Systems Security

Final Assignment

Spring 2019, Semester 2

Atte Karppinen
D18123298

Vigenère and RSA

Python is a popular script programming language for cryptology purposes because of its easy readability and vast libraries. Following ciphers are written in Python.

Running **Vigenère cipher** fires main method:

```
def main():
    key_word = "KISWAHILI"
    # Open text file and save it in string variable
    with open("encodedVigenere.txt", "r") as input_file:
        ciphertext = input_file.read()

    key = generate_key(ciphertext, key_word)
    print(decipher_message(ciphertext, key))
```

Which then opens separate file and reads the input into a string variable called ciphertext. After that generate_key method is called:

```
def generate_key(ciphertext, key_word):
    key = ""
    # Key gets splitted so each individual letter can be processed more easily
    key_splitted = list(key_word)
    # list_value is needed to match key length to ciphertext's
    list_value = 0

    for symbol in ciphertext:
        if symbol in alphabet:
            key = key + key_splitted[list_value]
            list_value += 1
            if list_value >= len(key_word):
                list_value = 0
        else:
            key = key + symbol
    return key
```

Each character is then processed in a loop. If character does not belong to English alphabet, it is just added to deciphered text (such as ‘, “ and whitespace). List_value increments on each loop, going key word repeatedly over. Because the first loop is the length of original

ciphertext, key will be the same length. Key is just keyword repeating. After key is generated, decipher method is called from the main method and printed to command line:

```
def decipher_message(ciphertext, key):
    plaintext = ""
    # Key gets splitted so each individual letter can be processed more easily
    key_splitted = list(key)

    for index, symbol in enumerate(ciphertext):
        if symbol in alphabet:
            # Capital ASCII values range from 65 to 90
            # +/-65 is required to count the
            # deciphered letter's numerical value
            # (Used formula is explained above the code)
            letter_value = (ord(symbol) - 65) - (ord(key_splitted[index]) -65)
            if letter_value < 0:
                letter_value += len(alphabet) + 65
                plaintext_character = chr(letter_value)
            else:
                letter_value += 65
                plaintext_character = chr(letter_value)

            plaintext = plaintext + plaintext_character
        else:
            plaintext = plaintext + symbol

    return plaintext
```

RSA cipher starts from main method as well. Admittedly, I have no idea how extended Euclidian to find the modular inverse works in the code. Source is referenced above the algorithm. I try to comment as much possible, so I do not have to explain so much here. Main method:

```
if __name__ == "__main__":
    return_values   = initialization()
    rsa_modulus     = return_values[0]
    derived_number  = return_values[1]
    private_key     = return_values[2]
    ciphertext      = encrypt('Testy test', rsa_modulus, derived_number)
    decrypt(ciphertext, private_key, rsa_modulus)
```

Initialization method where modulus, derived number and private key are calculated:

```
def initialization():
    min_int = 1000
    max_int = 2000
    # SystemRandom is a class that uses the os.urandom() function for
    # generating random numbers from sources provided by the operating system.
```

```
    # Suitable for cryptographic reasons
    prime1 = SystemRandom().randint(min_int, max_int)
    prime2 = SystemRandom().randint(min_int, max_int)

    while not is_prime_number(prime1):
        prime1 = SystemRandom().randint(min_int, max_int)

    while not is_prime_number(prime2):
        prime2 = SystemRandom().randint(min_int, max_int)

    rsa_modulus = prime1 * prime2

    # Φ(n)
    totient = (prime1 - 1) * (prime2 - 1)

    # 1 < Derived Number (e) < Φ(n) (totient) and gcd(e, Φ) = 1
    # e.d=1 mod ø(n) and 0≤d≤n BUT accept only positive private key!
    derived_number = SystemRandom().randint(min_int, totient)
    private_key = extended_gcd(derived_number, totient)
    while (not gcd_is_1(derived_number, totient)) or (not
is_prime_number(derived_number)) or private_key < 0:
        derived_number = SystemRandom().randint(min_int, totient)
        private_key = extended_gcd(derived_number, totient)

    return rsa_modulus, derived_number, private_key
```

Helper scripts to check prime numbers and to use Euclidian algorithms:

```
def is_prime_number(number):
    # An integer greater than one is called a prime number if its only
positive divisors (factors) are one and itself.
    if number > 1:
        # Check if number is divisible by any smaller integer
        for x in range(2, number):
            if number % x == 0:
                return False
        return True

    else:
        return False
```

```
def gcd_is_1(derived_number, totient):

    while(totient):
       derived_number, totient = totient, derived_number % totient

    if derived_number == 1:
        return True
```

```
    else:
        return False

# Modified from this:
# https://crypto.stackexchange.com/a/19530
# (Check in main code that private key is positive)
def extended_gcd(derived_number, totient):
    private_key, y, u, v = 0, 1, 1, 0
    while derived_number != 0:
        q, r = totient//derived_number, totient%derived_number
        m, n = private_key-u*q, y-v*q
        totient, derived_number, private_key, y, u, v = derived_number, r, u,
v, m, n
    return private_key
```

And finally, encryption and decryption methods:

```
def encrypt(plaintext, rsa_modulus, derived_number):
    ciphertext = []
    for symbol in plaintext:
        # Cipher = m^e mod n
        cipher_symbol = ord(symbol)**derived_number % rsa_modulus
        ciphertext.append(cipher_symbol)

    print('Ciphertext: ', ciphertext)
    return ciphertext


def decrypt(ciphertext, private_key, rsa_modulus):
    plaintext = ''
    for symbol in ciphertext:
        # Decipher C^d mod n
        plaintext_symbol = int(symbol ** private_key % rsa_modulus)
        plaintext_symbol = chr(plaintext_symbol)
        plaintext += plaintext_symbol

    print('Plaintext ', plaintext)
```

Vigenère cipher was possible to solve without any aid from internet but implementing RSA without libraries was a difficult task. Euclidian algorithms are mathematically possible to understand but repeating them in code was so complex that I had to search forums for the solution. Python's SystemRandom class provides secure random numbers which can be used for cryptographic purposes.

Cryptographic Tools

Confirming that used numbers are valid for cryptographic purposes using **Sagemath**. Printed numbers used in one example run in RSA algorithm:
Prime numbers p and q = 179 and 193,
Totient (Phi) = 34176
Modulus (N) = 34547
Derived number (e) = 21647
Private key (d) = 9839
Ciphertext: [12846, 30022, 19932, 25843, 10162, 3480, 25843, 30022, 19932, 25843]
Plaintext: Testy test

```
sage: is_prime(179)          True
sage: is_prime(193)          True
sage: gcd(21647, 34176)      1      // gcd(e, phi) should produce 1
sage: 21647 < 34176          True   // e < n (phi)

From https://doc.sagemath.org/html/en/thematic_tutorials/numtheory_rsa.html:
“Having computed a value for d, we then use the command mod(d*e, phi) to check that d*e is
indeed congruent to 1 modulo phi.”

sage: n = 34547
sage: e = 21647
sage: phi = 34176
sage: bezout = xgcd(e, phi); bezout
(1, 9839, -6232)
sage: d = Integer(mod(bezout[1], phi)) ; d
9839
sage: mod(d * e, phi)        1,
```

which it is. And so, same result can be computed using Sagemath’s calculation as when using my own formulas.

Security Monitoring and Vulnerability

Nagios is a tool for continuous monitoring. Be it either software or hardware. It is usually run on a server that is a replica of the development environment and so the release is immediately tested for glitches and bugs. Next is vulnerabilities that Nagios can identify

1. FREAK Vulnerability. OpenSSL vulnerability allowing attackers to intercept HTTPS connections and make the use cryptography that can be broken or even altered (Nagios, 2019).
2. Heartbleed Bug. OpenSSL vulnerability as well. Allows stealing normally protected data. System's memory can be read by anyone on the internet, compromising secret keys and thus compromising everything. Attacker can steal or impersonate a legit user (Synopsys, 2019).
3. Domain Name System Security Extensions (DNSSEC). Nagios provides a plugin to check that DNSSEC is up-to-date and does not include expired signatures. Otherwise DDoS attackers could use DNSSEC reflection attack, changing 80-byte query into a 2313-byte response resulting in data breaches (Henrique, 2019).
4. Various web vulnerabilities such as poorly configured cookies, exposed session ID and parameters passed with form. Anyone monitoring traffic could steal these secrets.
5. Poodle Attack. SSL 3.0 systems that use cipher-block ciphering may be vulnerable to this attack. Padding Oracle On Downgraded Legacy Encryption (POODLE) enables data decryption and extraction from inside an encrypted transaction, mostly using web browsers (Us-cert.gov, 2019).
6. Elevated permissions. Nagios checks logs for any elevated permissions which attacker might have made. This means that system has already been breached and backdoor has been created (Exchange.nagios.org, 2019).
7. HTTP redirections. Nagios checks headers for any given redirects. Attacker might redirect user to a copy of the actual page or just inserting his own spyware or using other Man-in-the-Middle attack methods.
8. Check open ports. Monitoring open ports is essential, since any unauthorized origin may be using and communicating with server if open port exists. Enables multiple exploits.
9. Check new users created in Linux systems. Same as elevated permissions, attacker could possibly make a new super user and control the whole system if new user is made.
10. Blacklist IPs. Nagios can monitor incoming traffic and check whether IPs are blacklisted (banned). This might prevent Distributed Denial of Service (DDoS) attacks. DDoS creates enough traffic against a server, crashing it and maybe enables data stealing due to stack overflow.

Kali Linux

1. **Curl ifconfig.me** for my public IP, which is 37.228.230.XXX (last bit removed for security reasons). Using just **ifconfig**, the internal wireless IP is 10.0.2.15. Wireless connection info is presented after eth0, and if using wireless that info (IP included) comes after wlan0 tag.
2. Command **nslookup <hostname>** reveals info about host. For example, **nslookup Microsoft.com** returns multiple IP addresses that belong to Microsoft.com.
3. Nslookup works the other way as well**. Nslookup 8.8.8.8** (well-known Google DNS IP address) returns name=google-public-dns-a.google.com.
4. Moving or copying file from Linux machines is easy enough. Problems may arise if you try to move to Windows machine (not native SSH). So, first download SSH server from windows options (Apps->Manage optional features), then enable Windows developer mode, make machine discoverable and know your username and password (not pin) for the windows machine. From the Kali's side it does not matter where file is located. Use this in command line: **scp path/to/local/file username@windows_machine_ip:path/to/destination** (like C:/temp). After that your password is required and that's it.
5. Nmap can be used to check the host completely or only specific ports with -p flag. For example, **nmap <host>** (which I replaced with my windows machine IP) yields three ports, their state (only showing OPEN ports) and the service that is using the port. Now, **nmap -p 22 <host>** yields only that one port, state (filtered) and service which is ssh.
6. Like stated in previous answer, without specifying ports, nmap returns all open ports that have some services listening on them.
7. Wget is a tool used to fetch web pages and saving offline versions of them. **Wget https://bozicb.github.io/lectures/SECURITY/** saves offline version to the directory command line is currently in.
8. If given the -S flag when using wget, it downloads the file and prints HTTP headers in command line. So, **wget -S <host>**. These can be then saved into a different file for example.
9. Curl can be used for the same purpose as wget but viewing headers one's own machine sends is easier using curl. Command for that: **curl -v -H "" https://bozicb.github.io/lectures/SECURITY/.** -v means verbose and it prints what happens "under the hood". -H is for extra headers, the ones in request. The request headers were: GET /lectures/SECURITY/ HTTP/2, Host, User-Agent and Accept.

Kali Linux is a Linux distro designed mainly for penetration testing. It automatically includes numerous tools used by security professionals and even tools that anti-virus programs dislike (for obvious reasons). Every other Linux distro could be modified to be like Kali, but Kali is just a ready package easy to install and to start to use. Because of its vast selection of automation and testing tools, it's a popular choice.

Discussion

Personally, I am interested in cybersecurity, but besides the first labs about cryptography, the labs didn't offer much. One-hour lab is not nearly enough to learn anything about the topic and lab assignments were too vague most of the time. Maybe in future, go through the tools and services during lectures (using and testing/doing demos in front of the class), that might make the module easier to understand. And learning curve with tools like these is extremely steep, so I don't imagine many have the interest to learn them on their own time.

Now that small critic is out of the way, I enjoyed this final assignment. It had clear goals (except I have no idea how to test vigenere with sagemath) and just challenging enough tasks. At least for me the learning happens when I get to do something. So again, a suggestion for the future: if possible, include metasploitable in the module. Two virtual machines running (Kali and Metasploitable) and crack and study Metasploitable with Kali.

I have read a few books about cyber security, but it is challenging topic and that's why I had hopes of a more hands-on approach. My general idea and opinion towards security did not change much, but there were a lot of new information.

References

Nagios. (2019). FREAK Vulnerability Tester (CVE-2015-0204) - Nagios. [online] Available at: https://www.nagios.com/freak-vulnerability-tester/ [Accessed 25 Apr. 2019].

Synopsys, I. (2019). Heartbleed Bug. [online] Heartbleed.com. Available at: http://heartbleed.com/ [Accessed 25 Apr. 2019].

Henrique, R. (2019). check_dnssecurity - Nagios Exchange. [online] Exchange.nagios.org. Available at: https://exchange.nagios.org/directory/Plugins/Security/check_dnssecurity/details [Accessed 25 Apr. 2019].

Us-cert.gov. (2019). SSL 3.0 Protocol Vulnerability and POODLE Attack | US-CERT. [online] Available at: https://www.us-cert.gov/ncas/alerts/TA14-290A [Accessed 25 Apr. 2019].

Exchange.nagios.org. (2019). Security_Group_Changes - Nagios Exchange. [online] Available at: https://exchange.nagios.org/directory/Plugins/Security/Security_Group_Changes/details [Accessed 25 Apr. 2019].