**Министерство науки и высшего образования РФ**

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**факультет БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ**

**ТЕХНОЛОГИЙ**

Основы стеганографии

**Лабораторная работа №1**

Работу выполнил:

Студент группы N3347

Курятов Е. А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверено:

Давыдов В. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2020

**Цель работы:**

Целью данной лабораторной работы является получение практического опыта в применении текстовых стеганографических методов для сокрытия и последующего извлечения сообщения из стегоконтейнера, а также последующий анализ исходного текста и стегоконтейнера.

**Теория:**

Стеганография, использующая текстовые контейнеры для скрытия данных, называется текстовой. При скрытии информации используются допущения при расположении и количестве символов в тексте, не учитываемые при прочтении человеком и компьютерном анализе текстового файла. Это может быть дополнительное количество пробелов и знаков табуляции в разных частях строки, чередование некоторых не учитываемых служебных символов, больших и маленьких букв, букв из разных алфавитов, но похоже выглядящих. К методам текстовой стеганографии относят: форматирование, изменение порядка следования маркеров конца строки, метод хвостовых пробелов, метод знаков одинакового начертания и изменения кода пробела [1].

Метод знаков одинакового начертания предполагает замену или отказ от такой замены русского символа латинским символом того же начертания. Для скрытия стего файл-контейнер считывается посимвольно. Если текущий символ является буквой, осуществляется его замена на латинский символ из таблицы замен (если нужно закодировать бит 1 стеганосообщения). Если же нужно закодировать бит 0, символ просто пропускается. Приведу фрагмент программы, реализующий замену символов [2].

Метод хвостовых пробелов предполагает дописывание в конце каждой строки файла-контейнера одного пробела, в случае кодирования единичного бита стеганосообщения. Если нужно закодировать нулевой бит, пробел в конце строки не дописывается. При сокрытии стего текстовый файл считывается построчно. Из конца строки удаляются все пробельный символы (пробелы, знаки табуляции, символы возврата каретки и новой строки), а затем в зависимости от значения текущего бита стеганосообщения, представленного в двоичном виде, принимается решение о дописывании в конец строки одного пробела. Преобразованная таким образом строка записывается в файл-результат [3].

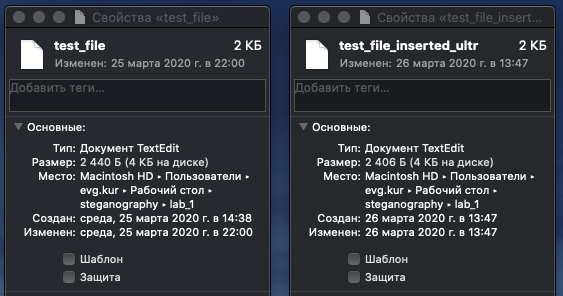
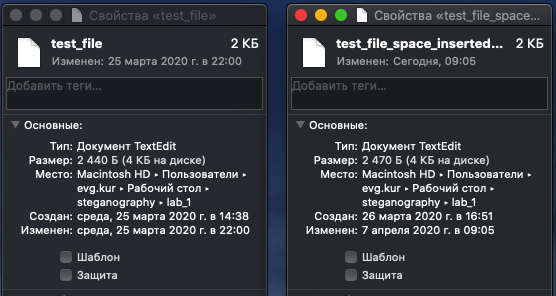
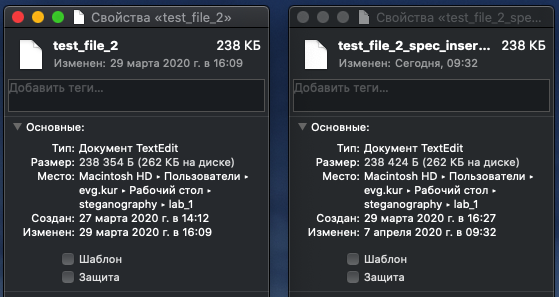
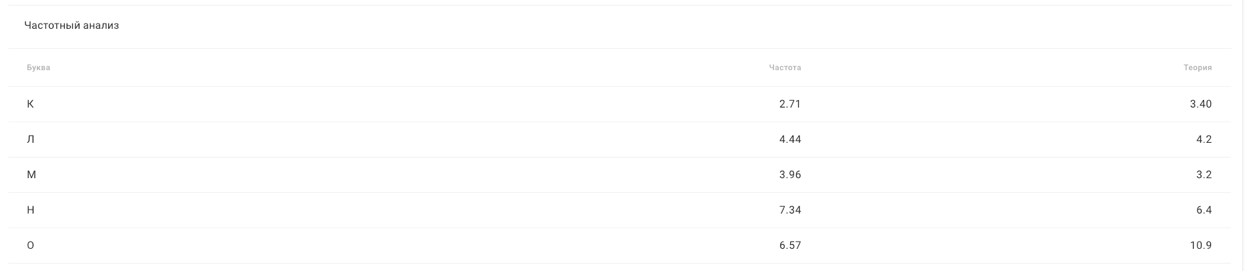
Так же существует метод добавления служебных символов, который предполагает встраивание в текст служебных символов: и тех которые мы можем увидеть в повседневной жизни, например запятых или тире, так и каких-либо специальных, непечатных.

**Практика:**

Для реализации вышеописанных методов было написано четыре программных модуля на языке Python версии 2.7. Такой выбор связан с тем, что у данного языка имеется обширный набор методов работы со строками и хорошо продуманная система работы с ООП. Первые три модуля – непосредственная реализация трех методов, а четвертый – реализация функций, необходимых для функционирования всех трех модулей.

Разберем структуру модулей на примере реализующего первый метод (листинг в приложении 1). В этом модуле присутствуют два вспомогательных метода \_\_init\_\_ и \_\_detect\_lenguage (которые запускаются соответственно при инициализации класса и для обнаружения языка стегконтейнера) и два основных insert\_text и get\_secret\_text: для внедрения секретного текста и для его поиска. Для операций пере вода из строки в битовую строку и обратно используются методы четвертого модуля steganography (листинг в приложении 4). Остальные модули реализованы сходным образом и отличаются только вспомогательными методами.

Благодаря написанию подобных модулей стал возможен анализ реализованных методов:

1. Подсчет объема встраивания и оценка целесообразности..
   1. **Метод прямой замены символов**  
      В этом методе я использовал замену русской буквы «о» на английскую и наоборот. Частотная характеристика буквы «о» - ~11%, соответственно объем встраивания для русского текста 0.11\*k бит, где k – количество символов в контейнере.
   2. **Метод с использованием дополнительных пробелов**В этом методе я встраивал пробелы в конце строк – соответственно объем встраивания для данного метода k бит, где k – количество символов переноса строки в документе
   3. **Метод с использованием специальных символов**В этом методе я заменял один из двух символов тире на символ длинного тире, соответственно для данного метода объем встраивания k бит – где k это количество продублированных тире.
2. Оценка размера контейнера до встраивания и после
   1. **Метод прямой замены символов**В этом методе при использовании русскоязычного контейнера размер файла УМЕНЬШАЕТСЯ на l бит, где l – количество битов-единиц. Это происходит от того, что в кодировки UTF-8 символы английской раскладки на 1 бит короче. В моем случае было зашифрованно сообщение длиной 56 бит: 11101011101100111010011100101100001110101111001011101011 – в результате файл уменьшился на 34 бита ~1,4% исходного файла  
      
   2. **Метод с использованием дополнительных пробелов**В данном методе объем файла будет увеличиваться на 6 бит за каждый бит-единицу в передаваемом сообщении. В моем случае было зашифровано сообщение: 11011011100101110011111000011101100111010111011 – длиной 47 бит, а объем файла увеличился на 30 бит ~1,22% исходного файла  
      
   3. **Метод с использованием специальных символов**В данном методе объем будет увеличиваться на 2 бита за каждый бит-1 в передаваемом сообщении.   
      
3. Проведение экспертной оценки: содержится ли в файле некоторое сообщение. Метод: статистический анализ.
   1. **Метод прямой замены символов**Этот метод является уязвимым для статистического анализа, так как завязан на замене самой часто-встречающейся буквы русского алфавита на другую. Падение характерной частоты может вызвать подозрения при анализе. Более того в тексте появляется только одна буква английского алфавита, встречающаяся с большей частотой.  
      
   2. **Метод с использованием дополнительных пробелов**Этот метод частично уязвим менее уязвим для частотного анализа, так как в обычном тексте количество пробелов больше чем <количество слов>\*2-<количество строк>, тогда как после добавления текста в контейнер количество пробелов становится равным <количество слов>\*2-<количество строк>+<количество битов-1 вв спрятанном собщении>.  
      
   3. **Метод с использованием специальных символов**Этот метод наиболее уязвим к полному частотному анализу, так как в тексте появляется символ, которого по идее этого текста в нем быть не должно, так как для данного метода подбирается контейнер в котором все длинные тире заменены на двойные короткие. Однако многие сервисы не делают частотного анализа по всем символам – только по буквам и в таком случае обнаружить факт наличия спрятанного текста не получится.

**Вывод:**

При выполнении данной лабораторной работы мною были изучены основные методы текстовой стеганографии. Я научился применять их и проводить последующую оценку их применению. По результатам работы были сделаны следующие выводы:

Метод замены символов одинакового начертания легко обнаружим при статистическом анализе, в отличие от метода с использованием дополнительных пробелов, а реализованный мною метод добавления служебных символов, работает только в определенных стегконтейнерах и при присмотре в текстовых редакторах с моношринными шрифтами.

Если я захочу в будущем поместить стего в текст, то, скорее всего, воспользуюсь методом с дополнительными пробелами, так как его сложнее всего найти в тексте.

**Список использованной литературы:**

1. Текстовая стеганография: Запись между строк [Электронный ресурс] // Блог о стеганографии. 2012. 28 апреля. URL: https://www.nestego.ru/2012/04/blog-post\_28.html?m=0 (дата обращения 09.04.2020)
2. Текстовая стеганография: Метод знаков одинакового начертания [Электронный ресурс] // Блог о стеганографии. 2012. 5 мая. URL: http://www.nestego.ru/2012/05/blog-post\_05.html (дата обращения 09.04.2020)
3. Текстовая стеганография: Метод хвостовых пробелов [Электронный ресурс] // Блог о стеганографии. 2012. 3 мая. URL: http://www.nestego.ru/2012/05/blog-post\_03.html (дата обращения 09.04.2020)

**Приложения:**

Приложение 1: Модуль реализующий метод прямой замены символов

|  |
| --- |
|  |
| class language\_swap: |
|  | steg\_core = 'empty class' |
|  | en\_symbol = 'o' |
|  | ru\_symbol = 'о'.decode('utf-8') |
|  | service\_symbol\_count = 1 |
|  | inserting\_symbol = False |
|  |  |
|  | def \_\_init\_\_(self): |
|  | self.steg\_core = steganography.core() |
|  | self.steg\_core.log('SUCCESS', ' === Lab1 language swap method initialized === ') |
|  |  |
|  | def \_\_detect\_lenguage(self, contents): |
|  | i = 0 |
|  | while( (contents[i] != self.en\_symbol) and (contents[i] != self.ru\_symbol) ): |
|  | i += 1 |
|  | if ( contents[i] == self.en\_symbol ): |
|  | self.inserting\_symbol = self.ru\_symbol |
|  | return 1 |
|  |  |
|  | elif ( contents[i] == self.ru\_symbol ): |
|  | self.inserting\_symbol = self.en\_symbol |
|  | return 1 |
|  | else: |
|  | self.steg\_core.log('ERROR', 'Can not find ' + self.en\_symbol + 'in the text.') |
|  | return 0 |
|  |  |
|  |  |
|  | def insert\_text(self, file\_path, insert\_text, lang): |
|  |  |
|  | # Initialize inserting |
|  |  |
|  | self.steg\_core.log('SUCCESS', 'Inserting \'' + str(insert\_text) + '\' into ' + str(file\_path)) |
|  | inserted\_file\_name = file\_path + '\_swap\_inserted\_' + insert\_text[0:4] |
|  |  |
|  | # Get contents to insert |
|  |  |
|  | inserting\_file = open(file\_path, 'r') |
|  | inserting\_contents = inserting\_file.read().decode('utf-8') |
|  | inserting\_file.close() |
|  |  |
|  | # Detect file lenguage |
|  |  |
|  | detected = self.\_\_detect\_lenguage(inserting\_contents) |
|  |  |
|  | if ( not detected or not self.inserting\_symbol ): |
|  | self.steg\_core.log('ERROR', 'Inserting stopped') |
|  | return 0 |
|  |  |
|  | self.steg\_core.log('TEST', 'detected symbol: ' + str(ord(self.inserting\_symbol[0]))) |
|  |  |
|  | # Check if inserting text is too long |
|  |  |
|  | bit\_string = self.steg\_core.str\_to\_bit(insert\_text) |
|  | available\_bit\_count = 0 |
|  | for contents\_symbol in inserting\_contents: |
|  | if ( (contents\_symbol != self.inserting\_symbol[0]) and ((contents\_symbol == self.en\_symbol) or (contents\_symbol == self.ru\_symbol)) ): |
|  | available\_bit\_count += 1 |
|  |  |
|  | self.steg\_core.log('TEST', 'available\_bit\_count: ' + str(available\_bit\_count)) |
|  |  |
|  | if ( available\_bit\_count < len(bit\_string) + self.service\_symbol\_count ): |
|  | self.steg\_core.log('ERROR', 'Iserting text is too long! It contains ' + str(len(bit\_string)) + ' bits, when text can contain only ' + str(available\_bit\_count) + ' bits.') |
|  | return 0 |
|  |  |
|  | # Insert secret test to contents variable |
|  |  |
|  | self.steg\_core.log('TEST', 'bit\_string: ' + bit\_string) |
|  | for i in range(0,len(bit\_string)): |
|  | counter = 0 |
|  | for j in range(0,len(inserting\_contents)): |
|  | if ((inserting\_contents[j] == self.ru\_symbol) or (inserting\_contents[j] == self.en\_symbol)): |
|  | if ( counter == i and bit\_string[i] == '1'): |
|  | inserting\_contents = inserting\_contents[0:j] + self.inserting\_symbol + inserting\_contents[j+1:] |
|  | counter += 1; |
|  |  |
|  |  |
|  | changed\_file = open(inserted\_file\_name, 'w') |
|  | changed\_file.write(inserting\_contents.encode('utf-8')) |
|  | changed\_file.close() |
|  |  |
|  | self.steg\_core.log('SUCCESS', 'Text successfully inserted!') |
|  |  |
|  | def get\_secret\_text(self, file\_path, lang): |
|  |  |
|  | self.steg\_core.log('SUCCESS', 'Getting secret text from ' + str(file\_path)) |
|  |  |
|  | # Get contents from file with secret text |
|  |  |
|  | file = open(file\_path, 'r') |
|  | contents = file.read().decode('utf-8') |
|  | file.close() |
|  |  |
|  | # Detect 1 letter |
|  |  |
|  | bit\_letter = '' |
|  |  |
|  | for i in contents: |
|  | if ( (i != self.en\_symbol) and (i != self.ru\_symbol) ): |
|  | if ( ord(i) < 128 ): |
|  | bit\_letter = self.en\_symbol |
|  | else: |
|  | bit\_letter = self.ru\_symbol |
|  |  |
|  | # Getting secret bit string |
|  |  |
|  | bit\_string = '' |
|  |  |
|  | for i in range(0,len(contents)): |
|  | if (contents[i] == self.en\_symbol or contents[i] == self.ru\_symbol): |
|  | if ( contents[i] == bit\_letter ): |
|  | bit\_string += '1' |
|  | else: |
|  | bit\_string += '0' |
|  |  |
|  | # Get string |
|  |  |
|  | secret\_text = self.steg\_core.bit\_str\_to\_text(bit\_string, lang) |
|  |  |
|  | self.steg\_core.log('SUCCESS', 'Secret text: ' + secret\_text) |
|  |  |

Приложение 2: Модуль реализующий метод дополнительных пробелов.

|  |
| --- |
| class space\_method: |
|  | steg\_core = 'empty class' |
|  |  |
|  | def \_\_init\_\_(self): |
|  | self.steg\_core = steganography.core() |
|  | self.steg\_core.log('SUCCESS', ' === Lab1 space method initialized === ') |
|  |  |
|  | def insert\_text(self, file\_path, insert\_text): |
|  |  |
|  | # Initialize inserting |
|  |  |
|  | self.steg\_core.log('SUCCESS', 'Inserting \'' + str(insert\_text) + '\' into ' + str(file\_path)) |
|  | inserted\_file\_name = file\_path + '\_space\_inserted\_' + insert\_text[0:4] |
|  |  |
|  | # Check if inserting text is too long |
|  |  |
|  | bit\_string = self.steg\_core.str\_to\_bit(insert\_text) |
|  | available\_bit\_count = sum(1 for line in open(file\_path, 'r')) |
|  |  |
|  | self.steg\_core.log('TEST', 'available\_bit\_count: ' + str(available\_bit\_count)) |
|  |  |
|  | if ( available\_bit\_count < len(bit\_string) ): |
|  | self.steg\_core.log('ERROR', 'Iserting text is too long! It contains ' + str(len(bit\_string)) + ' bits, when text can contain only ' + str(available\_bit\_count) + ' bits.') |
|  | return 0 |
|  |  |
|  | # Insert secret text |
|  |  |
|  | new\_file = open(inserted\_file\_name, 'w') |
|  |  |
|  | i = 0 |
|  | for line in open(file\_path, 'r'): |
|  | if ( i < len(bit\_string) and bit\_string[i] == '1' ): |
|  | new\_file.write(line[:len(line)-1] + ' \n') |
|  | else: |
|  | new\_file.write(line) |
|  | i += 1 |
|  |  |
|  | def read\_secret\_file(self, file\_path, lang): |
|  | # Read secret file |
|  |  |
|  | bit\_string = '' |
|  |  |
|  | for line in open(file\_path, 'r'): |
|  | if ( line[len(line)-2] == ' ' ): |
|  | bit\_string += '1' |
|  | else: |
|  | bit\_string += '0' |
|  |  |
|  | print(bit\_string) |
|  |  |
|  | secret\_text = self.steg\_core.bit\_str\_to\_text(bit\_string, lang) |
|  | self.steg\_core.log('SUCCESS', 'Secret text: ' + secret\_text) |

Приложение 3: Модуль реализующий метод со специальными символами

|  |
| --- |
| class spec\_symbol\_method: |
|  |  |
|  | def \_\_init\_\_(self): |
|  | self.steg\_core = steganography.core() |
|  | self.steg\_core.log('SUCCESS', ' === Lab1 spec symbol method initialized === ') |
|  |  |
|  | def insert\_text(self, file\_path, insert\_text): |
|  |  |
|  | # Initialize inserting |
|  |  |
|  | self.steg\_core.log('SUCCESS', 'Inserting \'' + str(insert\_text) + '\' into ' + str(file\_path)) |
|  | inserted\_file\_name = file\_path + '\_spec\_inserted\_' + insert\_text[0:4] |
|  |  |
|  | # Check if inserting text is too long |
|  |  |
|  | file = open(file\_path, 'r') |
|  | file\_contents = file.read() |
|  | file.close() |
|  |  |
|  | available\_bit\_count = 0 |
|  |  |
|  | bit\_string = self.steg\_core.str\_to\_bit(insert\_text) |
|  | for i in file\_contents: |
|  | if ( i == '-' ): |
|  | available\_bit\_count += 1 |
|  |  |
|  | self.steg\_core.log('TEST', 'available\_bit\_count: ' + str(available\_bit\_count)) |
|  |  |
|  | if ( available\_bit\_count < len(bit\_string) ): |
|  | self.steg\_core.log('ERROR', 'Iserting text is too long! It contains ' + str(len(bit\_string)) + ' bits, when text can contain only ' + str(available\_bit\_count) + ' bits.') |
|  | return 0 |
|  |  |
|  | # Insert secret text |
|  |  |
|  | new\_file = open(inserted\_file\_name, 'w') |
|  |  |
|  | file\_contents = file\_contents.decode('utf-8') |
|  |  |
|  | i = 0 |
|  | for j in range(0,len(file\_contents)): |
|  | if ( file\_contents[j] == '-' and file\_contents[j+1] == '-' ): |
|  | if ( bit\_string[i] == '1' ): |
|  | file\_contents = file\_contents[0:j+1] + '–'.decode('utf-8') + file\_contents[j+2:] |
|  | i += 1 |
|  | if ( i >= len(bit\_string) ): break |
|  |  |
|  | changed\_file = open(inserted\_file\_name, 'w') |
|  | changed\_file.write(file\_contents.encode('utf-8')) |
|  | changed\_file.close() |
|  |  |
|  | def get\_secret\_text(self, file\_path, lang): |
|  |  |
|  | self.steg\_core.log('SUCCESS', 'Getting secret text from ' + str(file\_path)) |
|  |  |
|  | file = open(file\_path, 'r') |
|  | contents = file.read().decode('utf-8') |
|  | file.close() |
|  |  |
|  | bit\_string = '' |
|  |  |
|  | for i in range(0,len(contents)): |
|  | if (contents[i] == '-' and contents[i+1] == '-'): |
|  | bit\_string += '0' |
|  | if (contents[i] == '-' and contents[i+1] == '–'.decode('utf-8')): |
|  | bit\_string += '1' |
|  |  |
|  | secret\_text = self.steg\_core.bit\_str\_to\_text(bit\_string, lang) |
|  |  |
|  | self.steg\_core.log('SUCCESS', 'Secret text: ' + secret\_text) |

Приложение 4: Вспомогательный модуль:

|  |
| --- |
| class core: |
|  | logPath = "./logs/main\_log.log" |
|  |  |
|  | def log(self, type, msg): |
|  | LOG\_TYPE = { |
|  | 'TEST': '\033[36mTEST:\033[0m', |
|  | 'SUCCESS': '\033[32mSUCCESS:\033[0m', |
|  | 'ERROR': '\033[31mERROR:\033[0m', |
|  | 'WARNING': '\033[35mWARNING:\033[0m', |
|  | } |
|  |  |
|  | self.logfile = open(self.logPath, 'a') |
|  | self.logfile.write(LOG\_TYPE[type] + ' ' + str(msg) + '\n\r') |
|  | self.logfile.close() |
|  | print(LOG\_TYPE[type] + ' ' + str(msg)) |
|  |  |
|  | def str\_to\_bit(self, string): |
|  | bit\_string = '' |
|  | for i in string : |
|  | bit\_string += ''.join(format(ord(i), 'b')) |
|  |  |
|  | return bit\_string |
|  |  |
|  | def bit\_str\_to\_int(self, bit\_str): |
|  | i = len(bit\_str)-1 |
|  | result = 0 |
|  | while i >= 0 : |
|  | result += int(bit\_str[i]) \* pow(2,(len(bit\_str)-i-1)) |
|  | i -= 1 |
|  |  |
|  | return result |
|  |  |
|  | def bit\_str\_to\_text(self, bit\_string, lang): |
|  |  |
|  | secret\_text = '' |
|  |  |
|  | if ( lang == 'en' ): |
|  | char\_bit\_len = 7 |
|  | else: |
|  | char\_bit\_len = 8 |
|  |  |
|  | for i in range(char\_bit\_len, len(bit\_string), char\_bit\_len): |
|  | if ((bit\_string[i-char\_bit\_len:i]) == '0000000'): |
|  | break |
|  | secret\_text += chr(self.bit\_str\_to\_int(bit\_string[i-char\_bit\_len:i])) |
|  |  |
|  | return secret\_text |