CIS4362 Introduction to Cryptology

GinVPN-Final Project

Alexander Krasny

December 9, 2020

# System Intentions

GinVPN is designed to provide security for users when browsing the internet in unsecured locations. It uses a proxy to intercept and encrypt HTTP requests with an AES-256bit encryption algorithm and sends them to a server that decrypts the messages, and performs the request. The response from this request is encrypted, and sent back to the proxy, where it is decrypted and sent back to the original requester.

# System Functionality

GinVPN has and AES module that performs the encryption. The constructor takes a 32byte key that can be either a string or bytes. Then it performs key expansion and generates the required round keys to perform 256 bit encryption. It has encrypt and decrypt methods that run the through the steps of the AES modules using all of the required helper methods.

GinVPN utilized proxy.py, an open source proxy library that allows the creation of plugins that can modify outgoing requests and incoming responses. GinVPN incorporates a plugin for proxy.py that checks every outgoing HTTP/HTTPS request. For HTTP requests, the plugin creates a byte string containing the method and the url, separated by spaces, the headers, separated by b‘\r\n’, and the body, separated by an extra b‘\r\n’. Spaces are used for the first part, because urls never contain spaces, and the other characters are used because headers and body in HTTP requests are separated by CRLF or carriage return line feed. Responses are encrypted later on in a similar manner, but with the status code encrypted as 2 bytes instead of the method. After the proxy formats the request, it encrypts it using the AES module, and then the proxy modifies the request to change the destination to the GinVPN server.

Upon receiving a request, the server separates the components from the request, and formats them into a manner that can be used by the python requests modules. Then the server makes the request, and formats the response in the aforementioned format. Next, it encrypts it and sends it back to the proxy plugin. The plugin first receives the response header which contains the zander-approved flag, marking that the next chunk the program receives as needing decryption. It decrypts the next chunk it receives and turns it into a response object using the build\_http\_response() method provided by the proxy.py library. This request is sent back to the requester as part of proxy plugin.

# Testing

The system was tested using curl requests to various websites. The tests folder will include several test commands with output files. The encryption algorithm is slow, so the server gets overwhelmed with multiple requests, so the system functions optimally with one request at a time.

# Resources

* <https://pypi.org/project/proxy.py/>
* <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>
* <https://docs.aiohttp.org/en/stable/>
* <https://requests.readthedocs.io/en/master/>
* <https://www.tutorialspoint.com/http/http_requests.htm>
* <https://stackoverflow.com/questions/45442396/a-pure-python-way-to-calculate-the-multiplicative-inverse-in-gf28-using-pytho>
* <https://packaging.python.org/>
* <https://www.w3schools.com/python/gloss_python_global_variables.asp>
* <https://docs.python.org/3/library/urllib.parse.html>
* <https://www.codeproject.com/Questions/1060070/ConnectionResetError-WinError-An-existing-connecti>
* <https://requests.readthedocs.io/en/master/user/quickstart/#raw-response-content>
* <https://docs.python.org/3/c-api/memoryview.html>
* <https://www.w3schools.com/python/python_try_except.asp>
* <https://www.geeksforgeeks.org/response-content-python-requests/>
* <https://www.w3resource.com/python/python-bytes.php>

# Development Environment

The program was developed on a Windows 10 machine with Visual Studio Code with the python extension. It’s hosted on PyPi and github, with the PyPi live version being updated after major changes.

# File Structure

GinVPN

| \_\_init.py

| gin\_vpn\_config.py

| gin\_vpn\_server.py

|

|--------- AES

| | \_\_init\_\_.py

| └--------- AES.py

|

└--------- Plugin

| \_\_init\_\_.py

└--------- gin\_vpn\_plugin.py

# Software Architecture

## AES.py

class AES:

Contains all the functions needed to perform AES encryption and decryption

def \_\_init\_\_(self, master\_key):

Constructor for AES class. Generates round keys give a master key.

def xor\_bytes(self, a,b):

returns a bytes object containing a xor b

def sub\_word(self, word):

returns the s\_box transformation on the bytes in word

def rot\_word(self, word):

returns the word rotated left one position

def key\_expansion(self, key, word\_array):

generates 60 words from the master key and stores them in word array by modifying the last word with the previous 3 methods as specified by AES standards.

def add\_round\_key(self, txt, key):

returns txt xor-ed with key

def sub\_bytes(self, msg):

returns the s\_box transformation of msg

def shift\_rows(self, msg):

returns msg with rows shifted as specified by AES standards

def multiply(self, b,a):

return galios multiplication of b and a

def mix\_col(self, col):

returns col galios multiplied by the column\_mix array.

def mix\_cols(self, msg):

return an bytearray made from concatenation of each column of message after mix\_col

def inv\_sub\_bytes(self, msg):

returns the msg after the application of the inverse s\_box transformation

def inv\_shift\_rows(self, msg):

returns msg with row shifts of shift\_rows reversed as specified by AES standards

def inv\_mix\_col(self, col):

returns col galios multiplied by the inv\_column\_mix array.

def inv\_mix\_cols(self, msg):

return an bytearray made from concatenation of each column of message after inv\_mix\_col

def pad(self, msg):

returns the msg padded to a multiple of 16 bytes with the byte representing the number of pad characters

def un\_pad(self, msg):

returns the reverse of the padding process

def word\_keys(self, words):

returns the concatenation of words

def cipher(self, plaintext):

returns the result of the AES encryption algorithms rounds on one block of the full message using the mutation functions

def inv\_cipher(self, ciphertext):

returns the result of the AES decryption algorithms rounds on one block of the full cipher text using the inverse mutation functions

def decrypt(self, msg):

returns the result of inv\_cipher on all blocks of msg

def encrypt(self, msg, string=True):

returns the result of encrypt on all blocks of msg. if string is True, encrypt expects a string, and if false it expects bytes.

## gin\_vpn\_config.py

def get\_valid\_int\_selection(prompts, min, max, default=''):

prompts-array of 3 messages: initial prompt, out of range error, and invalid selection error

min and max - inclusive bounds of int input

default string value of default int

prompts the use to input a number between min and max. If no entry, will default to the default, unless there is no default provided. Returns the value

def yes\_no():

prompts the user with a yes/no choice. Returns a bool, yes returns true.

def menu\_choice(name, options\_list):

Creates a menu with the name ‘name’ and options from options list. Will ask user to select an option, and uses get\_valid\_input\_selection() to get a choice integer to return.

def get\_string(prompt, default):

prompt- message to user about what to input

default- value to use if no values is provided

prompts user to input a string, and returns the value

def gen\_key():

returns a 32 byte key created using the python secrets module

def save\_key\_svr\_port(filename, key, server\_adr,server\_port):

saves current options to a python file that acts as a module to import variables from for all programs

def main():

Checks if GinSettings.py exists. This file contains the definitions of variables the programs use. If it doesn’t exit, it runs first time setup, which creates a new key and prompts the user to enter server information.

## gin\_vpn\_plugin.py

class GinVPNPlugin(HttpProxyBasePlugin):

Contains the definition of the proxy.py plugin

def \_\_init\_\_(self, \*args: Any, \*\*kwargs: Any) -> None:

Runs the super constructor, and adds the server, port and an instance of AES to the object

def before\_upstream\_connection(

            self, request: HttpParser) -> Optional[HttpParser]:

Run every time the proxy receives an HTTP/S request. On HTTP requests, this method creates a bytestring of the url, headers, and request body. It runs AES.encrypt() and then modifies the request to send to server.

def handle\_client\_request(

            self, request: HttpParser) -> Optional[HttpParser]:

Not used as part of this plugin, but runs for every chunk of data sent by client.

def handle\_upstream\_chunk(self, chunk: memoryview) -> memoryview:

called whenever a chunk is received from the server.

Parses request into an Http response object. If the header’s contain ‘zander-approved’, it means the next chunk it receives is encrypted text.

When it receives this encrypted text, it runs AES.decrypt(), and then parses the result into its constituent parts, and constructs it into a response object to send back to the client.

def on\_upstream\_connection\_close(self) -> None:

Not used as part of this plugin, but called when connection to the client is torn down.

## gin\_vpn\_server.py

def format\_response(r):

creates a bytestring out of a request.response object. It formats it as specified in the system functionality section.

def make\_request(req\_type, req\_url, headers\_dict, req\_body):

Uses requests module to make a request given request type (i.e. GET, POST, HEAD, ect.), a dictionary of headers, and the request body in byte form.

async def recv\_msg(request):

Called when a POST request is made to the / path of the server. It decrypts the body of the request, and splits it up into constituent parts. It then creates a dict of headers from a list. Next, it calls make\_request() and formats the request and encrypts it. It then puts the encrypted response in the body of its response, with the ‘zander-approved’ header, unless there is an error, which cause it to return a 500 response with the error message.

def main():

Main is the entry point of GinSever. It imports the setting created in GinConfig, makes an AES object, and sets up the async event loop and webserver.