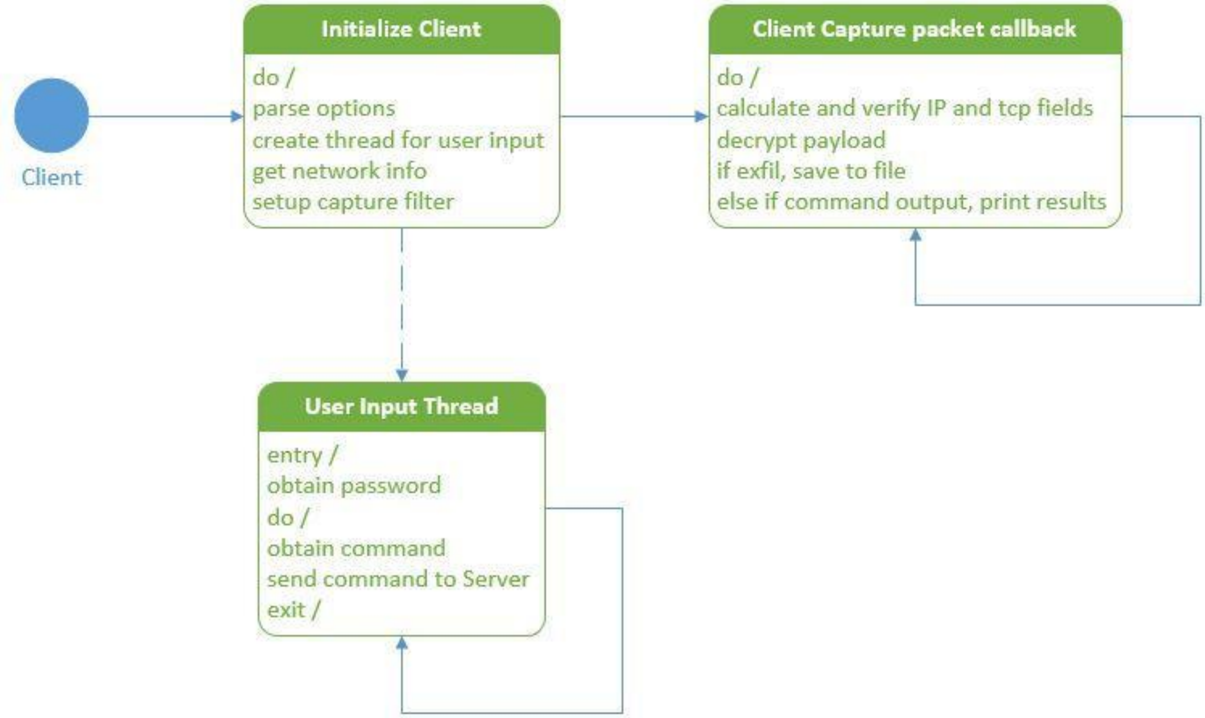
Exfil Thread

```
{
    Obtain directory locations and destination address
    Obtain inotify descriptors for file creation or modification
    Select loop on inotify descriptors
        On file creation or modification pass file descriptor to file send function
}
```

File Send Function

```
{
    package file contents into packet payloads
    if the first packet
        file creation mode
        else
                file append mode
        send packets over covert channel
}
```

Client State Transition Diagram



Client Pseudo Code

Start Client

```
{
    start thread to process user input
    start capturing packets
}
```

User Input Thread

```
{
    Get password from user prompt
    loop until user types quit
        Get command from user prompt
        encrypt password and command as payload
        send packet to server
}
```

Client Capture Packet Callback

```
{
    //same as capture callback function in server pseudo code.
    //use client conditional statements
}
```

Task Assignments

Task	Assigned To	Due Date
Upgrading Existing Backdoor Encryption to ISAAC Cipher	Luke	Nov. 12
File Exfiltration	Ian and Luke	Nov. 15
Inotify Monitoring	Ian	Nov. 19
Integrate Additional Protocols (UDP, ICMP)	Luke	Nov. 23
File Command Processing (If Time Permits)	Ian	Nov. 23
Relay Implementation (If Time Permits)	Luke	Nov. 26
Client, Server Backdoor, and/or Relay Integration	Ian and Luke	Nov. 28
Final Testing In Labs	Ian and Luke	Nov. 30

Upgrading Existing Backdoor Encryption to ISAAC Cipher

Essentially, we take our existing backdoor assignment from COMP 8505 Assignment 2 that uses the simple XOR encryption scheme and upgrade it to the ISAAC scheme. This task is making sure that the ISAAC encryption scheme works for UNIX command processing.

File Exfiltration

This next phase of our project will be implementing a simple file extraction where a file is sent through our covert channel. This task includes our backdoor server reading and starting in accordance to our configuration file, modifying our packet payload so that it returns our contents of a file back to the client, as well as making sure that the encryption/decryption ISAAC scheme still works from the previous task. In addition, a covert channel will be in place so the packets sent will be in arbitrary intervals in order to avoid detection.

Inotify Monitoring

This task includes the creation of a process or thread that will monitor a file using the inotify utility and builds upon the file exfiltration task. Furthermore, the monitoring of a specific directory or file will be specified in the configuration file. As soon as a file is created or modified under the specified name, the file exfiltration will begin.

Integrate Additional Protocols (UDP, ICMP)

The user will be able to specify 2 more protocols, UDP and ICMP in addition to TCP in the configuration file. However, the data will be embedded in the payload portion of UDP and ICMP, just like the TCP so nothing else will change other than the headers themselves.

File Command Processing (If Time Permits)

This phase will implement the server's own defined commands for file processing. Not only will the client be able to send UNIX-type commands, but commands that the server itself would understand. Having this feature will allow both the client and the server to have control of when to send the file. (Refer to the "Covert Channel Payloads" portion of the Design Specifications section). Depending on the time constraints on how far we get done, this feature will be implemented if time permits.

Relay Implementation (If Time Permits)

A relay portion will be used as a way of extending our covert channels. The relay implementation will act as a separate application in addition to our attacking client and backdoor server. Essentially, this relay will listen on a particular port using libpcap as usual and then after capturing and authenticating the packet, it will be forwarded to another relay or client. Overall, this will be our own version of the Netcat Relay functionality. Like the File Command Processing Task, the relay will be implemented if time permits us to do so.

Client, Server Backdoor, and/or Relay Integration

After the previous tasks are done, we will integrate our client and server in the labs though we will integrate them when we are working on the previous tasks. This phase will undergo debugging phases and making sure that the integration of the client and server is fully complete.

Final Testing In Labs

After integration is done, we will start the testing phase of our applications to make sure that they conform to the requirements of the final project. A technical report will be written up that will include our final design and testing documents, as well as our summary of our protocol that we implemented for our covert channels and our recommendations on how to prevent such an activity.