Cybersecurity semplice (per 

davvero)

Rovesti Gabriel

*Cybersecurity semplice (per davvero)*

**Attenzione**

1

Il file non ha alcuna pretesa di correttezza; di fatto, è una riscrittura attenta di appunti, slide, materiale sparso in rete, approfondimenti personali dettagliati al meglio delle mie capacità. Credo comunque che, per scopo didattico e di piacere di imparare (sì, io studio per quello e non solo per l’esame) questo file possa essere utile. Semplice si pone, per davvero ci prova.

Thank me sometimes, it won’t kill you that much.

Gabriel

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Sommario 2 Lezione 0: Introduzione al corso e alle modalità (Conti e Bianchi) ................................................................... 3

Lezione 1 e 2: Panoramica generale (Conti)...................................................................................................... 5 Esercizi Lezione 2........................................................................................................................................... 7 Esercizi Lezione 3......................................................................................................................................... 13

Lezione 4: Strumenti crittografici/Cryptographic tools Pt. 1 (Conti)............................................................... 19 Esercizi Lezione 4......................................................................................................................................... 24 Lezione 5: Strumenti crittografici/Cryptographic tools Pt. 2 (Conti)............................................................... 38 Esercizi Lezione 5......................................................................................................................................... 43 Lezione 6: Autenticazione dell’utente/User authentication (Conti) ............................................................... 51 Esercizi Lezione 6......................................................................................................................................... 55 Lezione 7: Introduzione alle vulnerabilità del web/Introduction to Web Vulnerabilities (Conti)................... 59 Esercizi Lezione 7......................................................................................................................................... 60 Lezione 8: Ingredienti del Web/Ingredients of Web (Conti) ........................................................................... 66 Esercizi Lezione 8......................................................................................................................................... 70 Lezione 9: Language Vulnerabilities/Vulnerabilità del linguaggio (Conti)....................................................... 76 Esercizi Lezione 9......................................................................................................................................... 78 Lezione 10: Injection Attacks (Conti)............................................................................................................... 86 Esercizi Lezione 10....................................................................................................................................... 89 Lezione 11 - Intro to Reverse Engineering (Pier Paolo Tricomi)................................................................... 105 Esercizi Lezione 11..................................................................................................................................... 115 Lezione 12: Patching (Pier Paolo Tricomi) ..................................................................................................... 134 Esercizi Lezione 12..................................................................................................................................... 139 Lezione 14: Debugging (Pier Paolo Tricomi).................................................................................................. 149 Esercizi Lezione 13..................................................................................................................................... 154 Esercizi Lezione 14..................................................................................................................................... 177 Lezione 15: Shellcode (Pier Paolo Tricomi) ................................................................................................... 183 Esercizi Lezione 15..................................................................................................................................... 187 Lezione 16: PLT (Procedure Linkage Table) & GOT (Global Offset Table) – Pier Paolo Tricomi ................... 203 Esercizi Lezione 16..................................................................................................................................... 206 Lezione 17: Return Oriented Programming (ROP) – Pier Paolo Tricomi ....................................................... 214 Esercizi Lezione 17..................................................................................................................................... 217

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Lezione 0: Introduzione al corso e alle modalità 3 (Conti e Bianchi)

Sito di riferimento: https://www.math.unipd.it/~conti/teaching/CPP2223/index.html Professore: Mauro Conti

Assistenti Anno di Corso: Tommaso Bianchi / Pier Paolo Tricorni

Il corso tratta le seguenti tematiche:

● Cryptography

○ Ciphers; hash functions; symmetric/asymmetric encryption

● Web Vulnerabilities

○ Bad programming practices; injections; language vulnerabilities.

● Reverse Engineering

○ Reversing techniques; patching; anti-debug.

● Pawning

○ Buffer overflow; defenses; Return Oriented

Programming; Global Offset Table.

Ogni lezione consiste in ~30' di teoria (inutile all’esame, qui comunque ben affrontata) e ~60' di esercizi (in cui gli assistenti forniscono tutti gli aiuti del caso).

L'esame finale ha tre diversi formati, tra i quali gli studenti possono sceglierne uno

● Esame finale:

○ Serie di esercizi che coprono gli argomenti del corso.

● Tre esercizi pratici:

○ da risolvere solo durante il semestre di corso

● Un progetto di ricerca:

○ possibilmente interagendo anche con i ricercatori del gruppo SPRITZ (Security and Privacy Research Group at University of Padua), presieduto da Conti, aiutando i ricercatori del gruppo in qualche progetto esistente (software/hardware/modulo di qualche progetto, normalmente introdotto tramite Moodle) 🡪 https://spritz.math.unipd.it/

Un esempio di attacco famoso è fatto sul Transport Layer Security (TLS), 

che fornisce autenticazione, privacy e integrità dei dati tra due

applicazioni informatiche comunicanti. È il protocollo di sicurezza più

diffuso oggi ed è il più adatto per i browser web e altre applicazioni che

richiedono lo scambio sicuro di dati in rete. Esso permette di connettersi

in modo sicuro, scambiando certificati digitali per assicurarsi chi sono le

entità che comunicano.

Un attacco man in the middle (MITM) è un termine generale che indica

quando un autore si inserisce in una conversazione tra un utente e

un'applicazione, per origliare o per impersonare una delle parti, facendo

credere che sia in corso un normale scambio di informazioni.

L'obiettivo di un attacco è rubare informazioni personali, come credenziali di accesso, dettagli del conto e numeri di carta di credito. Gli obiettivi sono tipicamente gli utenti di applicazioni finanziarie, aziende SaaS, siti di e-commerce e altri siti web in cui è richiesto il login.

Approfondimento di esempio reale di sicurezza: TOR Browser

https://www.makeuseof.com/how-tor-addresses-work/

https://tor.stackexchange.com/questions/672/how-do-onion-addresses-exactly-work

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

4 Approfondimento: Flightradar24 https://www.flightradar24.com/how-it-works

Problemi di sicurezza a dispositivi fisici:

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7340599/

https://stackoverflow.com/questions/4083860/in-computer-security-what-are-covert-and-side-channels https://www.ictsecuritymagazine.com/articoli/stuxnet-iran-2009-il-case-study-per-disarticolare-la-cyber war/

https://arxiv.org/pdf/2005.07392.pdf

*Cosa significa sicurezza?* Mantenere informazioni di un contesto di interesse difficile da accedere.

1) La sicurezza non è solo "un prodotto" (ad esempio un firewall); è piuttosto un "processo", che deve essere gestito appropriatamente

2) Niente è sicuro al 100% (ne abbiamo bisogno? Quanto costerebbe?)

3) La sicurezza di un sistema equivale alla sicurezza della sua componente meno sicura (regola dell'anello più debole)

4) La sicurezza per oscurità non funziona mai (cryptography by obscurity, nascondere quindi le informazioni)

5) La crittografia è uno strumento potente ma... non basta

6) Non fare affidamento sugli utenti

Proprietà di sicurezza di base

• Riservatezza: per prevenire la divulgazione non autorizzata delle informazioni

• Integrità: per prevenire la modifica non autorizzata delle informazioni

• Disponibilità: per garantire l'accesso alle informazioni

• Autenticazione: a dimostrare l'identità rivendicata può essere l'autenticazione Dati o Entità • Non ripudio: per prevenire la falsa negazione delle azioni eseguite

• Autorizzazione: cosa può fare un utente

• Auditing: per registrare in modo sicuro le prove delle azioni eseguite

• Tolleranza agli attacchi: capacità di fornire un certo grado di servizio dopo guasti o attacchi • Disaster Recovery: possibilità di ripristinare uno stato sicuro

• Key-recovery (è il tentativo di un avversario di recuperare la chiave crittografica di uno schema di cifratura. Normalmente ciò significa che l'attaccante dispone di una o più coppie di messaggi in chiaro e del corrispondente testo cifrato.)

• Key-escrow (tecnica mediante la quale la chiave necessaria a decriptare dei dati criptati è conservata con un acconto di garanzia da terze parti in modo che, in particolari situazioni, possa essere recuperata per avere accesso a quei dati anche se coloro che hanno cifrato i dati non vogliono renderla disponibile)

• Digital Forensics

Esempi di meccanismi di sicurezza sono i seguenti:

• Numeri casuali (ad es. per i vettori di inizializzazione)

• Numeri pseudo casuali

• Crittografia/Decrittografia

• Funzioni hash

• Catena hash (invertita)

• Codice di integrità del messaggio (MIC)

• Codice di autenticazione dei messaggi (MAC e HMAC)

• Firme digitali

• Protocolli di scambio di chiavi (stabilimento)

• Protocolli di distribuzione delle chiavi

• Marcatura temporale

Tipi di attacco:

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

5 • Passivo: l'attaccante può leggere solo qualsiasi informazione – TEMPEST (intelligenza del segnale), che si riferisce allo spionaggio di sistemi informatici attraverso emanazioni, inclusi segnali radio o elettrici, suoni e vibrazioni non intenzionali.

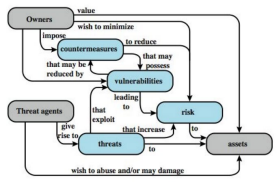
– Sniffing dei pacchetti

• Attivo: l'attaccante può leggere, modificare, generare, distruggere qualsiasi informazione

Lezione 1 e 2: Panoramica generale (Conti)

*Sicurezza informatica*: protezione offerta a un sistema informativo automatizzato

al fine di raggiungere gli obiettivi applicabili di preservare l'integrità, la disponibilità e la riservatezza delle risorse del sistema informativo (include hardware, software, firmware, informazioni/dati software, firmware, informazioni/dati e telecomunicazioni).

*Sfide* 

1. non è semplice

2. bisogna considerare i potenziali attacchi

3. le procedure utilizzate sono controintuitive

4. coinvolgono algoritmi e informazioni segrete

5. bisogna decidere dove impiegare i meccanismi

6. battaglia d'ingegno tra attaccante e

amministratore

7. non si percepiscono i vantaggi finché non si fallisce

8. richiede un monitoraggio regolare

9. troppo spesso un ripensamento

10. considerato un ostacolo all'utilizzo del sistema

Ci possono essere una serie di rischi, tale che cercando di introdurre delle contromisure, si possa in qualche modo avere un problema nel corso del tempo; queste dovranno essere aggiornata.

*Risorsa di sistema*: con le vulnerabilità può:

○ essere corrotta (perdita di integrità)

○ diventare non disponibile (perdita di confidenzialità)

○ diventare non disponibile (perdita di disponibilità)

Gli *attacchi* sono minacce portate avanti e possono essere

○ passivi

○ attivi

○ insider, problemi interni ad un’organizzazione

○ outsider, tutto ciò che viene da fuori, esterno a noi

Occorre adottare delle *contromisure*, quindi i mezzi utilizzati per affrontare gli attacchi alla sicurezza basati su tre principi cardine:

○ prevenire (cercando di evitare che gli attacchi esistano)

○ rilevare

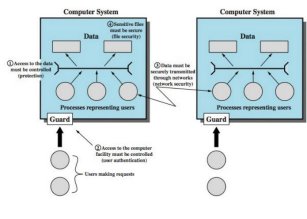
○ recuperare

Questo può portare a nuove vulnerabilità ed avrà una vulnerabilità residua

L'obiettivo è quello di minimizzare il rischio, dati i vincoli.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

6 Possono essere collegati dei problemi, quali: ● divulgazione non autorizzata 

- esposizione, intercettazione,

inferenza, intrusione

● inganno/deception

- mascherata, falsificazione, ripudio

● distruzione

- incapacità, corruzione, ostruzione

●usurpazione

- appropriazione indebita, uso

improprio

Si cerca di controllare l’accesso e

l’identificazione alla risorsa, tale che non

succeda nulla di inaspettato tra i *peers*, sia da

un punto di vista software (injection,

eavesdropping, ecc.) che hardware

(compromettendo fisicamente la macchina)

Similmente, possono esserci degli attacchi relativi alla sicurezza della rete, che possono essere classificati come passivi o attivi.

Gli attacchi *passivi* sono costituiti da intercettazioni (eavesdropping):

○ la divulgazione del contenuto dei messaggi

○ l'analisi del traffico

○ sono difficili da rilevare, quindi si cerca di prevenire

Gli attacchi *attivi* modificano/falsificano i dati:

○ masquerade 🡪 (Un attacco mascherato è un attacco che utilizza una falsa identità, ad esempio un'identità di rete, per ottenere un accesso non autorizzato alle informazioni del computer personale attraverso un'identificazione di accesso legittima. Se un processo di autorizzazione non è completamente protetto, può diventare estremamente vulnerabile a un attacco masquerade) ○ replay 🡪 (Un attacco replay si verifica quando un criminale informatico origlia una

comunicazione di rete sicura, la intercetta e poi la ritarda o la reinvia in modo fraudolento per indurre il destinatario a fare ciò che l'hacker desidera)

○ modifica

○ Denial of Service (DoS) 🡪 (Un attacco denial-of-service (DoS) si verifica quando gli utenti legittimi non sono in grado di accedere a sistemi informativi, dispositivi o altre risorse di rete a causa delle azioni di un attore malintenzionato della minaccia informatica.

○ Difficile da prevenire, quindi da rilevare

Ci sono anche dei requisiti funzionali di sicurezza, quali:

- Misure tecniche:

○ controllo degli accessi; identificazione e autenticazione; protezione del sistema e delle comunicazioni; integrità del sistema e delle integrità delle informazioni

- Controlli e procedure di gestione

○ consapevolezza e formazione; audit e responsabilità; certificazione, accreditamento e valutazioni di sicurezza; valutazioni di sicurezza; pianificazione di emergenza; manutenzione; protezione fisica e ambientale; pianificazione; sicurezza del personale; valutazione del rischio ; acquisizione di sistemi e servizi

- Sovrapposizione di competenze tecniche e gestionali:

○ gestione della configurazione; risposta agli incidenti; protezione dei media

La strategia da un punto di vista di sicurezza si basa su alcuni punti:

● Specificazione/politica (policies)

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

○ Cosa dovrebbe fare lo schema di sicurezza?

7

○ Codificare in politiche e procedure

● Attuazione/meccanismi (implementazioni delle politiche e verificare siano rispettate) ○ Come si attua il meccanismo di sicurezza?

○ Prevenzione, rilevamento, risposta, recupero

● Correttezza/assicurazione

○ Funziona davvero?

○ Garanzia, valutazione

Viene raccomandato Ubuntu o comunque una distribuzione Linux per fare il corso.

Ad ogni lezione vengono date le challenges, sfide nel trovare le *flag* (CTF), codice/programma con vulnerabilità di sicurezza per cui, sfruttandola, si ottiene una flag.

Ciascun set di esercizi si concentra su una specifica vulnerabilità di sicurezza, con diversi livelli di sicurezza; comunque, ci sono gli assistenti.

Per cui:

1) viene data una challenge

2) si identifica l’area di applicazione

3) si definisce un set di vulnerabilità adatte

4) si impara anche da Google come fare

5) si ripete finché non si risolve

a. spesso ci sono vari modi di risolvere un problema

Esercizi Lezione 2

*Piccolo disclaimer* 🡪 Essendo che gli esercizi sono fatti in Python 3, magari nei lab esiste Python 2. Per poter eseguire un programma, normalmente da linea di comando si ha *python (file.py)*. Per attivare l’esecuzione sicura con Python 3, si ha il comando *python3 (file.py)*.

1) Esercizio 1: Testo e soluzione

Testo

*You are asked to define a Python code that, given a string,*

*prints a string with all the letters shifted by 2.*

*For example,*

*input = 'abc'*

*output = 'cde'*

Soluzione

*input = 'abc'*

*key = 2*

*#solution 1*

*output1 = ' ' #copy*

*for i in range(len(input)):*

*output1 += chr(ord(input[i]) + key)*

*print(output1)*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

8

*#solution 2*

*output2 = ' '*

*for c in input:*

*output2 += chr(ord(c) + key)*

*print(output2)*

*#solution 3*

*output3 = ' '.join([chr(ord(c) + key) for c in input])*

*print(output3)*

Si noti la scrittura equivalente a titolo di chiarezza:

[str(i) for i in mybin] 🡪 Crea una lista in cui itera su ogni elemento *str(i)* dentro *mybin*

.join((str(i) for i in mybin)) 🡪 Esegue il join della stringa *str(i)* direttamente dentro *mybin*

La scrittura del tipo newlist = [*expression* for *item* in *iterable* if *condition* == True] viene chiamata in Python come *list comprehension*, ed offre una sintassi breve basata su una lista esistente. Alternativamente, la sintassi può essere anche come *expression – condition – for loop*.

Note:

- La funzione *chr()* di Python accetta un argomento intero e restituisce la stringa che rappresenta un carattere in quel punto di codice. Poiché la funzione *chr()* prende un argomento intero e lo converte in carattere, esiste un intervallo valido per l'input

- La funzione *ord()* restituisce il numero che rappresenta il codice Unicode di un carattere specificato. - Il metodo *join()* prende tutti gli elementi di un oggetto iterabile e li unisce in una stringa. È necessario specificare una stringa come separatore.

2) Esercizio 2: Testo e soluzione

Testo

*Define a simple calculator.*

*The user uses the terminal, and it has three variables.*

*- input1: first integer number*

*- input2: second integer number*

*- type of operation: a number associated to the operation*

*(e.g., 0 for the addition)*

Soluzione

*import argparse #parsing arguments*

*parser = argparse.ArgumentParser()*

*parser.add\_argument("--x1", type = int) # adding the first two arguments and the operation to the parser parser.add\_argument("--x2", type = int)*

*parser.add\_argument("--y", type = int)*

*args = parser.parse\_args() #parsing all together*

*def main():*

*#get the arguments*

*x1 = args.x1*

*x2 = args.x2*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

9 *operation = args.y*

*#adding f before the " allows you to insert variables*

*#inside the string*

*print(f"x1={x1}\tx2={x2}")*

*#Interpret the operation*

*if operation == 0:*

*print(f"Addition={x1 + x2}")*

*elif operation == 1:*

*print(f"Subtraction={x1 - x2}")*

*elif operation == 2:*

*print(f"Multiplication={x1 \* x2}")*

*if operation == 3:*

*print(f"Division={x1 / x2}")*

*if \_\_name\_\_ == '\_\_main\_\_':*

*#usage example: python solution.py --x1 1 --x2 2 --y 3*

*main()*

Altra possibile soluzione:

*import argparse*

*import sys*

*parser = argparse.ArgumentParser(description='Process two inputs and use an operation') parser.add\_argument("--input1", type=int, help="first integer number")*

*parser.add\_argument("--input2", type=int, help="second integer number")*

*operation = parser.add\_argument("--operation", type=int, help="operation to be performed: 1 for addition, 2 for subtraction, 3 for multiplication, 4 for division")*

*args = parser.parse\_args()*

*if args.operation == 1:*

*print(args.input1+args.input2)*

*elif args.operation == 2:*

*print(args.input1-args.input2)*

*elif args.operation == 3:*

*print(args.input1\*args.input2)*

*elif args.operation == 4:*

*print(args.input1/args.input2)*

*else:*

*print("Invalid operation")*

*# execution command in terminal*

*# python ex2.py --input1 1 --input2 2 --operation 3*

Note:

- Il modulo *argparse* semplifica la scrittura di interfacce a riga di comando di facile utilizzo. Il programma definisce gli argomenti che richiede e argparse si occuperà di analizzare tali argomenti da *sys.argv*. Inoltre, il modulo argparse genera automaticamente messaggi di aiuto e di utilizzo ed emette errori quando l'utente fornisce al programma argomenti non validi.

- Questo serve a definire chiaramente le variabili utilizzate in un array *args* tale da predefinirsi i parametri che si devono utilizzare

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

10

3) Esercizio 3: Testo e soluzione

Testo

*Define a randomic password generator.*

*The password should contain 10 characters.*

*Type of characters: alphanumeric*

Soluzione

*import random*

*import string*

*# define the vocabulary*

*vocabulary = list(string.ascii\_letters) + list(string.digits)*

*# if you uncomment the following line, you allow the reproducibility (random characters) #random.seed(123)*

*# set final variable*

*password = ' '*

*# define number of iterations*

*password\_size = 10*

*for i in range(password\_size):*

*# shuffle the vocabulary*

*random.shuffle(vocabulary)*

*# update the password. We take the first element of the vector (at index 0, as we know) password += vocabulary[0]*

*print(f"Password generated=\t{password}") #\t stays for “tab”, as you might have guessed* Altra soluzione

*import random*

*import string*

*def password\_generator():*

*password = ""*

*for i in range(10):*

*password += random.choice(string.ascii\_letters + string.digits)*

*return password*

*print(password\_generator())*

Note:

- Il metodo *seed()* serve a inizializzare il generatore di numeri casuali. Il generatore di numeri casuali ha bisogno di un numero da cui partire (un valore di seme), per poter generare un numero casuale. - Importiamo *string* per prenderci tutti i caratteri Ascii e i numeri

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

11

21/10/2022 - Lezione 3: Codifica/Encoding (Conti)

La codifica delle informazioni è fondamentale, in maniera tale da poter standardizzare i dati e poter con essi interagire. Perché codificare i dati?

● Trasformare i dati in modo che possano essere consumati correttamente (e in modo sicuro) da un altro tipo di sistema.

● L'obiettivo non è quello di mantenere segrete le informazioni. Ad esempio, le chiavi non sono necessarie. ● La codifica può essere facilmente invertita. È facile riconoscere una strategia di codifica. ● La decodifica è l'operazione inversa.

Non parliamo di codifica, ma di criptaggio (encoding and encryption):

- Il criptaggio intende nascondere le informazioni, affinché nessun altro capisca (intercettando o “in the middle”) cosa si stia trasmettendo

- La codifica, invece, non intende nascondere le informazioni, in quanto sono sempre visibili; intende semplicemente una differente interpretazione

Alcuni esempi di codifica:

● base64 🡪 è un sistema di codifica che consente la traduzione di dati binari in stringhe di testo ASCII, rappresentando i dati sulla base di 64 caratteri ASCII diversi.

- Molto tipico del web in cui la lunghezza dei messaggi finali è *sempre* un multiplo di 4 (tale che, in un messaggio, si ricavino sempre 4 byte in vari modi)

- Possiede un alfabeto unico:

○ [A-Z, a-z, 0-9, +, /, =]

○ 0 = A, 1 = B, …, 26 = a, 27 = b, … 

○ 52 = 0, 53 = 1, …, 62 = +, 63 = /

- Padding

○ Spesso finisce con “==” oppure con “=”

Some examples are:

● pleasure -> cGxlYXN1cmU=

● leasure -> bGVhc3VyZQ==

● easure -> ZWFzdXJl

● asure -> YXN1cmU=

● sure -> c3VyZQ==

Il meccanismo Base64 utilizza 64 caratteri per la codifica. Questi caratteri sono costituiti da: - 10 valori numerici: cioè 0,1,2,3,...,9

- 26 alfabeti maiuscoli: A,B,C,D,...,Z

- 26 alfabeti minuscoli: a,b,c,d,...,z

- 2 caratteri speciali (questi caratteri dipendono dal sistema operativo): ad esempio +,/

I passaggi per codificare una stringa con l'algoritmo base64 sono i seguenti:

- Contare il numero di caratteri di una stringa. Se non è un multiplo di 3, allora lo si riempie con caratteri speciali (ad es. =) per renderlo multiplo di 3.

o Infatti, il segno di uguale viene messo per inserire un padding speciale di caratteri per rendere una stringa multiplo di 3 e poi renderla stampabile con 4 caratteri stampabili in ASCII. Spesso il segno è alla fine, ma non sempre.

- Convertire la stringa in formato binario ASCII a 8 bit utilizzando la tabella ASCII.

- Dopo la conversione in formato binario, dividere i dati binari in pezzi da 6 bit.

- Convertire i pezzi di dati binari a 6 bit in numeri decimali.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

12 - Convertire i decimali in stringhe secondo la tabella degli indici base64. Questa tabella può essere un esempio, ma come ho detto, 2 caratteri speciali possono variare.

- Ora abbiamo la versione codificata della stringa di input.

Facciamo un esempio: convertire la stringa THS in stringa con codifica base64.

- Conta il numero di caratteri: è già un multiplo di 3.

- Convertire in formato binario ASCII a 8 bit. Abbiamo ottenuto (T)01010100 (H)01001000 (S)01010011

- Dividere i dati binari in pezzi da 6 bit. Abbiamo ottenuto 010101 000100 100001 010011 - Convertire i dati binari a 6 bit in numeri decimali: abbiamo ottenuto 21 4 33 19.

- Convertire i decimali in stringhe secondo la tabella degli indici base64. Abbiamo ottenuto VEhT

Esempio importante: https://www.base64encode.org/

Si vede che, prendendo per esempio una stringa come *pleasure*, togliendo l’ultima lettera, viene aggiunta nella codifica a Base64 il segno “=” quando la stringa non è in formato multiplo di 3, trattandola quindi in suddivisione come multiplo di 4.

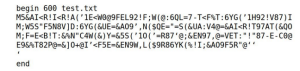
Suggerisco 🡪 *pleasur – pleasu – pleas – plea*

● Esadecimale 🡪 versione semplificata del binario che raggruppa cifre, è un sistema numerico posizionale in base 16, cioè che utilizza 16 simboli invece dei 10 del sistema numerico decimale tradizionale. Esso è simile a base64, con un alfabeto formato da [A-F, 0-9].

È molto usato per rappresentare indirizzi MAC (quelli univoci della scheda di rete di un dispositivo per identificarlo) e i dump di memoria (processo che consiste nel prendere tutte le informazioni contenute nella RAM e scriverle su un'unità di archiviazione. Gli sviluppatori usano comunemente i dump della memoria per raccogliere informazioni diagnostiche al momento di un arresto anomalo, per aiutarli a risolvere i problemi e a saperne di più sull'evento).

● Uuencoding 🡪 codifica da binario a testo per UNIX per i sistemi di e-mail

Essendo UNIX, inizia *sempre* con la keyword *begin* seguito dal *mode* (che specificano le modalità di accesso ai file tramite dei numeri, es. 600 permette di modificare un file seguito da *chmod*) e finisce *sempre* con la coppia ‘ (singolo apice/apostrofo) ed *end*.



Quando si analizzano i dati, questi potrebbero essere rappresentati in una codifica sconosciuta. Come identificare la codifica corretta?

○ esperienza (si riconosce praticandone spesso)

○ alfabeto (sulla base delle osservazioni precedenti)

○ pattern (stringhe ripetute o qualcosa che si ripete in modo diverso dal normale)

○ origine dei dati (in base alla provenienza, si può intuire il contesto)

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Esercizi Lezione 3 13

1) Valley of Fear: Testo, aiuti e soluzione

Testo

*The hard drive may be corrupted, but you were able to recover a small chunk of text (see "book.txt"). Scribbled on the back of the hard drive is a set of mysterious numbers. Can you discover the meaning behind these numbers? (1, 9, 4) (4, 2, 8) (4, 8, 3) (7, 1, 5) (8, 10, 1)*

Viene dato quindi un file txt “book.txt”, generico file di testo con una serie di parole tratte da un libro. Da questa lezione iniziano ad essere inclusi degli “hints”, quindi degli aiuti/suggerimenti per cercare di trovare la soluzione.

Aiuti:

*1)* How are poems and books organized? Then, have a look at how the challenge is “organized”. This should help you to understand the meaning of the triplet *<num1, num2, num3>*

2) There is a mapping between each number triplet position and the “text organization”.

Soluzione

Il suggerimento ci dice che ognuno di questi tre numeri potrebbe rappresentare una parola in un messaggio. Ci sono 9 paragrafi in *book.txt* con un massimo di 17 righe. L'intuizione è che ogni tripla utilizzi il seguente schema per decodificare una parola:

*(1, 9, 4) = (paragrafo, riga, parola)*

Seguendo l'intuizione di cui sopra, otteniamo:

(1, 9, 4) === the

(4, 2, 8) === flag

(4, 8, 3) === is

(7, 1, 5) === Ceremonial

(8, 10, 1) === plates

La flag è: “The flag is Ceremonial plates.”

Scrivendo una soluzione sotto forma di script Python:

#The hard drive may be corrupted, but you were able to recover a small chunk of text (see "book.txt"). #Scribbled on the back of the hard drive is a set of mysterious numbers. Can you discover the meaning behind these numbers? (1, 9, 4) (4, 2, 8) (4, 8, 3) (7, 1, 5) (8, 10, 1)

*keys = [(1, 9, 4), (4, 2, 8), (4, 8, 3), (7, 1, 5), (8, 10, 1)]*

*with open("book.txt") as book:*

*text = book.read()* #read the content

*paragraphs = [p.split("\n") for p in text.split("\n\n")]* #split the content into paragraphs and breaking the lines

#saving the words from paragraph into a list

#using the lambda function to define inline

#a function to search inside the paragraphs and splitting the spaces in between

*words = list(map(lambda p: list(map(lambda s: s.split(" "), p)), paragraphs))*

#split the paragraphs into sentences

*flag = " ".join(words[key[0] - 1][key[1] - 1][key[2] - 1] for key in keys)*

#get the words from the keys array [i-1] to avoid going out of bounds

*print(flag)*

2) Sherlock: Testo, aiuti e soluzione

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

14 Testo *Sherlock has a mystery in front of him. Help him to find the flag.*

Viene dato anche qui un file txt “challenge.txt” contenente l’intero libro “The Adventures of Sherlock Holmes” in versione txt del 1999, codificato in Ascii.

Aiuti:

1) This is a longer text. In CTF, and in general, in security assessment, we need to carefully look at everything! As you might notice (?), there are some upper characters. These upper characters compose a hidden message. Try to retrieve them, and then print them!

2) You retrieved a secret message (if not, you need to use the previous hint). The message is a binary message: maybe if you convert it in proper 8-digit size binary numbers (e.g., 10010000 10011010), you can then convert it -- somehow (?) -- to a proper ASCII message.

Soluzione

La prima cosa da fare è notare che ci sono alcuni caratteri maiuscoli; quindi, possiamo aprirli e filtrarli (si può usare un qualsiasi editor di testo per farlo, loro suggeriscono Python usando il codice che segue)

*import os*

# open the text file and read its content

*with open('challenge.txt', 'r') as file:*

*challenge = file.read()*

# take the uppercase letters

*insight=' '.join([c for c in challenge if c.isupper()])* # we filter them and take only the upper ones *print(insight)*

L'output è una stringa con una serie di "ZERO" e "ONE". Possiamo prima convertirli nella loro rappresentazione numerica.

*insight = insight.replace('ZERO', '0')*

*insight = insight.replace('ONE', '1')*

Possiamo poi osservare la lunghezza di questa nuova stringa: è un multiplo di 8. La nuova intuizione è che questa stringa rappresenta una serie di caratteri Unicode scritti in binario. È sufficiente invertire il processo, riconvertendo il tutto in caratteri unicode (da cui le moltiplicazioni per 8 con il 2) e ciclando iterativamente, quindi stampando.

*result=' '.join(chr(int(insight[i\*8:i\*8+8],2)) for i in range(len(insight)//8))*

#here, we do cycle iterating the text, flooring it by 8 characters and taking each group of 8 bytes as binary *print(result)*

E la flag viene trovata:

*BITSCTF{h1d3\_1n\_pl41n\_5173*

Altra soluzione:

*s=' '*

*with open('challenge.txt', 'r') as file:*

*challenge = file.read()*

*for i in challenge:*

#Checking if the character is a letter

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

15 *if i.isupper():* #there are some uppercase letters, so we take them out *s=s+i*

*s=s.replace('ZERO', '0') .replace('ONE', '1')* #replacing ZERO and ONE with 0 and 1

#print(s) #here we do have a string printed like '010000100100...'

#the idea is having the string 's' and taking each 8 bytes as one and converting them into letters #so, we must split the string into 8 bytes chunks, then convert them into letters

#and finally print the flag

*for i in range(0, len(s), 8):*

#the idea is iterating starting from '0', going up until the end of the string and taking each 8 bytes as one #chr() function is used to get a string representing of a character which points to a Unicode code integer. #then we go ahead up until the last ' ' (so, it ends at the last space character)

*print(chr(int(s[i:i+8], 2)), end='')* #2 is for binary and “end” is for formatting after each character #BITSCTF{h1d3\_1n\_pl41n\_5173}

3) Ultraencoded: Testo, aiuti e soluzione

Testo

*Fady didn't understand well the difference between encryption and encoding,*

*so instead of encrypting some secret message to pass to his friend, he*

*encoded it!*

*The flag should be in the format: ALEXCTF*

Viene dato un file senza estensione chiamato “zero\_one” composto solamente da molte stringhe sequenziali ZERO ONE ZERO ONE….

Aiuti:

1) The challenge here is to understand in which type of encoding is the message. You should see it as an onion, there are multiple encoding strategies on top of previous encoding steps! You should reverse the encoding path.

Soluzione

Possiamo iniziare aprendo il file e convertendo le stringhe nel loro formato numerico, come nell'esercizio precedente. Si noti che qui ci sono spazi aggiuntivi da rimuovere.

# Aprire il file

*with open('zero\_one', 'r') as file:*

*input = file.read()*

# sostituire gli zeri e gli uni nella rappresentazione numerica

*input = input.replace('ZERO', '0')*

*input = input.replace('UNO', '1')*

*input = input.replace(' ', '') # eliminare gli spazi aggiuntivi*

# eliminare gli spazi prima e dopo l’inizio della stringa

*input = input.strip()*

Seguendo i passi dell'esercizio due, convertiamo la stringa dalla rappresentazione binaria in Ascii.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

*result=' '.join(chr(int(input[i\*8:i\*8+8],2)) for i in range(len(input)//8))*

16

Si può notare che ci sono alcuni caratteri strani: è il codice MORSE. Si va a definire quindi una struttura *dict/dizionario* in Python, implementato come la struttura dati *map*, quindi con una serie di coppie chiave valore. La struttura generica è così composta:

*d = {*

*<key>: <value>,*

*<key>: <value>,*

*.*

*.*

*.*

*<key>: <value>*

*}*

Prima, però, a seguito della decodifica e della stampa di *result*, si può notare che gli ultimi caratteri sono “==” e quindi, eseguo una decodifica da base64 ad Ascii come segue:

*import base64*

*decoded = base64.b64decode(result).decode('ascii')*

*print(decoded)*

Mi definisco quindi una map per il codice MORSE come segue:

# the output is a morse code

*alpha2morse = {'A': '.-', 'B': '-...', 'C': '-.-.',*

*'D': '-..', 'E': '.', 'F': '..-.',*

*'G': '--.', 'H': '....', 'I': '..',*

*'J': '.---', 'K': '-.-', 'L': '.-..',*

*'M': '--', 'N': '-.', 'O': '---',*

*'P': '.--.', 'Q': '--.-', 'R': '.-.',*

*'S': '...', 'T': '-', 'U': '..-',*

*'V': '...-', 'W': '.--', 'X': '-..-',*

*'Y': '-.--', 'Z': '--..',*

*'0': '-----', '1': '.----', '2': '..---',*

*'3': '...--', '4': '....-', '5': '.....',*

*'6': '-....', '7': '--...', '8': '---..',*

*'9': '----.' }*

A questo punto, faccio un reverse sulle coppie chiave-valore degli elementi della mappa e poi converto il Morse a stringa, stampando:

#reversing morse map 🡪 we use a dictionary (which uses a key-value mapping and the {}, curly brackets) #iterating the reverse (key/value instead of value/key inside all the Morse map items)

*morse2alpha = {value:key for key,value in alpha2morse.items()}*

#convert morse to string 🡪 we get the ith character of the corresponding string word and then print it *decoded2 = ' '.join(morse2alpha.get(i) for i in decoded.split())*

*print(decoded2)*

In ultimo, ecco la flag:

*ALEXCTFTH15O1SO5UP3RO5ECR3TOTXT*

Altra soluzione (più compatta)

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

17 #Fady didn't understand well the difference between encryption and encoding, #so instead of encrypting some secret message to pass to his friend, he

#encoded it!

#The flag should be in the format: ALEXCTF

*import base64*

*with open("zero\_one", 'r') as file:*

*file = file.read()*

#We have a file with ‘ZERO ONE ZERO ONE …’ so, we convert them into numbers and cut off the spaces #(as you can see, it’s possible to concatenate all of the replace functions into a line as did here) *file = file.replace('ZERO', '0').replace('ONE', '1').replace(' ', '')*

*file = file.strip()* #strip() function removes spaces at the beginning and at the end of the string *flag = ' '*

#here we do have a string printed like '010000100100...'

#so, we need what we did in the previous challenge (taking each block of 8 bytes, converting them in #binary and then joined in the “flag” string; we must do this, otherwise we miss the first string) *for i in range(0, len(file), 8):*

*flag=' '.join([flag, chr(int(file[i:i+8], 2))])*

#now, we do have a string like “ahsgdjsb…. =”; notice it ends with an ‘=’ sign, so it should be a base64 one *decoded\_flag=base64.b64decode(flag)* #remember *import base64* and then decode it

*print(decoded\_flag)*

#here the string printed is as the following:

#b'.- .-.. . -..- -.-. - ..-. - .... .---- ..... --- .---- ... --- ..... ..- .--. ...-- .-. --- ..... . -.-. .-. ...-- - --- - -..- -' #we need to decode it using morse code, cause the -.-.- pattern is typical of that encoding #The Morse Code has many characters, so we need to define a map of morse code

#Then we need to split the string into words, then translating and lastly finding the flag *morse\_code = {*

*'.-': 'A', '-...': 'B', '-.-.': 'C', '-..': 'D', '.': 'E', '..-.': 'F',*

*'--.': 'G', '....': 'H', '..': 'I', '.---': 'J', '-.-': 'K', '.-..': 'L',*

*'--': 'M', '-.': 'N', '---': 'O', '.--.': 'P', '--.-': 'Q', '.-.': 'R',*

*'...': 'S', '-': 'T', '..-': 'U', '...-': 'V', '.--': 'W', '-..-': 'X',*

*'-.--': 'Y', '--..': 'Z', '.----': '1', '..---': '2', '...--': '3',*

*'....-': '4', '.....': '5', '-....': '6', '--...': '7', '---..': '8',*

*'----.': '9', '-----': '0', '': ' '*

*}*

#now we need to find the couple of characters, convert them into bytes

#and then decode them into ascii; we look for the flag inside the b64 decoded string, to find #the matching value between key and position inside the morse map

*flag = ' '*

*for i in decoded\_flag.decode('ascii').split():* #split the string into ascii words

*flag = ' '.join([flag, morse\_code[i]])* #join the characters to form the flag

*print(flag)*

*# ALEXCTFTH15O1SO5UP3RO5ECR3TOTXT*

Ulteriore soluzione (ancora più compatta) [cfr. https://0xd13a.github.io/ctfs/alexctf2017/ultracoded/]

*from pwn import \** #imported because here is used the *b64d* function

*import morse\_talk as mtalk* #library to read morse code

*with open('zero\_one', 'r') as f:*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

18 *data = f.read().translate(None, ' \n')*

*data = data.replace("ZERO","0").replace("ONE","1")*

*data = b64d(''.join(chr(int(data[i:i+8], 2)) for i in xrange(0, len(data), 8)))*

*data = mtalk.decode(data)*

*print data*

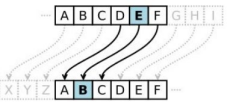
*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Lezione 4: Strumenti crittografici/Cryptographic tools Pt. 1 19 (Conti)

Storicamente parlando, uno dei primi esempi di crittografia si ha avuto con la Caesar Cipher. Il Cifrario di Cesare è uno dei primi e più semplici metodi di crittografia. È semplicemente un tipo di cifrario a sostituzione, cioè ogni lettera di un dato testo viene sostituita da una lettera con un numero fisso di posizioni in basso nell'alfabeto.

Ad esempio, con uno spostamento di 1, A verrebbe sostituita da B, B diventerebbe C e così via. Il metodo sembra prendere il nome da Giulio Cesare, che pare lo utilizzasse per comunicare con i suoi funzionari.

Per cifrare un dato testo abbiamo quindi bisogno di 

un valore intero, noto come *shift*, che indica il

numero di posizioni in cui ogni lettera del testo è

stata spostata verso il basso.

La cifratura può essere rappresentata utilizzando

l'aritmetica modulare, trasformando prima le lettere

in numeri, secondo lo schema A = 0, B = 1,..., Z = 25.

Questo shift viene deciso a priori e si shiftano tutte le

possibili combinazioni.

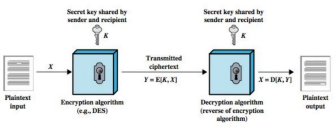
La crittografia/encryption utilizza complessi algoritmi matematici e chiavi digitali per criptare i dati. Un algoritmo di crittografia (cifrario) e una chiave di crittografia codificano i dati in un testo cifrato. Una volta che il testo cifrato viene trasmesso al destinatario, la stessa chiave o una chiave diversa (cifrario) viene utilizzata per decodificare il testo cifrato nel valore originale. Il problema principale è la gente che può intrufolarsi nella trasmissione e carpire dati che non sarebbero a loro riservati.

Le chiavi di crittografia sono il segreto della crittografia dei dati. Sono essenzialmente codici e funzionano come le chiavi fisiche: solo la chiave giusta sblocca i dati crittografati. La generazione delle chiavi di crittografia può essere effettuata manualmente o con un software che rimescola i dati con un algoritmo e crea una chiave di crittografia.

I due metodi di crittografia più distinti sono quello simmetrico e quello asimmetrico.

1) Crittografia simmetrica

La crittografia a chiave simmetrica, detta anche crittografia a chiave privata, utilizza una sola chiave per crittografare e decrittografare i dati. Per ottenere comunicazioni sicure, il mittente e il destinatario devono disporre della stessa chiave. La chiave fornisce un livello ininterrotto di crittografia dall'inizio alla fine, utilizzando la stessa chiave per le chiavi di crittografia e decrittografia. La singola chiave può assumere la forma di una password, di un codice o di una stringa di numeri generati casualmente. Esempi popolari di crittografia simmetrica sono AES, DES e Triple DES.



*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

20 2) Crittografia asimmetrica

La crittografia a chiave asimmetrica, nota anche come crittografia a chiave pubblica, utilizza due chiavi diverse: una pubblica per criptare e una privata per decriptare. La crittografia asimmetrica offre una maggiore sicurezza grazie alla verifica dell'origine dei dati e al non ripudio (l'autore non può contestarne la paternità). Tuttavia, rallenta il processo di trasmissione, la velocità della rete e le prestazioni della macchina. Un esempio popolare di crittografia asimmetrica è RSA.

Nella crittografia simmetrica ci sono delle minacce/threats:

- Crittoanalisi

○ Si basa sulla natura dell'algoritmo

○ Più una certa conoscenza delle caratteristiche del testo in chiaro/plaintext

○ Anche alcune coppie campione testo in chiaro-testo in chiaro

○ Sfrutta le caratteristiche dell'algoritmo per dedurre un testo in chiaro o una chiave specifica - Attacco di forza bruta/bruteforce attack

○ Prova tutte le chiavi possibili su un testo cifrato fino a quando non si ottiene una traduzione intelligibile in testo in chiaro

La *ricerca esaustiva delle chiavi/exhaustive key search*, o *ricerca a forza bruta*, è la tecnica di base che consiste nel provare a turno ogni possibile chiave fino a individuare quella corretta. Per identificare la chiave corretta può essere necessario possedere un testo in chiaro e il corrispondente testo cifrato, oppure, se il testo in chiaro ha qualche caratteristica riconoscibile, può essere sufficiente il solo testo cifrato. La ricerca esaustiva delle chiavi può essere effettuata su qualsiasi cifrario e, a volte, una debolezza nel programma delle chiavi del cifrario può contribuire a migliorare l'efficienza di un attacco di ricerca esaustiva delle chiavi.

Gli algoritmi popolari di crittazione sono quattro, DES/Triple-DES, AES, RSA.

- DES è il vecchio “standard di crittografia dei dati” degli anni Settanta. La sua dimensione chiave è troppo corta per una corretta sicurezza (56 bit efficaci, questo può essere forzato brutalmente, come è stato dimostrato più di dieci anni fa ). Inoltre, DES utilizza blocchi a 64 bit con una chiave da 56 bit, il che solleva alcuni potenziali problemi quando si crittografa diversi gigabyte di dati con la stessa chiave (un gigabyte non è così grande al giorno d’oggi).

- 3DES è un trucco per riutilizzare le implementazioni di DES, mediante il collegamento in cascata di tre istanze di DES (con chiavi distinte) e usando due/tre chiave uniche. È molto più sicuro, ma anche molto più lento, soprattutto nel software

- AES è il successore di DES come algoritmo di crittografia simmetrica standard per le organizzazioni federali statunitensi (e come standard anche per quasi tutti gli altri). AES accetta chiavi da 128, 192 o 256 bit (128 bit è già molto indistruttibile), utilizza blocchi a 128 bit (quindi non ci sono problemi) ed è efficiente sia nel software che nell’hardware. È stato selezionato attraverso una competizione aperta che ha coinvolto centinaia di crittografi durante diversi anni. Fondamentalmente, non puoi avere di meglio.

Quindi, in caso di dubbio, usa AES. Rimane tuttavia ancora molti diffuso DES, nella sua variante Triple-DES.

Si noti che un codice a blocchi è una casella che crittografa i “blocchi” (blocchi di dati a 128 bit con AES). Quando si crittografa un “messaggio” che può essere più lungo di 128 bit, il messaggio deve essere diviso in blocchi e il modo in cui si esegue lo split è chiamato modalità di funzionamento o “concatenamento”. La modalità naive (split semplice) è chiamata ECB e presenta problemi. L’uso corretto di un codice a blocchi non è semplice ed è più importante della selezione tra, ad esempio, AES o 3DES.

Proprio di blocchi parliamo quando si deve cifrare, in particolare abbiamo i block cipher/cifrari a blocchi e gli stream cipher/cifrario di flusso.

I cifrari a blocchi criptano i dati in blocchi di lunghezza prestabilita, mentre i cifrari a flusso non lo fanno e criptano il testo in chiaro un byte alla volta. I due approcci di crittografia, pertanto, variano notevolmente in termini di implementazione e casi d'uso.

*Scritto da Gabriel*

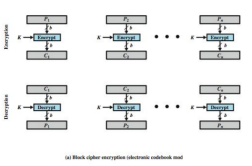
*Cybersecurity semplice (per davvero)*

I block cipher convertono i dati in chiaro in testo cifrato in blocchi di dimensioni fisse. La dimensione dei 21

blocchi dipende generalmente dallo schema di crittografia ed è solitamente in ottavi (blocchi di 64 o 128 bit). Se la lunghezza del testo in chiaro non è un multiplo di 8, lo schema di crittografia utilizza un padding per garantire blocchi completi. Ad esempio, per eseguire una crittografia a 128 bit su un testo in chiaro di

150 bit, lo schema di crittografia prevede due blocchi, uno con 128 bit e uno con i 22 bit rimanenti. All'ultimo blocco vengono aggiunti i bit ridondanti per rendere l'intero blocco uguale alla dimensione del blocco di testo cifrato dello schema di crittografia.

Essi utilizzano chiavi e algoritmi simmetrici per eseguire la crittografia e la decrittografia dei dati, ma per funzionare richiedono anche un vettore di inizializzazione (IV). Un vettore di inizializzazione è una sequenza pseudorandom o casuale di caratteri utilizzata per crittografare il primo blocco di caratteri del blocco di testo in chiaro. Il testo cifrato risultante per il primo blocco di caratteri funge da vettore di inizializzazione per i blocchi successivi. Pertanto, il cifrario simmetrico produce un blocco di testo cifrato unico per ogni iterazione, mentre il IV viene trasmesso insieme alla chiave simmetrica e non richiede la cifratura. Esempi di algoritmi che li utilizzano: AES/DES

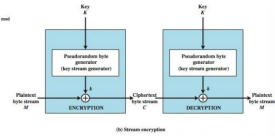


Uno stream cipher cripta una stringa continua di cifre binarie applicando trasformazioni variabili nel tempo ai dati del testo in chiaro. Pertanto, questo tipo di crittografia funziona bit per bit, utilizzando flussi di chiavi per generare testo cifrato per messaggi in chiaro di lunghezza arbitraria. Il cifrario combina una chiave (128/256 bit) e una cifra nonce (64-128 bit) per produrre il flusso di chiavi, un numero pseudorandom sottoposto a XOR con il testo in chiaro per produrre il testo cifrato. Mentre la chiave e il nonce possono essere riutilizzati, il flusso di chiavi deve essere unico per ogni iterazione di crittografia per garantire la sicurezza. I cifrari a flusso ottengono questo risultato utilizzando registri a spostamento a retroazione per generare un nonce unico (numero usato una sola volta) per creare il flusso di chiavi.

Gli schemi di crittografia che utilizzano i cifrari a flusso hanno meno probabilità di propagare errori a livello di sistema, poiché un errore nella traduzione di un bit non influisce in genere sull'intero blocco di testo in chiaro. La crittografia a flusso avviene inoltre in modo lineare e continuo, il che la rende più semplice e veloce da implementare. D'altro canto, i cifrari a flusso mancano di diffusione, poiché ogni cifra del testo in chiaro è mappata su un'uscita del testo cifrato. Inoltre, non convalidano l'autenticità, rendendoli vulnerabili alle inserzioni. Se gli hacker violano l'algoritmo di crittografia, possono inserire o modificare il messaggio crittografato senza essere scoperti. I cifrari a flusso sono utilizzati principalmente per crittografare i dati in applicazioni in cui la quantità di testo in chiaro non può essere determinata e in casi di utilizzo a bassa latenza.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

22 Esempi di algoritmi che usano questo principio: RC4 (altro algoritmo molto usato in crittografia) 

Analizziamo il seguente esempio:

- Cifrario a sostituzione: l’alfabeto è shiftato di una serie di caratteri. Chiaramente è molto basilare e non sicuro. Un esempio è proprio il cifrario di Cesare.

Per esempio, avendo il testo

“QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD”

La soluzione è provare tutti i possibili shift/tutte le combinazioni alfabetiche, tramite crittoanalisi e forza bruta.

La soluzione è: “THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG”

Si usa spesso anche la XOR negli algoritmi crittografici, operazione che tra due numeri:

- Restituisce 1 quando questi sono diversi

- Restituisce 0 quando questi sono uguali

- È rappresentata dal simbolo “^”

Esempi:

*enc\_message = clear\_message ^ key*

*clear\_message = enc\_message ^ key*

*key = clear\_message ^ enc\_message*

La XOR ha alcune proprietà da considerare:

- È commutativa

- É associativa

- Tutti gli XOR con 0 danno come risultato l’altro termine: *a ^ 0 = a*

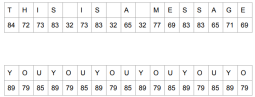
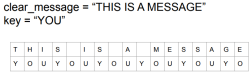
- Tutti gli XOR con lo stesso termine danno zero: *a ^ a = 0*

La XOR viene usata tra una chiave ed un messaggio:

- Spesso len(key) << len(message)

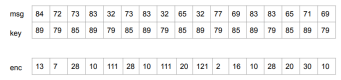
- “Ripetiamo la chiave” nel messaggio

Esempio:

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

La XOR tra due interi è il risultato della XOR della loro rappresentazione binaria:

23 

● 84 = 1010100

● 89 = 1011001

● 13 = 0001101

È interessante notare che se:

- La chiave ha la stessa dimensione del messaggio

- La chiave è tenuta segreta e generata in modo veramente casuale

Allora il cifrario XOR è certamente impossibile da decifrare. Questo è noto come one-time pad. Tuttavia, un semplice XOR non dovrebbe essere utilizzato in produzione perché la lunghezza della chiave deve essere troppo lunga per essere pratica.

XOR è un modo economico per criptare i dati con una password. Ne esistono di due tipi principali: - Crittografia XOR a singolo byte

o La crittografia XOR a singolo byte è banale da forzare, poiché ci sono solo 255 combinazioni di chiavi da provare.

- Crittografia XOR a più byte

o La crittografia XOR a più byte diventa esponenzialmente più difficile quanto più lunga è la chiave, ma se il testo crittografato è abbastanza lungo, l'analisi della frequenza dei caratteri è un metodo valido per trovare la chiave. L'analisi della frequenza dei caratteri consiste nel dividere il testo cifrato in gruppi in base al numero di caratteri della chiave. Questi gruppi vengono poi forzati utilizzando l'idea che alcune lettere appaiono più frequentemente nell'alfabeto inglese rispetto ad altre.

Il metodo Kasiski è un metodo crittoanalitico per l'attacco del *cifrario di Vigenère.*

É il più semplice dei cifrari polialfabetici. Si basa sull'uso di un versetto per controllare l'alternanza degli alfabeti di sostituzione. Il metodo si può considerare una generalizzazione del cifrario di Cesare; invece di spostare sempre dello stesso numero di posti la lettera da cifrare, questa viene spostata di un numero di posti variabile ma ripetuto, determinato in base ad una parola chiave, da concordarsi tra mittente e destinatario, e da scrivere ripetutamente sotto il messaggio, carattere per carattere; la chiave era detta anche verme, per il motivo che, essendo in genere molto più corta del messaggio) e dei cifrari ad esso simili.

Il testo cifrato si ottiene spostando la lettera chiara di un numero fisso di caratteri, pari al numero ordinale della lettera corrispondente del verme. Di fatto si esegue una somma aritmetica tra l'ordinale del chiaro (A = 0, B = 1, C = 2...) e quello del verme; se si supera l'ultima lettera, la Z, si ricomincia dalla A, secondo la logica delle aritmetiche finite.

Il vantaggio rispetto ai cifrari monoalfabetici (come il cifrario di Cesare o quelli per sostituzione delle lettere con simboli/altre lettere) è evidente: il testo è cifrato con n alfabeti cifranti. In questo modo, la stessa lettera viene cifrata (se ripetuta consecutivamente) n volte; ciò rende quindi più complessa la crittoanalisi del testo cifrato.

Tornando al metodo Kasiski, il maggiore Kasiski (suo creatore) notò che spesso in un crittogramma di Vigénère si possono notare sequenze di caratteri identiche, poste ad una certa distanza fra di loro; questa distanza può, con una certa probabilità, corrispondere alla lunghezza della chiave, o a un suo multiplo.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

24 In genere la stessa lettera con il cifrario di Vigénère viene cifrata in modo diverso nelle sue varie occorrenze, come si confà ai cifrari polialfabetici, ma se due lettere del testo in chiaro sono poste ad una distanza pari alla lunghezza della chiave (o un suo multiplo), questo fa sì che vengano cifrate nello stesso modo. 

Individuando tutte le sequenze ripetute (cosa che

avviene frequentemente in un testo lungo), si può

dedurre quasi certamente che la lunghezza della

chiave è il massimo comun divisore tra le distanze

tra sequenze ripetute, o al più un suo multiplo. Di

fianco, un esempio.

Conoscere la lunghezza n della chiave permette di ricondurre il messaggio cifrato ad n messaggi intercalati cifrati con un cifrario di Cesare facilmente decifrabile.

Esercizi Lezione 4

1) Julius: Testo, aiuti e soluzione

Testo

*Julius,Q2Flc2FyCg==*

*-------------------*

*World of Cryptography is like that Unsolved Rubik's Cube, given to a child that has no idea about it. A new combination at every turn.*

*Can you solve this one, with weird name?*

*ciphertext: fYZ7ipGIjFtsXpNLbHdPbXdaam1PS1c5lQ==*

Aiuti:

1) The challenge description contains two “secret” messages. Both end with ==.

Does this recall you an encoding strategy?

2) This is a Caesar cipher. A brute force is enough to break it.

Soluzione

# in the first line of the description contains a hint 🡪 Julius,Q2Flc2FyCg==

# since it ends with ==, the first hypothesis is that this is a base64 encoding

# let's decode it!

*import base64*

*enc\_b64 = 'Q2Flc2FyCg=='*

#we define a function. It might be helpful in future

*def base64tostring(text):*

*return base64.b64decode(text).decode('utf-8', errors="ignore")*

*print(f"Decoding=\t{base64tostring(enc\_b64)}")*

# the hints says "caesar", and we know a famous cipher with this name. We can apply the reverse to the # ciphered text given in description

*puzzle = 'fYZ7ipGIjFtsXpNLbHdPbXdaam1PS1c5lQ=='*

*print("The length of the puzzle is:\t", len(puzzle))*

# the length is a multiple of 4, and the alphabet seems too regular (no punctuation).

# we can think that this is a base64 encoded string

*puzzle\_dec = base64tostring(puzzle)*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

25 *print("Decoded puzzle:", puzzle\_dec)*

# now we have a lot of no sense characters. We can try to apply the caesar cipher

*def caesar\_cracker(text, from\_ = -30, to\_=+30):*

*for i in range(from\_, to\_): #keys [-30, 30]*

#decode

*curr\_step = ' '.join([chr(ord(c) + i) for c in text])*

#print

*print(f"Step={i}\t{curr\_step}")*

*caesar\_cracker(puzzle\_dec)*

## we look for some readable flags. We find one at step -24

#FLAG: ecCTF3T\_7U\_BRU73?!

Altra soluzione

*import base64*

*ciphertext = 'fYZ7ipGIjFtsXpNLbHdPbXdaam1PS1c5lQ=='*

#We can see the string is a base64 one, so we define a function to decode it

*def base64decode(text):*

*return str(base64.b64decode(text).decode('ascii', 'ignore'))*

#Fundamental to put “str” and “ignore” to read all the characters without having problems

*decoded=base64decode(ciphertext)*

*print(decoded)*

# As of now, we have this kind of string (with the *b* that stays for *binary*, as visible):

# b'}\x86{\x8a\x91\x88\x8c[l^\x93KlwOmwZjmOKW9\x95' (25 characters in total)

# Let's try to implement a bruteforce algorithm to decrypt it, making a shift of 24 to get the right one *flag = ' '*

*for i in range(len(decoded)):*

*code = ord(decoded[i]) - 24*

*flag += chr(code)*

*print(flag)*

Una possibile implementazione generale del *Caesar cipher* è la seguente, prendendo un testo generico e uno shift custom da inserire; distingue tra lettere maiuscole e lettere minuscole, realizzando uno shift pari a tutte le lettere dell’alfabeto partendo dal carattere in quella specifica posizione.

*def caesarCipher(plainText, shift):*

*cipherText = ""*

*for i in range(len(plainText)):*

*if plainText[i].isupper():*

*cipherText += chr((ord(plainText[i]) + shift - 65) % 26 + 65)*

#65 is the ASCII value of 'A' and 26 is the number of letters in the alphabet

*else:*

*cipherText += chr((ord(plainText[i]) + shift - 97) % 26 + 97)*

#97 is the ASCII value of 'a' and 26 is the number of letters in the alphabet

*return cipherText*

La decodifica non cambia rispetto a questa funzione, dato che letteralmente sottrae lo shift fatto:

*def caesarCipherDecoder(cipherText, shift):*

*plainText = ""*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

26 *for i in range(len(cipherText)): if cipherText [i].isupper():*

*plainText += chr((ord(cipherText [i]) - shift - 65) % 26 + 65)*

#65 is the ASCII value of 'A' and 26 is the number of letters in the alphabet

*else:*

*plainText += chr((ord(cipherText [i]) - shift - 97) % 26 + 97)*

#97 is the ASCII value of 'a' and 26 is the number of letters in the alphabet

*return cipherText*

2) I Agree: Testo, aiuti e soluzione

Testo

*Crack the cipher: vhixoieemksktorywzvhxzijqni*

*Your clue is: "caesar is everything. But he took it to the next level."*

Aiuto:

1) You might try with a Caesar … but it is not enough.

The challenge description tells that this is “the next level of Caesar cipher”. What is it? Have a look on Google, if you find the cipher, you solve the exercise.

Soluzione

# The description says: "caesar is everything"

# it is a caesar cipher, and we need to crack it as in the previous exercise

*puzzle = 'vhixoieemksktorywzvhxzijqni'*

# Specifying the steps of the for loop, where \t means “tab”

*def caesar\_cracker(text, from\_ = -30, to\_=+30):*

*for i in range(from\_, to\_): #keys [-30, 30]*

# decode

*curr\_step = ' '.join([chr(ord(c) + i) for c in text])*

# print

*print(f"Step={i}\t{curr\_step}")*

*caesar\_cracker(puzzle)*

# I don't see any proper flag. We need to find another way

#The description says that it is the "next level" of caesar.

# After some investigation, we can find that the evolution of a caesar cipher is the vigenere cipher. # However, the vigenere also requires a key.

# Google can help us! There are some online bruteforce services for these kinds of ciphers. # https://www.guballa.de/vigenere-solver

# The flag is reached: theforceisstrongwiththisone. The key is "caesar" ... as the hint suggested

Segue una possibile implementazione realizzata da me in Python, interessante a livello di codice e per capire il ragionamento di Vigénère:

#The hint "to the next level" says everything

#cause it refers to the Vigénère cipher, which is a Caesar cipher that uses an entire keyword *text='vhixoieemksktorywzvhxzijqni'*

#The key word is "caesar" and we define an algorithm to use the Vigénère cipher

*def vigenere(text, key):*

*key = key \* (len(text) // len(key) + 1)* #repeat the key word and floor “//” division between text/text+1

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

27 *return ' '.join([chr((ord(text[i]) - ord(key[i])) % 26 + ord('a')) for i in range(len(text))])* #the algorithm is the same as the Caesar cipher but we use the key word instead of a single letter #so, we make a shift subtracting from the key word from the text modulo 26 to keep the letters in the #alphabet and adding the ascii code of 'a' to get the right letter; we do this for every letter in the text *print (vigenere(text, 'caesar'))*

#the result is "theforceisstrongwiththisone"

Un’implementazione generale di codifica di Vigenere che assomiglia a quella del Caesar Cipher è la seguente:

#Vigenere Cipher function

*def vigenereCipher(plainText, key):*

*cipherText = ""*

*for i in range(len(plainText)):*

*if plainText[i].isupper():*

*cipherText += chr((ord(plainText[i]) + ord(key[i % len(key)]) - 65) % 26 + 65)* #here we modulo also the key length and this is the uppercase implementation

*else:*

*cipherText += chr((ord(plainText[i]) + ord(key[i % len(key)]) - 97) % 26 + 97)* #here we modulo also the key length and this is the lowercase implementation

*return cipherText*

La decodifica, come prima, letteralmente sottrae la posizione della lunghezza della chiave realizzato lo shift a seconda della lettera maiuscola/minuscola:

*def vigenereCipherDecoder(cipherText, key):*

*plainText = ""*

*for i in range(len(cipherText)):*

*if cipherText[i].isupper():*

*plainText += chr((ord(cipherText[i]) - ord(key[i % len(key)]) - 65) % 26 + 65)* #65 is the ASCII value of 'A' and 26 is the number of letters in the alphabet

*else:*

*plainText += chr((ord(cipherText[i]) - ord(key[i % len(key)]) - 97) % 26 + 97)* #97 is the ASCII value of 'a' and 26 is the number of letters in the alphabet

*return plainText*

3) Alphabet Soup: Testo, aiuti e soluzioni

Testo:

(Viene dato un file “encrypted.txt” con il seguente testo):

MKXU IDKMI DM BDASKMI NLU XCPJNDICFQ! K VDMGUC KW PDT GKG NLKB HP LFMG DC TBUG PDTC CUBDTCXUB. K'Q BTCU MDV PDT VFMN F WAFI BD LUCU KN KB WAFI

GDKMINLKBHPLFMGKBQDCUWTMNLFMFMDMAKMUNDDA

Aiuti:

1) Ok, in this exercise we need to do some educated guesses, a.k.a., cryptoanalysis.

You might want to: i) get the frequency of the characters and ii) map it somehow (remember that this is an English text).

2) Here we report some mappings:

K -> i

F -> a

P -> y

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

28

Soluzione

# we only have an encrypted message

*puzzle = "MKXU IDKMI DM BDASKMI NLU XCPJNDICFQ! K VDMGUC KW PDT GKG NLKB HP LFMG DC TBUG PDTC CUBDTCXUB. K'Q BTCU MDV PDT VFMN F WAFI BD LUCU KN KB WAFI*

*GDKMINLKBHPLFMGKBQDCUWTMNLFMFMDMAKMUNDDA"*

# since I do not have any clue, I try to use the cryptanalysis strategy

# in the ciphers, in general, each letter is associated to a given alphabet

# we can try to find associations.

#the first step is to see the frequency of each character with this function

*chr2freq = {}*

*for c in puzzle:*

*if c not in chr2freq:*

*chr2freq[c] = 1*

*else:*

*chr2freq[c] += 1*

# sort the dictionary by value in descending order and store it in a list of tuples (key, value) *sorted\_x = sorted(chr2freq.items(), key=lambda kv: kv[1], reverse = True)*

#lambda is used to create an anonymous function, in this case, it is used to sort the dictionary by value #taking each time the first element as saw by *kv*

*print(sorted\_x)*

#This here prints: [(' ', 30), ('D', 17), ('M', 15), ('K', 14), ('U', 11), ('B', 10), ('C', 10), ('F', 9), ('N', 8), ('I', 7), ('L', #7), ('G', 7), ('T', 7), ('P', 6), ('A', 5), ('W', 4), ('X', 3), ('Q', 3), ('V', 3), ('H', 2), ('S', 1), ('J', 1), ('!', 1), ('.', 1)]

# hypothesis 1: the text is english written

# we can find online what are the most used english characters

# e.g., http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html # we see that the 'K' is "alone", and we can think to

# K = I

*voc = {'K': 'i'}*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle) #this is a list comprehension that substitutes the character in the string with the one in the guess dictionary*

*print(voc, '\n' ,dec)*

# then there is an 'i'Q', which is an M

*voc['Q'] = 'm'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

# F -> 'a'

*voc['F'] = 'a'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

## the semilast word contains four letters, and the third character

# is an 'a'. This word could be flag

*voc['W'] = 'f'*

*voc['A'] = 'l'*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

29 *voc['I'] = 'g' dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

# not a lot of info...

# however, there is a word with GiG ... G must be a 'D'

*voc['G'] = 'd'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

# then there is a sentence with "if PDT did"

# PDT could be "you", a likely word with letters not used yet

*voc['P'] = 'y'*

*voc['D'] = 'o'*

*voc['T'] = 'u'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

# slightly better. The second word is goiMg

# M->n

*voc['M'] = 'n'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

# back on the second sentence

# if you did NLiB Hy ... seems "if you did this by"

*voc['N'] = 't'*

*voc['L'] = 'h'*

*voc['B'] = 's'*

*voc['H'] = 'b'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

#the fourth word must be "solving"

*voc['S'] = 'v'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

#fifth word is “the”

*voc['U'] = 'e'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

#then, "I wonder if you ..."

*voc['V'] = 'w'*

*voc['C'] = 'r'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

#ready to conclude. we can see the flag .. but let’s finish the job

#niXe -> nice

*voc['X'] = 'c'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

30 *print(voc, '\n' ,dec)*

#the last word of the first sentence is cryptogram

*voc['J'] = 'p'*

*dec = ' '.join(c if c not in voc else voc[c] for c in puzzle)*

*print(voc, '\n' ,dec)*

## we did it

# nice going on solving the cryptogram!

# i wonder if you did this by hand or used your resources.

# i'm sure now you want a flag so here it is

# flag doingthisbyhandismorefunthananonlinetool

Volendo fare le cose che non siano manuali, tool molto usati dalle challenge online nei cifrari di sostituzione (e, per questa challenge, da tutto il resto del web) sono i seguenti:

https://www.quipqiup.com/

https://www.guballa.de/substitution-solver

Inserendo letteralmente il testo, al primo colpo viene fuori la flag e il testo tradotto; si usano quindi strumenti di risoluzione di questo tipo.

4) DES Solver: Testo, aiuti e soluzione

(*esercizio utile a livello implementativo/visivo; abbastanza complesso e più degli esami stessi*) Altri riferimenti di soluzioni al link: https://ctftime.org/task/5405

Testo

*Larry is working on an encryption algorithm based on DES.*

*He has not worked out all the kinks yet, but he thinks it works.*

*Your job is to confirm that you can decrypt a message, given the algorithm and parameters used. His system works as follows:*

*1. Choose a plaintext that is divisible into 12bit 'blocks'*

*2. Choose a key at least 8bits in length*

*3. For each block from i=0 while i<N perform the following operations*

*4. Repeat the following operations on block i, from r=0 while r<R*

*5. Divide the block into 2 6bit sections Lr,Rr*

*6. Using Rr, "expand" the value from 6bits to 8bits.*

*Do this by remapping the values using their index, e.g.*

*1 2 3 4 5 6 -> 1 2 4 3 4 3 5 6*

*7. XOR the result of this with 8bits of the Key beginning with Key[iR+r] and wrapping back to the beginning if necessary.*

*8. Divide the result into 2 4bit sections S1, S2*

*9. Calculate the 2 3bit values using the two "S boxes" below, using S1 and S2 as input respectively.*

*S1 0 1 2 3 4 5 6 7*

*0 101 010 001 110 011 100 111 000*

*1 001 100 110 010 000 111 101 011*

*S2 0 1 2 3 4 5 6 7*

*0 100 000 110 101 111 001 011 010*

*1 101 011 000 111 110 010 001 100*

*10. Concatenate the results of the S-boxes into 1 6bit value*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

*11. XOR the result with Lr*

31

*12. Use Rr as Lr and your altered Rr (result of previous step) as Rr for any further computation on block i*

*13. increment r*

*He has encrypted a message using Key="Mu", and R=2. See if you can decipher it into plaintext. Submit your result to Larry in the format Gigem{plaintext}.*

*Binary of ciphertext: 01100101 00100010 10001100 01011000 00010001 10000101*

Aiuti:

1) This is a tough one! You need to carefully reverse the process presented in the exercise description. Here a couple of functions written for you:

def rule9b0(b):

#get indexed

row = int(b[0])

col = int(b[1:], 2)

matrix = [['101','010','001','110','011',’100','111','000'], ['001', '100','110','010','000','111','101',’011']

]

return matrix[row][col]

def rule9b1(b):

#get indexed

row = int(b[0])

col = int(b[1:], 2)

matrix = [['100','000','110','101','111','001','011','010'], ['101','011','000','111','110','010','001','100']]

return matrix[row][col]

*2) This is my encryption algorithm.*

*def rule9b0(b):*

*#get indexed*

*row = int(b[0])*

*col = int(b[1:], 2)*

*matrix = [['101','010', '001', '110', '011', '100', '111', '000'],*

*['001','100','110','010','000','111','101','011']]*

*return matrix[row][col]*

*def rule9b1(b):*

*#get indexed*

*row = int(b[0])*

*col = int(b[1:], 2)*

*matrix = [['100','000','110','101','111','001','011','010'],*

*['101','011','000','111','110','010','001','100']]*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

32 *return matrix[row][col]*

*# we need to convert the text into bits*

*def string2binary(text):*

*return ' '.join(f"{ord(c):08b}" for c in text)*

*def binary2string(text):*

*return ' '.join(f"{ord(c):08b}" for c in text)*

*def splitblock(block):*

*Lr = block[:6]*

*Rr = block[6:]*

*return Lr, Rr*

*def expand\_miniblock(b):*

*return b[0] + b[1] + b[3] + b[2] + b[3] + b[2] + b[4] + b[5]*

*def xor(a, b):*

*res = int(a, 2) ^ int(b, 2)*

*return f"{res:08b}"*

*def encrypt(text, key, R):*

*text\_encr = ' '*

*#Rule 1.*

*text\_bin = string2binary(text)*

*if (len(text\_bin) % 12 != 0):*

*raise Exception(f'Rule 1 not respected.')*

*#Rule 2*

*key\_bin = string2binary(key)*

*if (len(text\_bin) < 8):*

*raise Exception('Rule 2 not respected')*

*#rule3*

*#we have some blocks ...*

*for bnum in range(len(text\_bin) // 12):*

*i = bnum*

*#define the block*

*from\_ = 0 + 12\*bnum*

*to\_ = 12 \* (bnum + 1)*

*block = text\_bin[from\_:to\_]*

*#rule4*

*for r in range(R):*

*#rule5*

*Lr, Rr = splitblock(block)*

*#rule6*

*Rr\_expanded = expand\_miniblock(Rr)*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

33 *#Rule7 curr\_key = key\_bin[(i\* R + r) : ((i\* R + r)+8)]*

*Rr\_exp\_xor\_key = xor(Rr\_expanded, curr\_key)*

*#Rule8*

*Rr\_exp\_xor\_key\_0 = Rr\_exp\_xor\_key[:4]*

*Rr\_exp\_xor\_key\_1 = Rr\_exp\_xor\_key[4:]*

*#Rule9*

*Rr\_exp\_xor\_key\_0\_conv = rule9b0(Rr\_exp\_xor\_key\_0)*

*Rr\_exp\_xor\_key\_1\_conv = rule9b1(Rr\_exp\_xor\_key\_1)*

*#Rule10*

*Rr\_sboxes = Rr\_exp\_xor\_key\_0\_conv + Rr\_exp\_xor\_key\_1\_conv*

*if len(Rr\_sboxes) != 6:*

*raise Exception("Error on Rule 10")*

*#Rule11*

*Rr\_alt = xor(Lr, Rr\_sboxes)[2:]*

*#Rule12*

*block = Rr + Rr\_alt*

*#Rule13*

*#end of step*

*#append the result.*

*text\_encr += block*

*return text\_encr*

Soluzione

#the description give us a simplified version of DES.

#the first idea is to reconstruct the process

#Rule 1. Choose a plaintext that is divisible into 12bit 'blocks'

#Rule 2. Choose a key at least 8bits in length

#Rule 3. For each block from i=0 while i<N perform the following operations

#Rule 4. Repeat the following operations on block i, from r=0 while r<R

#Rule 5. Divide the block into 2 6bit sections Lr,Rr

#Rule 6. Using Rr, "expand" the value from 6bits to 8bits.

# Do this by remapping the values using their index, e.g.

# 1 2 3 4 5 6 -> 1 2 4 3 4 3 5 6

#Rule 7. XOR the result of this with 8bits of the Key beginning with Key[iR+r] and wrapping back to the # beginning if necessary.

#Rule 8. Divide the result into 2 4bit sections S1, S2

#Rule 9. Calculate the 2 3bit values using the two "S boxes" below, using S1 and S2 as input respectively. #

# S1 0 1 2 3 4 5 6 7

# 0 101 010 001 110 011 100 111 000

# 1 001 100 110 010 000 111 101 011

#

# S2 0 1 2 3 4 5 6 7

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

# 0 100 000 110 101 111 001 011 010

34

# 1 101 011 000 111 110 010 001 100

#Rule10. Concatenate the results of the S-boxes into 1 6bit value

#Rule11. XOR the result with Lr

#Rule12. Use Rr as Lr and your altered Rr (result of previous step) as Rr for any further computation on block i

#Rule13 increment r

# Let’s find the other two 3-bit values in a matrix for the S1 and S2 block of rule 9

*def rule9b0(b):*

# get indexed

*row = int(b[0])*

*col = int(b[1:], 2)*

*matrix = [['101','010', '001', '110', '011', '100', '111', '000'],*

*['001','100','110','010','000','111','101','011']]*

*return matrix[row][col]*

*def rule9b1(b):*

# get indexed

*row = int(b[0])*

*col = int(b[1:], 2)*

*matrix = [['100','000','110','101','111','001','011','010'],*

*['101','011','000','111','110','010','001','100']]*

*return matrix[row][col]*

# We need to convert the text into bits (8 binary bits, hence the “08b” in the code)

*def string2binary(text):*

*return ' '.join(f"{ord(c):08b}" for c in text)*

# Similarly, we convert the binary to the string (similar reasoning with the same code, but in reverse) *def binary2string(text):*

*return ' '.join(f"{ord(c):08b}" for c in text)*

# Made for rule 5 (Divide the block into 2 6bit sections Lr,Rr)

# To do that, we just iterate in row the blocks, iterating up until 6 for the first one

# and, for the second one, starting from the 6th block all the way to the end

*def splitblock(block):*

*Lr = block[:6]*

*Rr = block[6:]*

*return Lr, Rr*

# Made for rule 6

# Using Rr, "expand" the value from 6bits to 8bits. Do this by remapping the values using their index, e.g. # 1 2 3 4 5 6 -> 1 2 4 3 4 3 5 6. In the code we use this exact reasoning

*def expand\_miniblock(b):*

*return b[0] + b[1] + b[3] + b[2] + b[3] + b[2] + b[4] + b[5]*

# XOR function already defined for rule 7 🡪 XOR the result of this with 8bits of the Key beginning with # Key[iR+r] and wrapping back to the beginning if necessary.

# To accomplish the function, we take “a” and “b” in binary and return the result XORed in string form *def xor(a, b):*

*res = int(a, 2) ^ int(b, 2)*

*return f"{res:08b}"*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

35

# The whole function of encryption (the decryption follows up right ahead this one); R stays for “rounds”, # so it does around a binary

*def encrypt(text, key, R):*

*text\_encr = ' '*

#Rule 1 🡪 Choose a plaintext that is divisible into 12bit 'blocks' (so we use the modulo, as done with # even numbers usually)

*text\_bin = string2binary(text)*

*if (len(text\_bin) % 12 != 0):*

*raise Exception(f'Rule 1 not respected.')*

#Rule 2 🡪 Choose a key at least 8 bits in length (so just checking the length with “len”) *key\_bin = string2binary(key)*

*if (len(text\_bin) < 8):*

*raise Exception('Rule 2 not respected')*

# Rule 3 🡪 For each block from i=0 while i<N perform the following operations

# What we are doing here with the double slash (//) is the “floor” division, so

# divides the first number by the second number and rounds the result down to the nearest integer # (or whole number). We make this to have 12-bit divisible blocks

*for bnum in range(len(text\_bin) // 12):*

*i = bnum*

# define the block, with a start index for a 12-bit block and the end is similar but adding one to the sum *from\_ = 0 + 12\*bnum*

*to\_ = 12 \* (bnum + 1)*

*block = text\_bin[from\_:to\_]* # So the block will iterate all the way in the text

# Rule 4 🡪 Repeat the following operations on block i, from r=0 while r<R

*for r in range(R):*

# Rule 5 🡪 Divide the block into 2 6bit sections Lr,Rr

*Lr, Rr = splitblock(block)*

# Rule 6 🡪Using Rr, "expand" the value from 6bits to 8bits. Do this by remapping the values using # their index - e.g., 1 2 3 4 5 6 -> 1 2 4 3 4 3 5 6

*Rr\_expanded = expand\_miniblock(Rr)*

# Rule 7 🡪 XOR the result of this with 8bits of the Key beginning with Key[iR+r] and wrapping back # the beginning if necessary.

*curr\_key = key\_bin[(i\* R + r) : ((i\* R + r)+8)]*

*Rr\_exp\_xor\_key = xor(Rr\_expanded, curr\_key)*

# Rule 8 🡪 Divide the result into two 4-bit sections S1, S2

*Rr\_exp\_xor\_key\_0 = Rr\_exp\_xor\_key[:4]*

*Rr\_exp\_xor\_key\_1 = Rr\_exp\_xor\_key[4:]*

# Rule9 🡪 Calculate the two 3-bit values using the two "S boxes" below, using S1 and S2 as input #respectively.

*Rr\_exp\_xor\_key\_0\_conv = rule9b0(Rr\_exp\_xor\_key\_0)*

*Rr\_exp\_xor\_key\_1\_conv = rule9b1(Rr\_exp\_xor\_key\_1)*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

36 #Rule10 🡪 Concatenate the results of the S-boxes into one 6bit value *Rr\_sboxes = Rr\_exp\_xor\_key\_0\_conv + Rr\_exp\_xor\_key\_1\_conv*

*if len(Rr\_sboxes) != 6:*

*raise Exception("Error on Rule 10")*

#Rule11 🡪 XOR the result with Lr

*Rr\_alt = xor(Lr, Rr\_sboxes)[2:]*

#Rule12 🡪 Use Rr as Lr and your altered Rr (result of previous step) as Rr for any further #computation on block i

*block = Rr + Rr\_alt*

#end of step - Rule13 🡪 Increment R (already incremented because of the *for* lop nature) #append the result.

*text\_encr += block*

*return text\_encr*

# solution

*def decrypt(text, key, R):*

*text\_dec = ' '*

#the text is already in the bit format.

#we only need to convert the key

*text\_bin = text*

*key\_bin = string2binary(key)*

*if (len(text\_bin) < 8):*

*raise Exception('Rule 2 not respected')*

#like the previous cycle, we need to iterate over the blocks.

#since the blocks are independent between each other, we can use the

#same order of the encryption algorithm

*for bnum in range(len(text\_bin) // 12):*

*i = bnum*

#define the block

*from\_ = 0 + 12\*bnum*

*to\_ = 12 \* (bnum + 1)*

*block = text\_bin[from\_:to\_]*

#we now need to reverse the rule4 loop

#since this time the results obtained in a specific round affect the

#next one, we use the reverse order

#our goal is to retrieve the original Lr and Rr

*for r in range(R-1, -1, -1):*

#in the first round we obtain the components

# block = Lr + Rr

*Rr, Rr\_alt = splitblock(block)*

#to reverse Rule11 we can use the xor properties

#e.g.: A xor B = C, C xor B = A

#however, we need to have one of the components at least (A or B) since

# we have C

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

# N.B. we have something useful. Which is Rr.

37

# We know half of the info! we can easily obtain Rr\_sboxes

# compute from rule6 to rule10, where Lr is not involved at all

# Rule 6

# Using Rr, "expand" the value from 6bits to 8bits. Do this by remapping the values using their # index, e.g., 1 2 3 4 5 6 -> 1 2 4 3 4 3 5 6

*Rr\_expanded = expand\_miniblock(Rr)*

#Rule 7 🡪 XOR the result of this with 8bits of the Key beginning with Key[iR+r] and wrapping back # the beginning if necessary.

*curr\_key = key\_bin[(i\* R + r) : ((i\* R + r)+8)]*

*Rr\_exp\_xor\_key = xor(Rr\_expanded, curr\_key)*

#Rule 8 🡪 Divide the result into 2 4bit sections S1, S2

*Rr\_exp\_xor\_key\_0 = Rr\_exp\_xor\_key[:4]*

*Rr\_exp\_xor\_key\_1 = Rr\_exp\_xor\_key[4:]*

#Rule9 🡪 Calculate the 2 3bit values using the two "S boxes" below, using S1 and S2 as input #respectively.

*Rr\_exp\_xor\_key\_0\_conv = rule9b0(Rr\_exp\_xor\_key\_0)*

*Rr\_exp\_xor\_key\_1\_conv = rule9b1(Rr\_exp\_xor\_key\_1)*

*Rr\_sboxes = Rr\_exp\_xor\_key\_0\_conv + Rr\_exp\_xor\_key\_1\_conv*

#Rule10 🡪 Concatenate the results of the S-boxes into 1 6bit value

*if len(Rr\_sboxes) != 6:*

*raise Exception("Error on Rule 10")*

#we can finally obtain Lr

*Lr = xor(Rr\_alt, Rr\_sboxes)*

*Lr = Lr[2:]*

*block = Lr + Rr*

#obtain the new block

*new\_block = Lr + Rr*

#print(new\_block)

# raise Exception('# DEBUG: ')

#append the result.

*text\_dec += new\_block*

#Convert from 8digit bits into the integer, and then

#in the ascii representation

*res = ' '*

*for i in range(len(text\_dec) // 8):*

*res += chr(int(text\_dec[(i \* 8): ((i+1) \* 8)] ,2))*

*print(res)*

#print(encrypt('abc', 'Mu', 2))

*puzzle = "011001010010001010001100010110000001000110000101"*

*key\_ex = 'Mu'*

*R\_ex = 2*

#decrypt(puzzle, key\_ex, R\_ex)

#flag: Min0n!

decrypt(encrypt('Min0n!', 'Mu', 2), 'Mu', 2) #driver code example

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Lezione 5: Strumenti crittografici/Cryptographic tools Pt. 2 38 (Conti)

Un altro tipo di minaccia che esiste per i dati è la mancanza di autenticazione dei messaggi/message authentication. In questo caso, l'utente non è sicuro dell'autore del messaggio. L'autenticazione del messaggio può essere fornita mediante tecniche crittografiche che utilizzano chiavi segrete come nel caso della crittografia.

Requisiti di un meccanismo autenticazione:

• Rivelazione: Significa rilasciare il contenuto del messaggio a qualcuno che non possiede una chiave crittografica appropriata.

• Analisi del traffico: Determinazione del modello di traffico attraverso la durata della connessione e la frequenza delle connessioni tra le diverse parti.

• Inganno: Aggiunta di messaggi fuori contesto da una fonte fraudolenta in una rete di comunicazione. Questo porta alla sfiducia tra le parti che comunicano e può anche causare la perdita di dati critici.

• Modifica del contenuto: Modifica del contenuto di un messaggio. Ciò include l'inserimento di nuove informazioni o la cancellazione/modifica di quelle esistenti.

• Modifica della sequenza: Modifica dell'ordine dei messaggi tra le parti. Ciò include l'inserimento, la cancellazione e il riordino dei messaggi.

• Modifica dei tempi: Include il replay e il ritardo dei messaggi inviati tra le diverse parti. In questo modo viene interrotto anche il tracciamento della sessione.

• Rifiuto della fonte: Quando la fonte nega di essere l'originatore di un messaggio.

• Rifiuto della destinazione: Quando il destinatario del messaggio nega la ricezione.

In generale:

- Protegge dagli attacchi attivi

- Verifica l'autenticità del messaggio ricevuto

o Contenuto inalterato

o Da una fonte autentica

o Tempestivo e nella sequenza corretta

- Può utilizzare la crittografia convenzionale

o Solo il mittente e il destinatario hanno la chiave necessaria

- Oppure un meccanismo di autenticazione separato

o Aggiunta del tag di autenticazione al messaggio in chiaro

L'algoritmo MAC è una tecnica crittografica a chiave simmetrica per fornire l'autenticazione dei messaggi. Per stabilire il processo di MAC, il mittente e il destinatario condividono una chiave simmetrica K. In sostanza, il MAC è una somma di controllo crittografata generata sul messaggio sottostante e inviata insieme a un messaggio per garantirne l'autenticazione.

Il processo di utilizzo del MAC per l'autenticazione è illustrato nell’immagine seguente.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

39 Cerchiamo ora di comprendere l'intero processo in dettaglio:

- Il mittente utilizza un algoritmo MAC pubblicamente noto, inserisce il messaggio e la chiave segreta K e produce un valore MAC.

- Come l'hash, anche la funzione MAC comprime un input arbitrariamente lungo in un output di lunghezza fissa. La differenza principale tra hash e MAC è che il MAC utilizza la chiave segreta durante la compressione.

- Il mittente inoltra il messaggio insieme al MAC. In questa sede si assume che il messaggio sia inviato in chiaro, poiché si tratta di fornire l'autenticazione dell'origine del messaggio e non la riservatezza. Se è richiesta la riservatezza, il messaggio deve essere crittografato.

- Quando riceve il messaggio e il MAC, il ricevitore inserisce il messaggio ricevuto e la chiave segreta condivisa K nell'algoritmo MAC e computa nuovamente il valore MAC.

- Il ricevitore verifica ora l'uguaglianza tra il MAC appena calcolato e quello ricevuto dal mittente. Se coincidono, il destinatario accetta il messaggio e si assicura che sia stato inviato dal mittente previsto.

- Se il MAC calcolato non corrisponde a quello inviato dal mittente, il destinatario non può stabilire se è il messaggio a essere stato alterato o se è l'origine a essere stata falsificata. In definitiva, il destinatario assume con sicurezza che il messaggio non sia autentico.

*Limitazioni del MAC*

Esistono due limitazioni principali del MAC, entrambe dovute alla natura simmetrica del suo funzionamento.

1) Stabilire un segreto condiviso.

- Può fornire l'autenticazione dei messaggi tra utenti legittimi prestabiliti che dispongono di una chiave condivisa.

- Ciò richiede la creazione di un segreto condiviso prima dell'uso del MAC.

2) Impossibilità di fornire il non ripudio

- La non ripudiabilità è la garanzia che l'originatore di un messaggio non possa negare messaggi e impegni o azioni precedentemente inviati.

- La tecnica MAC non fornisce un servizio di non ripudio. Se il mittente e il destinatario sono coinvolti in una disputa sull'origine del messaggio, i MAC non possono fornire una prova che il messaggio sia stato effettivamente inviato dal mittente.

- Anche se nessuna terza parte può calcolare il MAC, il mittente potrebbe negare di aver inviato il messaggio e sostenere che il destinatario lo ha falsificato, poiché è impossibile determinare quale delle due parti ha calcolato il MAC.

Similmente, gli algoritmi di hash sicuro, noti anche come SHA 

(Secure Hash Algorithms), sono una famiglia di funzioni

crittografiche progettate per garantire la sicurezza dei dati.

Funzionano trasformando i dati con una funzione hash: un

algoritmo che consiste in operazioni bitwise (bit per bit),

addizioni in modulo e funzioni di compressione.

La funzione hash produce quindi una stringa di dimensioni fisse

che non assomiglia affatto all'originale. Questi algoritmi sono

progettati per essere funzioni unidirezionali, il che significa che

una volta trasformati nei rispettivi valori hash, è praticamente

impossibile ritrasformarli nei dati originali. Alcuni algoritmi di

interesse sono SHA-1, SHA-2 e SHA-3, ognuno dei quali è stato

successivamente progettato con una crittografia sempre più

forte in risposta agli attacchi degli hacker. SHA-0, ad esempio, è

ormai obsoleto a causa delle vulnerabilità ampiamente esposte.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

40 Un'applicazione comune di SHA è la crittografia delle password, in quanto il lato server deve tenere traccia solo del valore hash di un utente specifico, piuttosto che della password vera e propria. Ciò è utile nel caso in cui un utente malintenzionato si introduca nel database, in quanto troverà solo le funzioni hash e non le password effettive, per cui se dovesse inserire il valore hash come password, la funzione hash lo convertirebbe in un'altra stringa e quindi negherà l'accesso.

Inoltre, gli SHA presentano l'effetto valanga, per cui la modifica di pochissime lettere crittografate provoca un grande cambiamento nell'output; o al contrario, stringhe drasticamente diverse producono valori hash simili. Questo effetto fa sì che i valori hash non forniscano alcuna informazione sulla stringa in ingresso, come ad esempio la sua lunghezza originale. Inoltre, gli SHA sono utilizzati anche per rilevare la manomissione dei dati da parte degli aggressori: se un file di testo viene leggermente modificato e in modo appena percettibile, il valore hash del file modificato sarà diverso dal valore hash del file originale e la manomissione sarà piuttosto evidente.

Le proprietà di una funzione hash sono le seguenti:

● Applicato a dati di qualsiasi dimensione

● ���� produce un risultato di lunghezza fissa.

● ����(����) è relativamente facile da calcolare per qualsiasi dato ����.

● Proprietà unidirezionale

○ È computazionalmente inefficace trovare ���� tale che ����(����) = ℎ

● Debole resistenza alle collisioni

○ (dato ����) è computazionalmente impossibile trovare y ≠ x tale che ����(����) = ����(����)

● Forte resistenza alle collisioni

○ Computazionalmente impossibile trovare qualsiasi coppia (����, ����) tale che ����(����) = ����(����)

● Due approcci di attacco

○ Crittoanalisi

■ Sfruttare la debolezza logica negli algoritmi

○ Attacco a forza bruta

■ Prova molti input

■ Forza proporzionale alla dimensione del codice hash

● Algoritmo hash SHA più utilizzato

○ SHA-1 fornisce hash a 160 bit

○ Più recenti SHA-256, SHA-384, SHA-512 offrono dimensioni e sicurezza migliorate

L'asimmetrico è una forma di crittosistema in cui la crittografia e la decrittografia vengono eseguite utilizzando chiavi diverse: la chiave pubblica (nota a tutti) e la chiave privata (chiave segreta). Questa è nota come crittografia a chiave pubblica.

*Caratteristiche della chiave di crittografia pubblica:*

- La crittografia a chiave pubblica è importante perché non è possibile determinare la chiave di decrittografia conoscendo solo l'algoritmo crittografico e la chiave di crittografia.

- Una delle due chiavi (pubblica e privata) può essere utilizzata per la crittografia e l'altra per la decrittografia.

- Grazie al sistema di crittografia a chiave pubblica, le chiavi pubbliche possono essere liberamente condivise, consentendo agli utenti un metodo facile e conveniente per crittografare i contenuti e verificare le firme digitali, mentre le chiavi private possono essere tenute segrete, garantendo che solo i proprietari delle chiavi private possano decifrare i contenuti e creare firme digitali.

- Il crittosistema a chiave pubblica più utilizzato è RSA (Rivest-Shamir-Adleman). La difficoltà di trovare i fattori primi di un numero composto è la spina dorsale di RSA.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

41 *Esempio*:

Le chiavi pubbliche di ogni utente sono 

presenti nel Registro delle chiavi pubbliche. Se

B vuole inviare un messaggio riservato a C,

allora B cripta il messaggio utilizzando la

chiave pubblica di C. Quando C riceve il

messaggio da B, C può decriptarlo utilizzando

la propria chiave privata. Quando C riceve il

messaggio da B, può decifrarlo utilizzando la

propria chiave privata. Nessun altro

destinatario oltre a C può decifrare il

messaggio perché solo C conosce la sua chiave

privata.

*Componenti della crittografia a chiave pubblica:*

- Testo in chiaro / plaintext:

o È il messaggio leggibile o comprensibile. Questo messaggio viene dato in ingresso

all'algoritmo di crittografia.

- Testo cifrato / ciphertext:

o Il testo cifrato viene prodotto come output dell'algoritmo di crittografia. Questo messaggio non è semplicemente comprensibile.

- Algoritmo di crittografia:

o L'algoritmo di crittografia viene utilizzato per convertire il testo in chiaro in testo cifrato. - Algoritmo di decifrazione:

o Accetta in ingresso il testo cifrato e la chiave corrispondente (chiave privata o chiave pubblica) e produce il testo in chiaro originale.

- Chiave pubblica e privata:

o Una chiave privata (chiave segreta) o pubblica (nota a tutti) viene utilizzata per la

crittografia e l'altra per la decrittografia.

*Caratteristiche:* 

1. Creazione di coppie di chiavi semplice dal

punto di vista computazionale

2. Il mittente, che conosce la chiave pubblica,

può crittografare i messaggi con facilità

computazionale per crittografare i messaggi

3. Il destinatario, che conosce la chiave privata,

può decifrare il testo per decifrare il testo

cifrato

4. Computazionalmente non fattibile per

l'avversario determinare la chiave privata

dalla chiave pubblica

5. Computazionalmente non fattibile per

l'avversario recuperare il messaggio originale

6. Utile se una delle due chiavi può essere usata

per ogni ruolo

*Punti deboli della crittografia a chiave pubblica:*

- La crittografia a chiave pubblica è vulnerabile agli attacchi Brute-force.

- Questo algoritmo fallisce anche quando l'utente perde la sua chiave privata, allora la Crittografia a chiave pubblica diventa l'algoritmo più vulnerabile.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

42 - La crittografia a chiave pubblica è anche debole nei confronti degli attacchi man in the middle. In questo attacco una terza parte può interrompere la comunicazione a chiave pubblica e quindi modificare le chiavi pubbliche.

- Se la chiave privata dell'utente utilizzata per la creazione del certificato a un livello superiore nella gerarchia dei server PKI (Public Key Infrastructure) viene compromessa o divulgata

accidentalmente, è possibile anche un "attacco man-in-the-middle", che rende completamente insicuro qualsiasi certificato subordinato. Questo è anche il punto debole della crittografia a chiave pubblica.

*Applicazioni della crittografia a chiave pubblica:*

- Crittografia/decrittografia:

La riservatezza può essere ottenuta utilizzando la crittografia a chiave pubblica. In questo caso il testo in chiaro viene crittografato utilizzando la chiave pubblica del destinatario. In questo modo si garantisce che nessuno, al di fuori della chiave privata del destinatario, possa decifrare il testo cifrato.

- Firma digitale:

La firma digitale serve per l'autenticazione del mittente. In questo caso il mittente cripta il testo in chiaro utilizzando la propria chiave privata. Questo passaggio assicura l'autenticazione del mittente perché il destinatario può decifrare il testo cifrato solo con la chiave pubblica del mittente. - Scambio di chiavi:

Questo algoritmo può essere utilizzato sia per la gestione delle chiavi che per la trasmissione sicura dei dati.

I numeri casuali sono utili per una serie di scopi, come la generazione di chiavi di crittografia dei dati, la simulazione e la modellazione di fenomeni complessi e la selezione di campioni casuali da insiemi di dati più ampi. Sono stati utilizzati anche dal punto di vista estetico, ad esempio nella letteratura e nella musica, e sono ovviamente sempre più popolari nei giochi e nelle scommesse. Quando si parla di numeri singoli, un numero casuale è un numero estratto da un insieme di valori possibili, ognuno dei quali è ugualmente probabile, cioè una distribuzione uniforme. Quando si parla di una sequenza di numeri casuali, ogni numero estratto deve essere statisticamente indipendente dagli altri.

Con l'avvento dei computer, i programmatori hanno riconosciuto la necessità di introdurre la casualità in un programma informatico. Tuttavia, per quanto possa sembrare sorprendente, è difficile che un computer faccia qualcosa per caso. Un computer segue le sue istruzioni alla cieca ed è quindi completamente prevedibile. Esistono due approcci principali per generare numeri casuali utilizzando un computer: I generatori di numeri pseudo-casuali (PRNG) e i generatori di numeri casuali veri (TRNG). Questi approcci hanno caratteristiche molto diverse e ognuno ha i suoi pro e i suoi contro.

In sostanza, i PRNG sono algoritmi che utilizzano formule matematiche o semplicemente tabelle precalcolate per produrre sequenze di numeri che appaiono casuali.

La differenza di base tra i PRNG e i TRNG è facile da capire se si confrontano i numeri casuali generati dal computer con i lanci di un dado. Poiché i PRNG generano numeri casuali utilizzando formule matematiche o elenchi precalcolati, l'uso di un PRNG corrisponde al lancio di un dado molte volte e alla scrittura dei risultati. Ogni volta che si chiede il lancio di un dado, si ottiene il successivo dell'elenco. In effetti, i numeri sembrano casuali, ma in realtà sono predeterminati. I TRNG funzionano facendo in modo che un computer lanci effettivamente il dado o, più comunemente, utilizzano qualche altro fenomeno fisico che è più facile da collegare a un computer rispetto a un dado.

I PRNG sono efficienti, cioè possono produrre molti numeri in poco tempo, e deterministici, cioè una data sequenza di numeri può essere riprodotta in un secondo momento se si conosce il punto di partenza della sequenza. L'efficienza è una caratteristica positiva se l'applicazione ha bisogno di molti numeri, mentre il determinismo è utile se è necessario riprodurre la stessa sequenza di numeri in un secondo momento. I PRNG sono tipicamente periodici, il che significa che la sequenza si ripeterà. Anche se la periodicità non è

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

43 quasi mai una caratteristica desiderabile, i PRNG moderni hanno un periodo così lungo che può essere ignorato per la maggior parte degli scopi pratici.

Queste caratteristiche rendono i PRNG adatti ad applicazioni in cui sono richiesti molti numeri e in cui è utile che la stessa sequenza possa essere riprodotta facilmente. Esempi popolari di tali applicazioni sono le applicazioni di simulazione e modellazione. I PRNG non sono adatti ad applicazioni in cui è importante che i numeri siano davvero imprevedibili, come la crittografia dei dati e il gioco d'azzardo.

Rispetto ai PRNG, i TRNG estraggono la casualità da fenomeni fisici e la introducono in un computer. Si può immaginare che si tratti di un dado collegato a un computer, ma in genere si utilizza un fenomeno fisico che è più facile da collegare a un computer rispetto a un dado. Il fenomeno fisico può essere molto semplice, come le piccole variazioni nei movimenti del mouse o nel tempo che intercorre tra la pressione dei tasti. In pratica, però, bisogna fare attenzione alla fonte che si sceglie. Ad esempio, può essere complicato utilizzare le sequenze di tasti in questo modo, perché spesso le sequenze di tasti vengono bufferizzate dal sistema operativo del computer, il che significa che vengono raccolte diverse sequenze di tasti prima di essere inviate al programma che le attende. Per il programma in attesa dei tasti, sembrerà che i tasti siano stati premuti quasi simultaneamente e forse non c'è molta casualità.

Esercizi Lezione 5

1) Crack the Hash: Testo, aiuti e soluzione

Testo

*It’s happened again! Some of our beloved friends at linkedin.com forgot to salt their password hashes. Due to a rather interesting SQL injection issue, a hacker group has published the following MD5 hash online! Can you crack it?*

*Hash: `365d38c60c4e98ca5ca6dbc02d396e53`*

*Hint. Use a md5 cracker*

Aiuto:

1) The description suggests that the MD5 has been already cracked and that you can find the solution online. Write on your research engine (e.g., Google) “MD5 cracker”. You can find several web pages where you insert an MD5, and they give you a solution.

Soluzione

Molto semplice, letteralmente abbiamo una hash MD5 e quindi la traduciamo.

*We have the following MD5 hash:*

*365d38c60c4e98ca5ca6dbc02d396e53*

*The description suggests us to use a tool.*

*https://crackstation.net/*

*the password is password12345*

Normalmente, l'algoritmo crittografico MD5 non è sempre reversibile.

È possibile criptare una parola in MD5, ma non è possibile creare la funzione inversa per decriptare un hash MD5 nel testo in chiaro (perlomeno, non di tutte). Questa è la proprietà di una funzione di hash. Pertanto, purtroppo, non è neanche possibile passare per un’implementazione Python universale (è anche possibile

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

44 realizzare un algoritmo, ma potrebbe non funzionare mai, in quanto tocca ricercare tutte le possibili combinazioni e, casualmente per forza bruta, si potrebbe arrivarvi; tuttavia, può avere un tempo di esecuzione lungo *quanto la vita dell’universo*).

Una possibile implementazione totalmente bruteforce prova tutte le possibili combinazioni e funziona in alcuni casi limitati, come riporto di seguito:

*import hashlib* #library used for MD5/SHA, etc.

*def md5reversehashing(text):*

*for i in range(0, 100000000):* # 100000000 is the number of possible combinations

*hash\_object = hashlib.md5(str(i).encode())* # convert the number to a string and encode it *if hash\_object.hexdigest() == text:* # if the hash is equal to the text

*print(i)* # print the number

*break*

#Driver Code

*md5reversehashing('e10adc3949ba59abbe56e057f20f883e')* # 123456

#md5reversehashing('365d38c60c4e98ca5ca6dbc02d396e53') # This one coming from the challenges is not working

2) Ready XOR Not: Testo, aiuti e soluzione

Testo

*original data: "El Psy Congroo"*

*encrypted data: "IFhiPhZNYi0KWiUcCls="*

*encrypted flag: "I3gDKVh1Lh4EVyMDBFo="*

*The flag is a composition of two names (two animals (?)).*

Aiuti:

1) You can notice the following:

o The encrypted strings are in base64. Decode them!

o All the strings have the same length

o The challenge is named “xor” … this should suggest you something

2) To obtain the flag, you need to decrypt it.

Since we have an example of plaintext and its encrypted counterparts, you can use the “xor” property to find the key: remember, you need to first represent the characters as “ascii numbers”, xor them, and reconvert it into ascii char.

Soluzione

*import base64*

*original\_data = "El Psy Congroo"*

*encrypted\_data = "IFhiPhZNYi0KWiUcCls="*

*encrypted\_flag = "I3gDKVh1Lh4EVyMDBFo="*

# we can note that all the strings have the same length

# since we have an example of encryption, and we know that this is a xor,

# we can simply try to obtain the key in the example, and apply it to the

# crypted flag

# Usual function to decode base64 strings

*def base64tostring(text):*

*return base64.b64decode(text).decode('utf-8', errors="ignore")*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

45 #decode the encryption from base64 *enc\_data= base64tostring(encrypted\_data)*

*enc\_flag= base64tostring(encrypted\_flag)*

#we know apply the xor to obtain the key

# So, we take the position of each character of “x” and “y” and then, iterating,

# using the zip() function, which returns a zip object (an iterator of tuples where the first item in each #passed iterator is paired together), and then the second item in each passed iterator are paired # together etc. Given the ordinary XOR logic, we take the key via the XOR between raw and encoded data *key = ' '.join([chr(ord(x) ^ ord(y)) for x, y in zip(original\_data, enc\_data)])*

*print('key:\t',key)*

#this seems a reasonable key

*flag = ' '.join([chr(ord(x) ^ ord(y)) for x, y in zip(enc\_flag, key)])*

*print("Flag:\t", flag)*

#flag: FLAG=Alpacaman

3) Top: Testo, Aiuti e soluzione

Testo

*Perfectly secure. That is for sure! Or can break it and reveal my secret?*

*We are given an encryption script and a file which is encrypted with it*

Ci vengono appunto dati due file: un file Python “top.py”, che riportiamo sotto e un file criptato con quello, nello specifico “top\_secret” senza estensione. Qui di seguito, appunto, *top.py*.

*import random*

*import sys*

*import time*

# Let’s import the main modules, especially “time” to take the current system time, “random” # to use the time for the seed of the randomness choice and then “sys”, in order to

# open in writeback mode and writing the bytes with algorithm here

*cur\_time = str(time.time()).encode('ASCII')*

# So, the idea is taking the current time and encode the string in ASCII

*random.seed(cur\_time)* # then “plant” the seed for the randomness

*msg = input('Your message: ').encode('ASCII')* # The input, also, is in ASCII

*key = [random.randrange(256) for \_ in msg]*

# We put the random range up until all the 256 characters of ASCII Encoding, looping

# in all of the message

# What we are doing here is creating a zip function, so we apply the XOR function

# and then XORing the entire message + the current time with the XOR sign (0x88) of the current # time (in length) + the key itself; we can see we repeat the same data X number of times; that’s the # vulnerability

*c = [m ^ k for (m,k) in zip(msg + cur\_time, key + [0x88]\*len(cur\_time))]*

# In the end, we just write the file entirely in the buffer

*with open(sys.argv[0], "wb") as f:*

*f.write(bytes(c))*

Il file “top\_secret” è composto dal seguente testo:

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

46 f¸,¹ÚfœX‚^ß÷” qDµ8{»tI±y†•Vu³Þ[H° Å†ýã" ¿ô’oVŽ¥©—¹½¹»¿¹±¹»»¦°¿º°¿½º

Aiuti:

1) We are given a script that generated the encrypted file. The challenge here is to spot potential vulnerabilities. The first thing to do is to debug it; we can find the following steps:

● Set a seed with a time.

● Get an input string

● Define a key for the string

● Encryption: the final output is the concatenation of both encrypted message and encrypted time! 2) The time characters are encrypted with some fixed values (0x88). Since we are talking about a xor operation, we can retrieve the original time. To know the length, get a time variable and see how many characters we do have.

Then, once we have the “plain” time, we ca regenerate the original sequence of characters that compose the key … and apply it to the encrypted text. We finally have the flag.

Soluzione

#the encryption is composed by a xor between a character and a key.

#the message is the concatenation of msg + cur\_time

#the key is the concatenation of the list of keys + list of 0x88

####--------RESOLUTION -----------------

#we know that |msg| = |key|, and |cur\_time| = |[0x88]|

#we can use the xor property to retrieve the cur\_time of the execution

#useful to use "rb" to read in binary and open correctly the file

#otherwise, it won't work (infact, as a no extension file, is considered binary)

*with open("top\_secret", "rb") as f:*

*secret = f.read()*

#let's try to think about the algorithm

#there's a message which is encoded in ascii,

#a key selected randomly using the current time as the seed

#and a xor function between the message and the key

#and 0x88, multiplied X times with the length of the current time

#we can see that the time length is added many times

#so we can xor it subtracting from the ‘secret’ string the ‘cur\_time’ length, given how many times is #repeated, also knowing the XOR function has a vulnerability because of it

*sec\_time = secret[-len(cur\_time):]*

*plain\_time = ' '.join([chr(m ^ k) for (m, k) in zip(sec\_time, [0x88]\*len(cur\_time))])*

*print(f"Plain time:\t{plain\_time}")* #what we printed seems a correct datatime format

# we now leverage on the pseudonumber vulnerabilities

# the algorithm set a seed, so it is not random the generator.

# So, we plant the seed as the plain\_time encoded in ASCII in order to correctly read it

*random.seed(plain\_time.encode("ASCII"))*

# get the keys, so we iterate on each key into the secret string subtracting the current time # and then reapplying the given XOR function to the new plain text, printing it

*keys\_secret = [random.randrange(256) for \_ in secret[:-len(cur\_time)]]*

*plain\_text = ' '.join([chr(m ^ k) for (m, k) in zip(secret[:-len(cur\_time)], keys\_secret)])*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

47 *print(plain\_text)* #flag reached

# Here is your flag: 34C3\_otp\_top\_pto\_pot\_tpo\_opt\_wh0\_car3s

Stampa:

*75*

*Plain time: 1513719133.8728752*

*Here is your flag: 34C3\_otp\_top\_pto\_pot\_tpo\_opt\_wh0\_car3s*

4) Repeated XOR: Testo e soluzione

Testo

*There is a secret passcode hidden in the robot's "history of cryptography" module. But it's encrypted! Here it is, hex-encoded: encrypted.txt. Can you find the hidden passcode?*

*Hint:*

*Like the title suggests, this is repeating-key XOR. You should try to find the length of the key - it's probably around 10 bytes long, maybe a little less or a little more.*

*#Follow the following procedure*

*STEP 1: Key length identification*

*#1.1 set the key length to test*

*#1.2 shift the secret string by key\_length*

*#1.3 count the number of characters that are the same in the same position*

*between the original secret and its shifted version*

*#1.4 take the highest frequency over different key length by repeating 1.1 - 1.3*

*#STEP 2. Cryptoanalysis*

In coppia a questo, come accennato dal testo, si presenta il file “encrypted.txt” con un lungo testo in esadecimale (es. di alcuni caratteri: 2AE6BD0B6ECE21162AF4B20C6CC2 e via così).

Soluzione

#the goal of this challenge is to leverage on the xor weaknesses.

#we can read the file first

#the file is in hex encoded; it could be good to bring it in a proper form

#we know that FF is 256, i.e., we can represent the text in a decimal format,

#where each number can be encoded in ascii

*with open("encrypted.txt", 'r') as file:*

*secret\_hex = file.read()*

*def hex2dec(text):*

*res = []*

*for i in range(len(text)//2):* # we take each couple of bytes and then convert into “int” in 16 bytes form #get the current pair of hex

*curr = text[i\*2:(i+1)\*2]*

#convert to int the current paired two bytes string form and then express it in sixteen bytes each; #this way, we can convert the 16-bit form into the 10-bit form required (hex to decimal) *res.append(int(curr, 16))*

*return res*

*secret = hex2dec(secret\_hex)*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

48 #STEP 1: Key length identification #shift string -> it allows us the comparison

# To make a shift, we can split the text starting from the key length and then iterate to the end # summing the iteration from the beginning up until the key length

*def shift(text, key\_length):*

*return text[key\_length:] + text[:key\_length]*

#freq counter

#we compare the original sentence with its shifted version

#we count the amount of same characters in the same position

*def freq\_counter(s1, s2):*

*freq = sum([1 for (x, y) in zip(s1, s2) if x == y])*

# Note we sum one using a zip between s1 and s2 only if in the same position (so, to iterate on both) *return freq*

#test over different lengths.

#the hints suggests us that the length is circa 8. So, we look between [5, +15]

*for kl in range(5, 16):*

*print(f"Length:\t{kl}\tFreq:\t{freq\_counter(secret, shift(secret, kl))}")*

#the highest value is with length = 8.

#STEP 2: Cryptoanalysis

#split the corpus in 8-chars’ lengths

*def splitter(text, key\_length):*

*res = []*

*for i in range(key\_length):*

*res.append(text[i::key\_length])* # we take the i-th char of each block

# then, we return a string split according to the parameter “key\_length” passed

*return res*

*secret\_ = splitter(secret, 8)* # At this point, we split the string

#we need to define a method that show us the k-th most frequent character in

#a given string

*from collections import Counter*

*def k\_char(text, k):*

#use the counter 🡪 Counter is an unordered collection where elements are stored as Dict keys and their # count as dict value. Counter elements count can be positive, zero or negative integers. However, there is # no restriction on its keys and values. Although values are intended to be numbers, but we can store # other objects too.

*freq = Counter(text)* # We find the frequency of the text passed, counting how much it appears

# Here, we order the data collection like a list and then return a sorted list, using an iterable (the items # of the frequency count), a function (made shortened thanks to lambda, which sets the “key” parameter) #making it appear in reverse order

*ordered = sorted(freq.items(), key=lambda x: x[1], reverse=True)*

*return ordered[k][0]* # we can return the ordered elements, started from the first (this is like \*ordered)

## we can now see the top N frequent words

#print(k\_char(secret))

#we now work on the Cryptoanalysis, based on each column of the matrix M[secret//8 X 8] #we can first assume that the most common character in each column is the space ' '.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

49 *key\_sec = [k\_char(secret\_[0], 0), k\_char(secret\_[1], 0), k\_char(secret\_[2], 0), k\_char(secret\_[3], 0), k\_char(secret\_[4], 0), k\_char(secret\_[5], 0), k\_char(secret\_[6], 0), k\_char(secret\_[7], 0)]*

# What we are doing here is seeing the frequency of each of the eight characters of the secret # given the position and then having a list of the frequency of each one

#xor the key: to do that, we use the kth character in the list and then XOR it with the space character *real\_key = [k ^ ord(' ') for k in key\_sec]*

#decode the secret

*real\_message = ' '*

# Enumerate() method adds a counter to an iterable and returns it in a form of enumerating object. This # enumerated object can then be used directly for loops or converted into a list of tuples using the list() #method. Syntax 🡪 enumerate(iterable, start=0) 🡪 so, it iterates on secret

*for i, c in enumerate(secret):*

*key\_pos = i % 8*

# we then apply the XOR function as seen with “real\_key” but this time XORing the key in each position *real\_message+= chr(c ^ real\_key[key\_pos])*

*print(real\_message)*

#your flag is: 8eb31c92334eac8f6dacfbaaa5e40294a31e66e0

Altra soluzione (scritta da me)

Si può anche confrontare un buon writeup (completo di spiegazione estesa anche del ragionamento logico completo) al link: https://ehsan.dev/pico2014/cryptography/repeated\_xor.html

#we do have a txt file and we're gonna read it

*with open('encrypted.txt', 'r') as f:*

*data = f.read()*

*data=data.replace(' ', '').replace('\n', '')* #removing the \n from the end of the line and converting to ascii

#we do know the file is hex encoded, so we decode its bytes into

#decimal format and then we convert it to ascii

*def hex\_to\_dec(text):*

*result=[]*

#we take each pair of bytes and convert it to decimal (2), looping into hex (16)

*for i in range(0, len(text), 2):*

#taking each pair of bytes in the text and converting it to decimal

*current=" ".join(text[i:i+2])*

#appending the result considering each pair corresponds to a byte in hex

*result.append(int(current, 16))*

*return result*

*decoded\_data=hex\_to\_dec(data)* #we decode the data (when printed, it seems like a map of integers)

#We have a repeated xor problem here, so we have to make a frequency

#analysis in order to understand which characters are the most frequent ones

#and then we can guess the key

#First, we have to guess the key length; infact, if the key is shorter than the message,

#the key will repeat itself many times in order to cover the whole message.

#So, after converting to integers, we duplicate the key and xor

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

50 #the message with the duplicated key.

#Step 1 - Key length identification (we know it's probably 10 bytes long)

#Here, we do need to make an educate guess using the frequency analysis and a shift function, #just to play out with some different lengths

*def shift(text, n):*

*return text[n:] + text[:n]* #we shift the text by n bytes and sum them together

*def count\_same(a, b):* #we count the same bytes in the text and simply return it

*count=0*

*for x, y in zip(a,b):*

*if x == y:*

*count+=1*

*return count*

*def guess\_key\_length(text, key\_length):*

#what we are doing here is making a range based on the key length

#then counting the same bytes in the text and making a shift of "n" bytes

#in order to try all of the possible combinations

#we return the key length with the highest count of same bytes

#the higher the count, the more likely the key length is correct

*return max(range(1, key\_length), key=lambda n: count\_same(text, shift(text, n)))*

*print(guess\_key\_length(decoded\_data, 10))* #we print the key length, which is 8

#Up until here, we completed step (1)

#Step 2 - Cryptoanalysis

#We do know the key length, so we can guess the XOR is made with 8 bytes block in mind #Remember that the key is repeated every 8 bytes within the text,

#so the idea is to take the most frequent characters every 8 bytes based on the key length *from collections import Counter* #we import this in order to count the most frequent characters

#Given it should be English text, the most frequent characters would be

#e, t, a, o, i, n, s, h (in this order), thinking also ' ' (the space) should be the most frequent one

#Because XOR is its own inverse, we can find the key by XORing cipher text and known plain text #(given a character, in whatever column it appears, given the fixed key length, the XOR always gives the #same result). Thus we can find the key if we know the most common character in english and the most #common character in the nth column.

*def most\_frequent\_chars(text, key\_length):*

#we split the text into blocks of 8 bytes

#then we count the most frequent characters in each block

#and we return the result (*i::key\_length* represents every element of text in indexes from *i* to *key\_length*) *return [Counter(text[i::key\_length]).most\_common(1)[0][0] for i in range(key\_length)]* #we return the most frequent character (hence the (1)) in each block (given the key length) *key\_secure=most\_frequent\_chars(decoded\_data, 8)*

*print(key\_secure)* #we print the map of the most frequent characters in the text, split by 8

#now, we need to find the key, which is the XOR of the most frequent characters and the plain text itself *real\_key = [k ^ ord(' ') for k in key\_secure]*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

51

#decode the secret and print the flag

*for i in range(len(decoded\_data)):*

*decoded\_data[i] ^= real\_key[i % len(real\_key)]* #we xor the data with the key given the 8 bytes key split

*print(bytes(decoded\_data).decode('ascii'))*

#we print the final result (given it is the flag and all of the other text) as ascii

Lezione 6: Autenticazione dell’utente/User authentication (Conti)

L'autenticazione dell'utente verifica l'identità di chi tenta di accedere a una rete o a una risorsa informatica, autorizzando un trasferimento di credenziali da uomo a macchina durante le interazioni su una rete per confermare l'autenticità dell'utente. Il termine si contrappone all'autenticazione automatica, che è un metodo di autenticazione automatizzato che non richiede l'input dell'utente.

Essa aiuta a garantire che solo gli utenti autorizzati possano accedere a un sistema, impedendo agli utenti non autorizzati di accedere e potenzialmente danneggiare i sistemi, rubare informazioni o causare altri problemi. Quasi tutte le interazioni tra esseri umani e computer, ad eccezione degli account guest e di quelli che si collegano automaticamente, prevedono l'autenticazione dell'utente. Autorizza l'accesso su reti cablate e wireless per consentire l'accesso a sistemi e risorse collegati in rete e a Internet.

L'autenticazione dell'utente è un processo semplice e si compone di due operazioni:

1) Identificazione. Gli utenti devono dimostrare chi sono.

2) Verifica. Gli utenti devono dimostrare di essere chi dicono di essere e devono dimostrare di essere autorizzati a fare ciò che stanno cercando di fare.

L'autenticazione dell'utente può essere semplice, come richiedere all'utente di digitare un identificativo univoco, come l'ID utente, insieme a una password per accedere a un sistema. Ma può anche essere più complessa, ad esempio richiedendo all'utente di fornire informazioni su oggetti fisici o sull'ambiente o persino di compiere azioni, come mettere un dito su un lettore di impronte digitali.

Un esempio di autenticazione:

○ User real name: Alice Toklas

○ User ID: ABTOKLAS

○ Password: A.df1618hJb

Queste informazioni sono memorizzate in un sistema e solamente Alice può accedere con queste credenziali. Tuttavia, se non ben protette, gli attaccanti possono usare queste informazioni comunque.

Ci sono 4 modi per autenticare l’identità di un utente, per esempio basato sull’individuo: ○ Dati che conosce - ad es. password, PIN

○ Dati che possiede - ad es. chiave, token, smartcard

○ Dati che dimostrano chi è (biometria statica) - ad es. impronta digitale, retina

○ Dati che dimostrano una caratteristica propria (biometria dinamica) - ad es. voce, firma Essi possono essere usati insieme o in modo combinato e possono tutti fornire l’autenticazione all’utente e tutti hanno problemi. Un metodo classico di login sono i classici nome utente e password, confrontati dal sistema e poi confrontati per verificare corrispondano ai propri. Tramite l’ID dell’utente, si verificano i suoi privilegi.

Alcuni esempi di attacchi e vulnerabilità delle password:

- *Keystroke timing analysis*, in cui si legge la tempistica con cui i tasti vengono premuti e si analizza in millisecondi la misura con cui questi vengono usati. Similmente, si possono carpire informazioni sul movimento dei tasti da un punto di vista fisico.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

52 - *URL Hijacking/Typosquatting*, partendo dal fatto che può succedere che un attaccante acceda fisicamente ad una macchina, magari usando un cavo invisibile e in qualche modo modificando le informazioni utente. In questo modo, mandiamo una mail con un link fasullo; così facendo, andando sulla pagina si ha l’URL corretto, ma la pagina ha un link finto e non sarebbe proprio riconoscibile. Similmente, usiamo il *tabnagging*, quindi sfruttiamo le vecchie schede per inserire link malevoli oppure lo *UI redressing/iFrame overlay*, per inserire link malevoli su bottoni/elementi grafici.

- *Offline dictionary attack*: l’attaccante ha la hash della password bersaglio e cerca di ottenerla, sulla base di password comuni o informazioni note sul bersaglio. Si deve cercare di proteggere queste informazioni, possibilmente nascondendole in un posto sicuro.

- *Specific account attack*: l’attaccante bersaglia un account specifico e cerca di indovinare la password corretta, sulla base di password comuni o informazioni note sul bersaglio. Come contromisura si possono introdurre dei meccanismi di *lockout*, quindi lasciare ad esempio escluso un utente dopo un certo numero di tentativi di accesso.

- *Popular password guessing*: l'aggressore prova le password più diffuse

contro un'ampia gamma di account, sapendo che gli utenti tendono a scegliere password semplici, corte oppure ad inserire informazioni sensibili (affetti, amici, ecc.). Come contromisura, si cerca di non usare password comuni (possibilmente anche autogenerate, quindi usando un meccanismo che cerca di creare di un’apposita lunghezza scelta con o senza caratteri speciali, già prestabilita) o evitando di usare sottosequenze/pattern di caratteri utili.

- *Workstation hijacking/dirottamento della stazione di lavoro*: L'aggressore aspetta che una workstation sia incustodita, al fine di prenderne possesso. Come contromisura, oltre a meccanismi di log-out automatici, si cerca di individuare comportamenti anomali (accessi strani, lasciare fuori l’utente dopo un certo tempo, ecc.).

- *Exploiting user mistakes/Sfruttare gli errori degli utenti*: Gli utenti tendono ad annotare le password Ad esempio, post-it vicino al dispositivo protetto e tendono ad avere dispositivi con password preconfigurate. Come contromisura, si deve cercare di istruire gli utenti (invitandoli ad adottare dei comportamenti tali da non diffondere informazioni utili all’esterno) ed usare meccanismi di autenticazione combinata (es. password e token).

- *Exploiting multiple password uses*: Gli utenti tendono ad usare la stessa password (o password simili) in diversi sistemi. Se un attaccante indovina la password, può danneggiare molti sistemi. Come contromisura, si cerca anche qui di addestrare gli utenti a non adottare comportamenti vulnerabili (scegliendo password diverse e magari autogenerate ad esempio) e proibire il riutilizzo della password sui vari sistemi.

- *Password spraying attack*, per cui l'hacker tenta di autenticarsi utilizzando la stessa password su vari account prima di passare a un'altra password. Lo spraying di password è più efficace perché la maggior parte degli utenti di siti web imposta password semplici e la tecnica non viola le politiche di blocco poiché utilizza diversi account. Gli aggressori orchestrano lo spraying di password soprattutto nei siti Web in cui gli amministratori impostano una password standard predefinita per i nuovi utenti e gli account non registrati.

- *Electronic monitoring*: Se la password è comunicata tramite una rete, l’attaccante usa lo *sniffing* (si pone in ascolto sulla rete, cerca di carpire informazioni dai pacchetti sulla rete per rubare la password). Come contromisura, si cerca di usare canali sicuri/protetti anche da cortocircuiti oppure fisicamente buoni.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

53 - *Session hijacking*, quindi accedere in automatico ad una macchina in quanto già autenticati; meccanismi di log-off automatico. Anche per questo, si può usare l’*autenticazione a più fattori*. Se abilitata sul vostro account, un potenziale hacker può inviare una richiesta di accesso al vostro account solo al secondo fattore (normalmente, si tratta di un messaggio/notifica/dispositivo fisico). Gli hacker non avranno probabilmente accesso al vostro dispositivo mobile o all'impronta digitale, il che significa che saranno bloccati dal vostro account.

- *RDP Hacking (Remote Desktop Protocol*), facilmente ottenibile tramite manipolazione della rete, specie se non sicura oppure sfruttando buchi a livello di aggiornamento del software.

Siti ed indicazioni utili sulle password

- Have I Been Pwned: https://haveibeenpwned.com/

- Le password più comuni (classiche, *password* oppure *123456*): https://blog.dashlane.com/ten most-common-passwords/

o Sapendo tutte le password comuni, proviamo una bruteforce

- Usare un *password manager* (Bitwarden che consiglio personalmente) per salvarle tutte. Non usare file locali, Chrome per salvarle o applicazioni simili, file esterni o un quaderno per scriverle; è tutto molto scomodo e poco scalabile

- Le password non sono mai salvate in chiaro; in questo modo, il sysadmin/amministratore di sistema non saprà/non vedrà la password per quanto gestisca. Similmente, le password non spesso vengono cambiate; qualsiasi computazione si sia calcolata, è bene riutilizzarla per futuri calcoli.

- Esistono attacchi e challenge che sfruttano le password più comuni, tramite file cosiddetti “rockyou”, che contengono proprio quelle più comuni

Un meccanismo utilizzato sono le *hashed passwords*, quindi l’utente crea una nuova password ed essa viene combinata con un *salt* (dato random usato come input aggiuntivo di una *one-way function*, quindi una funzione facile da computare su ogni input ma difficile da invertire su un input casuale; essa permette di realizzare una funzione hash su dati, password o passphrase (insieme di parole/stringhe alfanumeriche usate per autenticazione)). L’ID, la password con la hash e il salt sono salvati su un file (password file) e queste funzioni hash sono designate per essere lente. Grazie al *salt*, per esempio un numero casuale, si complica il fatto di poter rubare la password.

Gli hash, quando vengono utilizzati per la sicurezza, devono essere lenti. Una funzione hash crittografica utilizzata per l'hashing delle password deve essere lenta da calcolare perché un algoritmo calcolato rapidamente potrebbe rendere più fattibili gli attacchi di forza bruta, soprattutto con la rapida evoluzione della potenza dell'hardware moderno. È possibile ottenere questo risultato rendendo lento il calcolo dell'hash utilizzando molte iterazioni interne o rendendo il calcolo intensivo della memoria.

Una funzione hash crittografica lenta 

ostacola questo processo, ma non lo

blocca, poiché la velocità di calcolo

dell'hash interessa sia gli utenti

intenzionati che quelli malintenzionati. È

importante raggiungere un buon

equilibrio tra velocità e usabilità delle

funzioni di hashing. Un utente ben

intenzionato non avrà un impatto

notevole sulle prestazioni quando tenta

un singolo login valido.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

54 Usiamo un salt per tre ragioni principali: 1) Prevenire la duplicazione delle password (se due utenti usano una stessa password, dei salt differenti possono produrre password diverse)

2) Aumentare la difficoltà degli attacchi a dizionario (se il salt ha ���� bit, allora il fattore sarà 2���� 3) Impossibile capire se la persona usa la stessa password in più sistemi

Alcuni attacchi comuni di *password cracking* sono i seguenti:

- Attacchi a dizionario 🡪 Provare ciascuna parola in un dizionario su un hash di un password file, provando tutte le password comuni e, se non ci sono corrispondenze, proviamo ogni possibile modifica (numeri, punteggiatura). Esse sono costose computazionalmente

- Rainbow table attack 🡪 È un metodo di cracking delle password che utilizza una tabella speciale (una "tabella arcobaleno") per decifrare gli hash delle password in un database. Le applicazioni non memorizzano le password in chiaro, ma le criptano utilizzando degli hash. Dopo che l'utente inserisce la propria password per accedere, questa viene convertita in hash e il risultato viene confrontato con gli hash memorizzati sul server per cercare una corrispondenza. Se corrisponde, l'utente viene autenticato e può accedere all'applicazione. La tabella arcobaleno si riferisce a una tabella precompilata che contiene il valore hash della password per ogni salt utilizzato durante il processo di autenticazione. Se gli hacker hanno accesso all'elenco degli hash delle password, possono decifrare tutte le password molto rapidamente con una tabella arcobaleno.

Infatti, gli utenti potrebbero scegliere password corte (facilmente indovinabili e i sistemi solitamente rifiutano password corte) oppure password indovinabili (quindi, gli attaccanti usano una lista di password similari, impiegandoci un’ora sui sistemi più veloci, con un solo successo per rubare i dati). Similmente, usando vari metodi per accedere (carte di credito, carte magnetiche, SIM, smart card, ecc.) o caratteristiche biologiche (firma, retina, voce, ecc.).

Per quanto riguarda l’uso di *memory card*, esse memorizzano ma non elaborano i dati. Normalmente sono delle carte a banda magnetica, ad esempio carta bancaria con una scheda di memoria elettronica, utilizzata da sola per l'accesso fisico, Può anche essere usata con password/PIN per l'uso del computer.

Gli svantaggi delle schede di memoria sono:

○ Necessitano di un lettore speciale (che aumenta il costo della soluzione di sicurezza). soluzione di sicurezza)

○ Problemi di perdita del token (non possiamo fidarci degli utenti)

○ Insoddisfazione dell'utente (non è totalmente approvato dagli utenti)

Le autenticazioni possono essere in locale o da remoto e, ovviamente, farlo su una rete può portare a vari problemi di intercettazione, replay (riutilizzo), ecc. Generalmente si usa la sfida-risposta: - L'utente invia l'identità

- L'host risponde con:

o un numero casuale ���� (detto anche nonce)

o Una funzione hash ℎ

o Una funzione f

- L'utente calcola ����(����, ℎ(����)) e invia di nuovo

- L'host confronta il valore dell'utente con il proprio valore calcolato, se corrisponde l'utente si autentica

- Protegge da una serie di attacchi

Possono essercene diverse: scan delle impronte digitali, autenticazione vocale, tramite il volto, OTP (codici temporanei/*One-Time Passcodes*, unici e legati ad un singolo account).

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Esercizi Lezione 6 55

1) Sniffing: Testo, aiuti e soluzione

Testo

*We sniffed a sensible http traffic.*

*Can you identify the password?*

*The attacked service is called bashNinja.*

*Hint. Use Wireshark.*

Ci viene dato un file in formato “.pcap”, che intende “packet capture”, normalmente usato da Wireshark, famoso *packet sniffer*, strumento di base per osservare i messaggi scambiati tra entità di protocollo. Come suggerisce il nome, esso copia passivamente (ossia “sniffa, annusa”) i messaggi che vengono inviati e ricevuti dal vostro computer; inoltre, mostra i contenuti dei vari campi di protocollo e dei messaggi catturati. Un packet sniffer è una entità passiva: osserva i messaggi inviati e ricevuti dalle applicazioni e dai protocolli in esecuzione sul vostro computer, ma non manda mai egli stesso dei pacchetti. Allo stesso modo, i pacchetti che riceve non sono mai stati inviati esplicitamente al packet sniffer. Al contrario, il packet sniffer riceve una copia dei pacchetti che sono spediti/ricevuti dalle applicazioni e dai protocolli in esecuzione. Esso usa:

- Una libreria di cattura dei pacchetti, copiando ogni frame a livello di collegamento - Un analizzatore di pacchetti, che esamina il contenuto di ciascuno di questi

Si noti che in Windows viene anche installato Npcap, libreria e driver per cattura di pacchetti, quando si usa Wireshark.

Aiuti:

1) We first filter the packets by http (see the bar with "App a display filter").

As filter, type "http". By inspecting the "Hypertext transfer protocol", you might find the flag. 2) We reduced the traffic. In this small set of packets, we can see some request of authentications (Info "GET / HTTP/1.1."). One of these packets has been accepted (see its following packet).

Soluzione

Il suggerimento richiama l'uso di Wireshark.

Per prima cosa filtriamo i pacchetti per http (vedi la barra con "Apri un filtro di visualizzazione"). Come filtro, digitare "http".

Abbiamo ridotto il traffico. In questo piccolo set di pacchetti, possiamo vedere alcune richieste di autenticazioni (Info "GET / HTTP / 1.1."). Uno di questi contenuti è stato accettato (vedere il pacchetto seguente).

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

56 

Ispezionando il campo "Hypertext Transfer Protocol", possiamo notare il campo Autorizzazione. Qui contiene la nostra bandiera all'interno delle credenziali, in una stringa strana:

“Basic YmFzaE5pbmphOmZsYWd7aGVscC1tZS1vYml3YW59”

Decodificando la seconda stringa, visibilmente sospetta, da base64, si ottiene la flag:

*flag: "bashNinja:flag{help-me-obiwan}"*

2) Gforce: Testo, aiuti e soluzioni

Testo

*While losing some time on the Internet you fell on an old blog-post about conspiracy theories... A self-proclaimed hacker attached a network capture in a comment of this post telling that he will be `0xdeadbeef` before finishing the work.*

*Even if the job seems risky you cannot help it, you wanna look at it... the adventure begins...* Similmente a prima, viene dato un file “pcap” che in questo caso è “exfil.pcap”.

Aiuti:

1) Thanks to this tool we can easily see that, in this communication, only 2 protocols are involved (ARP and TCP), and 2 IPs as well (192.168.56.1 and 192.168.56.102).

To have a better overview of the communication, we can go to “analyze / follow / TCP

stream ”, and this screen will show all the messages.

2) Search for a base64 message.

Soluzione

In questo file ci viene chiesto di trovare informazioni sensibili all'interno di un flusso di pacchetti di traffico. Ci viene dato un file "pcap", che può essere aperto con Wireshark. Possiamo quindi aprirlo e lo schermo dovrebbe assomigliare al seguente:

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

**57

Grazie a questo strumento possiamo facilmente vedere che, in questa comunicazione, sono coinvolti solo due protocolli (ARP e TCP) e anche 2 IP (192.168.56.1 e 192.168.56.102).

Per avere una migliore panoramica della comunicazione, possiamo andare su "Analizza-Analyze /Segui - Follow/Flusso TCP/TCP Flow", e questa schermata mostrerà tutti i messaggi TCP (tecnica utile per analizzare genericamente ogni pacchetto).

Inoltre, utile usare i filtri display (per esempio, la stessa barra di ricerca che evidenzio nel disegno; in alcuni casi, è la soluzione migliore.

In questo caso, analizzando qualche messaggio, si nota qualcosa di sospetto all’interno del pacchetto 369, facendoci notare che i dati devono essere multipli di 4: questo ci suggerisce che, forse, dovremmo guardare qualcosa in base64.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

58 Andando poco più in là, nel pacchetto con id 377 si nota una chiara stringa in base64, visibile dal doppio uguale nel padding (immagine a lato). 

Nello specifico, abbiamo questa stringa base64:

*SU5TQXtjMTgwN2EwYjZkNzcxMzI3NGQ3YmYzYzY0Nzc1NjJhYzQ3NTcwZTQ1MmY3N2I3ZDIwMmI4MWUxN DkxNzJkNmE3fQ==*

Quando tradotta, ci darà la flag:

*INSA{c1807a0b6d7713274d7bf3c6477562ac47570e452f77b7d202b81e149172d6a7}*

3) Remote Media Controller: Testo, aiuti e soluzione

Testo

*Caasi Vosima organized a party last night to show is new high-tech house to his friends, yet something went wrong with the multimedia player and the music was turned off.*

*It took some time to restart the music player and the party was like frozen for a moment. Caasi was able to recover some information collected just before the crash.*

*Help Caasi to find out what happened !*

Ancora una volta, abbiamo un file *pcap* chiamato “remote-media-controler.pcap” da analizzare.

Soluzione

Abbiamo un file contenente il traffico prima del crash del player multimediale. Possiamo ordinare i pacchetti sulla base dela loro lunghezza (cliccando su Length/Lunghezza sono ordinati in maniera crescente) e trovare un pacchetto con lunghezza uguale a circa 4K (in fondo a tutti i pacchetti).

Ogni pacchetto è costituito da poche informazioni, principalmente *PLAY*, *QUIT*, *OK*.

Se guardiamo il contenuto (non ci sono tanti pacchetti e non ci si mette molto), troviamo il pacchetto 52 chiaramente in forma base64:

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Scorrendo tutto verso il basso nello stesso pacchetto appena indicato, inoltre, possiamo trovare una serie 59 

di stringhe di intermezzo e alcuni caratteri esadecimali con scritto infine “flag.txt”,

facendo ulteriormente intuire che potrebbe servirci.

Possiamo decodificarlo da base64... ma niente. Tuttavia, se si guarda bene, si

riottiene una stringa base64 (dopo la prima decodifica, si ha ancora una stringa con

un insieme di ==). Quindi, si riprova esattamente 5 volte e il risultato è il seguente:

*Good job! You found the flag: INSA{TCP\_s0ck3t\_4n4lys1s\_c4n\_b3\_fun!}*

Lezione 7: Introduzione alle vulnerabilità del web/Introduction to Web Vulnerabilities (Conti)

L'hacking è un tentativo di sfruttare un sistema informatico o una rete privata all'interno di un computer. In parole povere, è l'accesso o il controllo non autorizzato dei sistemi di sicurezza delle reti informatiche per scopi illeciti.

Per descrivere meglio l'hacking, occorre innanzitutto comprendere gli hacker. Si può facilmente supporre che siano intelligenti e altamente esperti di computer. In realtà, violare un sistema di sicurezza richiede più intelligenza e competenza che crearne uno. Non esistono regole ferree che ci permettano di classificare gli hacker in comparti ordinati.

Tuttavia, nel linguaggio informatico generale, distinguiamo tra:

- *white hat*, che violano i propri sistemi di sicurezza per renderli più a prova di hacker. Nella maggior parte dei casi, fanno parte della stessa organizzazione

- *black hat*, che violano il sistema per prenderne il controllo a fini personali. Possono distruggere, rubare o addirittura impedire agli utenti autorizzati di accedere al sistema. Lo fanno trovando scappatoie e punti deboli nel sistema. Alcuni esperti informatici li chiamano cracker anziché hacker.

- *grey hat*, persone curiose che hanno competenze informatiche appena sufficienti per poter entrare in un sistema e individuare potenziali falle nel sistema di sicurezza della rete. Essi differiscono dai black hat nel senso che i primi informano l'amministratore del sistema di rete delle debolezze scoperte nel sistema, mentre i secondi cercano solo guadagni personali. Tutti i tipi di hacking sono considerati illegali, tranne il lavoro svolto dai white hat hacker.

Fatte queste permesse, il *difensore/defender* è colui che ha sviluppato un sistema bersaglio, una persona normale magari con più esperienza ma spesso non un esperto di sicurezza. Ciò che spesso accade è che chi sviluppa un sistema non si concentra sulla sua sicurezza, ma dà la priorità a:

- funzionalità: il sistema fa ciò che deve fare

- prestazioni: il sistema è efficiente

Se la sicurezza è un aspetto secondario, iniziano i problemi. L’obiettivo è come visto fino ad ora con le challenge, quindi il metodo *capture the flag/CTF*. Come procedere?

- Raccogliere quante più informazioni possibili sul bersaglio (sistema operativo, linguaggio usato, protocolli, servizi, ecc.)

- Per ciascun componente possiamo trovare (grazie al Web) le vulnerabilità, sia di programmi, di linguaggi di programmazione oppure di strumenti utilizzati

Esempio di strumenti da poter sfruttare (exploit):

- Siti che non permettono di leggere tutti gli articoli, chiedendoci di aggiornare l’account ad un piano premium 🡪 Cerchiamo di ottenere accesso infinito ai suoi contenuti

- Giochi che ci piacciono 🡪 Cerchiamo di ottenere accessi a contenuti non previsti

normalmente/zone non accessibili/craccarlo e poter avere accesso a contenuti non normalmente disponibili

- Siti internet 🡪 Tramite lo strumento “Ispeziona” presente nei browser

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

60 *Come usare Docker su Windows*

Indicazioni di massima dal link:

https://linuxhint.com/run-sh-file-windows/

• Attivare la modalità Sviluppatori di Windows

• Installare tra le funzionalità aggiuntive Windows Subsystem for Linux e spuntare su ON • Installare Ubuntu da Windows Store (es. versione 22.02)

• Installare Docker Desktop

• Attivare la CLI di Linux con il comando "bash"

(Non serve attivare sudo dockerd in quanto Docker Desktop attiva già il daemon di per sé) • *sudo chmod -R +rx ./*

• *sudo docker\_run.sh*

• *sudo docker\_build.sh*

*Come usare Docker su Ubuntu*

• *sudo apt update*

• *sudo apt install apt-transport-https ca-certificates curl software-properties-common* • *curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –*

• *sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"*

• *sudo apt install docker-ce*

*Come usare Docker Compose su Windows/Ubuntu*

(Avendo attivato Linux Subsystem e dopo aver installato Ubuntu)

Indicazioni di massima dal link:

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu 20-04

• *sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose- $(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose*

• *sudo chmod +x /usr/local/bin/docker-compose*

• *docker-compose --version*

• (Spostandosi nella cartella di interesse)

• *docker-compose up*

Esercizi Lezione 7

1) Ajax Not Soap: Testo e soluzione

Da questo esercizio, cominciano a comparire dei file Docker. Per farla molto corta, usano dei file chiamati container, i quali permettono di definire e scaricare un’applicazione pronta all’uso con un semplice script di configurazione (Dockerfile) appositamente scritto.

Testo

*# Ajax Not Soap*

*## Description*

*Javascript is checking the login password off of an ajax call, The verification is being done on the client side.*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

61 *making a direct call to the ajax page will return the expected password RULES = you don't have access to the 'web' folder.*

*Be sure that the entire folder has the right permissions.*

*To do it, open the terminal and write*

*chmod -R +rx ./*

*REMEMBER: do this operation for every exercise.*

*To execute the exercise, do the following on the terminal*

*sudo ./docker\_build.sh*

*and then*

*sudo ./docker\_run.sh*

*Check inside docker\_run the ip:port to use (in this case 127.0.0.1:8085)*

Quindi, ciò che occorre fare è: (lo ripeterò nei vari esercizi, mettendo di volta in volta l’indirizzo IP a cui collegarsi, normalmente visibile, quando non indicato dall’esercizio, direttamente dal daemon Docker da terminale [tramite il comando *sudo docker ps*])

- *chmod -R +rx ./* (imposta il permesso di lettura ed esecuzione, includendo tutte le sottodirectory con *-R*, che significa *Recursive* [dentro tutte le sottocartelle] partendo dal percorso attuale, qualunque esso sia, con *./*)

- In un’altra finestra di terminale (diversa da quella dei successivi comandi) 🡪 *sudo dockerd* In questo modo, si attiva il daemon di Docker. Deve rimanere aperta questa finestra.

(Basta farlo una volta sola, resta attivo dopo)

- *sudo ./docker\_build.sh*

- *sudo ./docker\_run.sh* (lasciare aperta questa finestra)

- Connettersi in un browser all’indirizzo *http://127.0.0.1:8085 (attenzione che è http e non https; così per tutti quegli indirizzi localhost come questo)*

o Si può sempre inserire *localhost:port* al posto di *http:/127.0.0.1:port*

o Se si inserisse come HTTPS, si evidenzi tutto l’indirizzo dalla barra apposita e lo si reincolli nella stessa barra; quando questo accade, è tutto evidenziato in azzurro e, in questo modo, si riesce a scrivere HTTP e non HTTPS

- Per vedere i container attivi e scoprirne l’indirizzo IP 🡪 *sudo docker ps*

Caso errore comune: *Docker - Name is already in use by container “id”*

- Usare *sudo docker system prune* per uccidere tutti i container

Nel caso (utile per evitare di attivare ogni volta il daemon di Docker), meglio attivare Docker all’avvio del sistema, così il daemon non muore ogni volta:

- *sudo systemctl enable docker.service*

- *sudo systemctl enable containerd.service*

Soluzione

La pagina web si presenta come segue:



*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

62 Il nostro obiettivo è probabilmente quello di trovare un nome utente e una password adeguati che ci permettano di accedere al sistema e di stampare la flag. Se si gioca un po' inserendo stringhe a caso all'interno delle caselle di testo si vede un messaggio che dice "nome utente non corretto" o "password non corretta". Possiamo analizzare il codice Javascript, premendo tasto destro su “Analizza/Inspect Source” e poi nella scheda “Inspector/Analisi pagina” espandendo il pezzo HTML con JavaScript.



Ci sono due funzioni principali, una che controlla il nome utente e una che controlla la password. password. Le coppie corrette di nome utente-password vengono recuperate tramite una funzione Ajax "webhooks" (cioè letteralmente, un URL che accetta un POST/GET/PUT).

Dato che si tratta di un controllo 

lato client, possiamo usare il

debugger del browser (scheda

Debugger di Inspect Source)

e impostare due punti di

interruzione/breakpoints sulle linee

che puliscono la variabile *data* (ad

esempio, *data=data.replace([...])*).

In questo modo, possiamo digitare

cose a caso sul nome utente e il

breakpoint ci mostra

il valore reale di *username*:

*Username = MrClean*

Ora conosciamo il nome utente e quindi possiamo inserire il valore corretto nel campo username del form visto sopra. Possiamo fare lo stesso per ottenere la password (digitando caratteri casuali e inserendo i breakpoint in corrispondenza del codice).

Questa volta il contenuto della password è la flag stessa 🡪 *Flag = flag{hj38dsjk324nkeasd9}*

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

**63 2) Console: Testo e soluzione

Testo

*You control the browser*

*http://127.0.0.1:8081*

*(use ./docker\_run.sh to run the server locally)*

*RULES: you don't have access to web*

Quindi, per avviare l’esercizio (spostandosi (cd) in preventivo nella cartella che contiene tutto *console*): - *chmod -R +rx ./*

- In un’altra finestra di terminale (diversa da quella dei successivi comandi) 🡪 *sudo dockerd* In questo modo, si attiva il daemon di Docker. Deve rimanere aperta questa finestra.

(Basta farlo una volta sola, resta attivo dopo)

- *sudo ./docker\_build.sh*

- *sudo ./docker\_run.sh* (lasciare aperta questa finestra)

- Aprire un browser all’indirizzo http://127.0.0.1:8081

Soluzione

La pagina web è come segue:



Non si può fare molto, quindi la prima cosa da fare è provare qualcosa. Sono disponibili due elementi: ● Una casella in cui è possibile inserire del testo

● Un pulsante

Possiamo fare una prova, come la seguente:



*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

64 Possiamo notare il codice Javascript. Dal flow, possiamo vedere che per prima 

cosa il valore che inseriamo viene

controllato con la funzione md5 e, se c'è

una corrispondenza, viene chiamata la

funzione *getThat("Y"),*

altrimenti *getThat("N").*

In particolare, si tralasci la funzione che

crea una hash ripetuta alcune volte di

MD5; è decisamente inutile, se non per

confondere/offuscare l’obiettivo.

Quando l'argomento è "Y", la funzione

getThat prende qualcosa da una pagina

esterna e ne stampa il suo contenuto;

altrimenti, viene visualizzato il

messaggio predefinito "Nope".

L'indizio principale è che per ottenere la flag, dobbiamo 

trovarci in una situazione del tipo

*getThat("Y")*. Tuttavia, questa è solo una funzione js

(funzione JavaScript), quindi nel nostro browser possiamo

andare nella sezione Console (seconda tab dopo tasto dx e

Ispeziona/Esamina) e scrivere proprio come si vede qui:

*In altro modo*, si può osservare che la stringa in MD5 non viene controllata lato server. La flag può essere ottenuta sfruttando il controllo da parte di *‘/1/key.php’*, che controlla se la richiesta provenga da XHR (XMLHttpRequest, utilizzati per interagire con i server. È possibile recuperare dati da un URL senza dover aggiornare l'intera pagina). Quindi, si può usare quanto segue, ottenendo la flag:

*$.ajax({*

*type: 'GET',*

*url: '1/key.php',*

*success: function (file\_html) {*

*foo.innerHTML=(file\_html)*

*}*

*});*

3) Das Blog: Testo e soluzione

Viene fornito un file README.md con il seguente testo:

*RULES = you cannot access to 'web' and 'other' folders.*

Quindi, per avviare l’esercizio (spostandosi [*cd*] in preventivo nella cartella che contiene tutto *Das Blog*): - *chmod -R +rx ./*

- In un’altra finestra di terminale (diversa da quella dei successivi comandi) 🡪 *sudo dockerd* In questo modo, si attiva il daemon di Docker. Deve rimanere aperta questa finestra.

(Basta farlo una volta sola, resta attivo dopo)

- *sudo ./docker\_build.sh*

- *sudo ./docker\_run.sh* (lasciare aperta questa finestra)

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

65 - Aprire un browser all’indirizzo http://127.0.0.1:8084 Soluzione

La pagina web è come segue:



Se inseriamo per esempio "test" nel nome utente e nella password, riceviamo il seguente messaggio: *Sorry, That Username / Password is incorrect.*

Ispezioniamo la pagina:



All'inizio dello screen precedente, si noti il commento. Proviamo ad inserire quelle credenziali come segue: *Username: JohnsTestUser*

*Password: AT3stAccountForT3sting*

Una volta impostato l’utente, si torni 

alla homepage e si nota che siamo

entrati con i permessi di DEFAULT.

Sembra che con questo account non abbiamo i permessi per raggiungere le "informazioni sensibili". Dobbiamo scoprire come vengono gestiti i permessi e possiamo provare con i cookie. I cookie sono nella sezione "Applicazione" del nostro strumento di debug.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

**66

Quello che si deve fare esattamente è:

- dopo aver immesso le credenziali di accesso precedente, si clicchi indietro una/due volte e si ritorni alla Home

- ora, viene visualizzato “You have DEFAULT permissions”

- in questa schermata, tasto destro e si clicca su “Inspect”, scheda “Cookies” e poi si imposta come “admin” nel campo “permissions”, nell’insieme che si vede sotto dei cookies nella pagina - si ricarica la stessa pagina



Fatte le operazioni come descritto, ecco che si ottiene la flag.



Lezione 8: Ingredienti del Web/Ingredients of Web (Conti)

Dietro alla maggior parte delle applicazioni che utilizziamo, ci sta l'architettura client-server, alternativamente chiamata modello client-server, è un'applicazione di rete che suddivide le attività e i carichi di lavoro tra client e server che risiedono sullo stesso sistema o sono collegati da una rete di computer.

L'architettura client-server è tipicamente caratterizzata da postazioni di lavoro, PC o altri dispositivi di più utenti, collegati a un server centrale tramite una connessione Internet o un'altra rete. Il client invia una richiesta di dati e il server accetta e soddisfa la richiesta, inviando i pacchetti di dati all'utente che ne ha bisogno. Questo modello è chiamato anche rete client-server o modello di network computing.

Un client è una persona o un'organizzazione che utilizza un servizio. Nel contesto informatico, il client è un computer/dispositivo, detto anche host, che utilizza effettivamente il servizio o accetta le informazioni. I dispositivi client includono laptop, workstation, dispositivi IoT e altri dispositivi simili compatibili con la rete. Nel mondo IT, un server è un computer remoto che fornisce accesso a dati e servizi. I server sono solitamente dispositivi fisici come i server rack, anche se l'ascesa del cloud computing ha portato i server

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

67 virtuali nell'equazione. Il server gestisce processi come la posta elettronica, l'hosting di applicazioni, le connessioni a Internet, la stampa e altro ancora.

Per riassumere brevemente: 

- In primo luogo, il cliente invia la propria richiesta

tramite un dispositivo abilitato alla rete.

- Il server di rete accetta ed elabora la richiesta

dell'utente.

- Infine, il server invia la risposta al client.

L'architettura client-server presenta tipicamente le seguenti caratteristiche:

- Le macchine client e server richiedono in genere risorse hardware e software diverse e provengono da altri fornitori.

- La rete ha una scalabilità orizzontale, che aumenta il numero di macchine client, e una scalabilità verticale, che sposta l'intero processo su server più potenti o su una configurazione multi-server. - Un computer server può fornire più servizi contemporaneamente, anche se ogni servizio richiede un programma server separato.

- Sia le applicazioni client che quelle server interagiscono direttamente con un protocollo di livello di trasporto. Questo processo stabilisce la comunicazione e consente alle entità di inviare e ricevere informazioni.

- Sia il computer client che quello server hanno bisogno di uno stack completo di protocolli. Il protocollo di trasporto impiega protocolli di livello inferiore per inviare e ricevere singoli messaggi.

Ciascun dispositivo, in una rete, è univocamente identificato da un indirizzo IP, essenziale per l’identificazione. Un dispositivo potrebbe avere multiple comunicazioni (tramite le porte) e i dispositivi comunicano tramite i protocolli.

Alcuni esempi di architettura client-server (in particolare, ci serve il terzo) sono:

- Server di posta elettronica: Grazie alla facilità e alla velocità, la posta elettronica ha soppiantato la posta tradizionale come forma principale di comunicazione aziendale. I server di posta elettronica, coadiuvati da varie marche di software dedicati, inviano e ricevono e-mail tra le parti.

- Server di file: Se si archiviano file su servizi basati su cloud come Google Docs o Microsoft Office, si utilizza un file server. I file server sono luoghi centralizzati per l'archiviazione dei file e sono accessibili da molti client.

- Server web: Questi server ad alte prestazioni ospitano molti siti web diversi, ai quali i clienti accedono tramite Internet. Ecco una spiegazione passo per passo:

o Il cliente/utente utilizza il proprio browser Web per inserire l'URL desiderato.

o Il browser chiede al sistema dei nomi di dominio (DNS) un indirizzo IP.

o Il server DNS trova l'indirizzo IP del server desiderato e lo invia al browser web.

o Il browser crea una richiesta HTTPS o http

o Il server/produttore invia all'utente i file corretti

o Il client/utente riceve i file inviati dal server e il processo si ripete se necessario.

Le applicazioni Web sono accessibili dai browser (Chrome, Firefox, Edge, Vivaldi, ecc.). Il layout (come sono disposti gli oggetti graficamente) è gestito da HTML, che non è un linguaggio di programmazione, ma un linguaggio di *markup* (infatti, lo stesso nome completo è HyperText Markup Language). L'HTML non è un linguaggio di programmazione per tre motivi:

1) non consente l'uso di variabili

2) non consente l'uso di dichiarazioni condizionali.

3) non fornisce strutture di looping iterativo.

Esso ha delle struttura che, tramite il CSS ad esempio (Cascade Style Sheets), permettono di arricchire le possibilità offerte da HTML per garantire effetti grafici altrimenti non ottenibili.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

68 Qui facciamo una distinzione tra:

- linguaggi lato client, in cui si ha un programma che viene eseguito sul computer client (browser) e si occupa dell'interfaccia/visualizzazione dell'utente e di qualsiasi altra elaborazione che può avvenire sul computer client, come la lettura/scrittura di cookie. Le operazioni sono come segue: 1) Interagire con la memoria temporanea

2) Crea pagine web interattive

3) Interagire con la memoria locale

4) Inviare richieste di dati al server

5) Inviare richieste al server

6) lavorare come interfaccia tra il server e l'utente

Di questi fanno parte Ajax, CSS, JavaScript

- linguaggi lato server, in cui si ha un programma che viene eseguito sul server e che si occupa della generazione del contenuto della pagina web. Le operazioni sono come segue:

1) Interrogazione del database

2) Operazioni sui database

3) Accedere/scrivere un file sul server.

4) Interagire con altri server.

5) Strutturare applicazioni web.

6) Elaborare l'input dell'utente. Ad esempio, se l'utente inserisce un testo nella casella di ricerca, esegue un algoritmo di ricerca sui dati memorizzati sul server e invia i risultati.

Di questi fanno parte PHP, Python, Ruby

Nelle trasmissione via web, il protocollo “base” è il ben noto *HTTP*: protocollo di livello applicativo per trasmettere documenti ipermediali, come l'HTML.

Col tempo, è stata sviluppata una versione sicura, nota come *HTTPS*, ottenuta rendendo http sicuro utilizzando una connessione *TLS* (Transport Layer Security, quindi un protocollo che garantisce comunicazione sicura) tra due host.

TLS garantisce la riservatezza, l'integrità dei dati, l'autenticazione del server e la resistenza a diversi attacchi specifici mentre i dati vengono trasmessi in rete.

Esistono in rete delle informazioni che tutti quotidianamente accettiamo chiamate *cookie*, che sono piccoli blocchi di dati creati da un server web mentre un utente naviga su un sito web e collocati sul computer o su un altro dispositivo dell'utente dal browser web di quest'ultimo. I cookie vengono memorizzati sul dispositivo utilizzato per accedere a un sito web e più di un cookie può essere memorizzato sul dispositivo dell'utente durante una sessione.

I cookie svolgono funzioni utili e talvolta essenziali sul web. Consentono ai server Web di memorizzare informazioni di stato (come gli articoli aggiunti al carrello in un negozio online) sul dispositivo dell'utente o di tracciare l'attività di navigazione dell'utente (tra cui il clic su determinati pulsanti, il login o la registrazione delle pagine visitate in passato). Possono anche essere utilizzati per salvare per un uso successivo le informazioni che l'utente ha precedentemente inserito nei campi dei moduli, come nomi, indirizzi, password e numeri di carte di pagamento.

Questo è utile, dato che HTTP è *stateless* di per sé, quindi non memorizza informazioni.

Tuttavia, si possono ovviamente avere dei problemi.

Il *cookie hijacking/dirottamento dei cookie* , chiamato anche *session hijacking*, è un modo per gli hacker di accedere e rubare i vostri dati personali e può anche impedirvi di accedere a determinati account. Il dirottamento dei cookie è altrettanto potente, a volte di più, della scoperta della vostra password. Possono essere visibili tramite apposita tab del browser tramite “Ispeziona” e cliccando la tab “Applicazione/Application”, infine cliccando la scheda “Cookies”.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

**69

I cookie vengono creati per identificare l'utente quando visita un nuovo sito web. Il server Web, che memorizza i dati del sito, invia un breve flusso di informazioni 

identificative al browser Web dell'utente.

I cookie del browser sono identificati e letti da

coppie "nome-valore". Queste indicano ai cookie

dove devono essere inviati e quali dati richiamare.

Il server invia il cookie solo quando vuole che il

browser web lo salvi. Se vi state chiedendo "dove

vengono memorizzati i cookie", è semplice: il

browser Web li memorizza localmente per ricordare

la coppia "nome-valore" che vi identifica.

Se l'utente torna a visitare il sito in futuro, il browser

web restituisce i dati al server web sotto forma di

cookie. A questo punto il browser lo rimanda al server per richiamare i dati delle sessioni precedenti.

I client richiedono al server alcune cose con alcuni metodi HTTP come ad esempio *GET, POST, PUT, HEAD, DELETE, PATCH, OPTION*S, ecc.

*GET* è utilizzato per richiedere dati da una risorsa specifica.

- Ad esempio, */test/demo.php?nome1=valore1&nome2=valore2*

- La stringa di query viene inviata nell'URL di una richiesta GET.

*POST* è utilizzato per inviare dati a un server per creare/aggiornare una risorsa

- I dati inviati sono memorizzati nel corpo della richiesta HTTP della richiesta

Le applicazioni di solito si aspettano alcuni input

- Ad esempio, una calcolatrice si aspetta dei numeri.

Dobbiamo controllare l'input che la nostra applicazione riceve.

Questo processo è chiamato validazione e *sanitizzazione* degli input (*input validation* and *sanitization*) L'applicazione elabora solo gli input fattibili, rifiutando quelli non fattibili. Dove mettere queste cose? - Lato client: può essere facilmente aggirato (ad esempio, se basato su JS).

- Lato server: aumentano l'overhead del server.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Esercizi Lezione 8 70

1) Ajax Not Borax: Testo, aiuti e soluzione

Testo

*http://127.0.0.1:8083/*

*(use ./docker\_run.sh to run the server locally)*

*Rules: you cannot access the 'web' folder, but you can use online tools.*

Quindi, per avviare l’esercizio (spostandosi (cd) in preventivo nella cartella che contiene tutto *Das Blog*): - *chmod -R +rx ./*

- In un’altra finestra di terminale (diversa da quella dei successivi comandi) 🡪 *sudo dockerd* In questo modo, si attiva il daemon di Docker. Deve rimanere aperta questa finestra.

(Basta farlo una volta sola, resta attivo dopo)

- *sudo ./docker\_build.sh*

- *sudo ./docker\_run.sh* (lasciare aperta questa finestra)

- Aprire un browser all’indirizzo http://127.0.0.1:8083/

Aiuti:

1) If you check the JS, you see two main functions, one that checks the username, and one that checks the password. The correct pairs of username-passwords are retrieved using a “webhooks” ajax function. These values are then compared with the MD5 function.

This is client-side control: by placing a breakpoint in the right place, we can retrieve the correct ***username***. Since this is an MD5 encoded username, you need to crack it.

Use some online tools.

2) If you are here, you successfully retrieved the *username*: “tideade”.

Now we aim to get the password. Let us focus on the second if-block. Now the comparison is between 2 MD5 values, i.e., the retrieved real password is in clear!

With the debugger, we obtain an encoded version of the solution.

Soluzione

La pagina web si presenta come segue:



Il nostro obiettivo è probabilmente quello di trovare un nome utente e una password adeguati che ci permettano di accedere al sistema e di stampare (?) una flag. Se si gioca un po' inserendo stringhe a caso all'interno delle caselle di testo si vede un messaggio che dice "nome utente non corretto" o "password non corretta".

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Possiamo analizzare il codice Javascript:

71



Ci sono due funzioni principali, una che controlla il nome utente e una che controlla la password. Le coppie corrette di nome utente-password vengono recuperate tramite una funzione ajax "*webhooks*". Questi valori vengono poi confrontati con la *funzione MD5*.

Nel browser, possiamo impostare un breakpoint nella riga che assegna "*data*" nel primo blocco if; l'MD5 del nome utente (reale) è *"c5644ca91d1307779ed493c4dedfdcb7"*

**

Possiamo decifrarlo? Notando il codice descritto, si ha una funzione JavaScript che fa intuire che si tratta di una hash MD5. Possiamo usare ad esempio il sito seguente: https://crackstation.net/



Il nome utente è "tideade" e se lo inseriamo nella nostra interfaccia vediamo che è quello giusto.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

72 Concentriamoci sul secondo blocco if. Ora il confronto è tra 2 valori MD5, cioè la password reale recuperata è in chiaro in XHR. Con il debugger otteniamo il seguente valore:

*"ZmxhZ3tzZDkwSjBkbkxLSjFsczlISmVkfQ==".*

**È una stringa in base64, ma non ci interessa. Se lo inseriamo nel campo, appare un messaggio: la password! Possiamo decodificarla e convertirla in una base normale, rivelando la flag:

*flag{sd90J0dnLKJ1ls9HJed*

2) Sweeeeeet: Testo, aiuti e soluzione

Testo

When you see a *docker-compose* file, use the following command to run the exercise:

*sudo docker-compose up*

If your machine does not provide *docker-compose*, you can install it by following this guide: *https://docs.docker.com/compose/install/*

To solve the exercise, you need first to inspect the app with the browser’s debugger, and then, once you understood what you need to solve it, we suggest you write a Python script.

Some useful Python libraries:

import hashlib

import codecs

import numpy as np

import requests

A request example:

#IP

ip = "127.0.0.1"

port= "8080"

#we first check that our MD5 works by comparing Md5(100) with #the one in the webpage

control = "f899139df5e1059396431415e770c6dd"

tester = 100

tester\_b = str.encode(str(tester))

tester\_md5 = hashlib.md5(tester\_b).hexdigest()

print(f"tester={tester\_md5 == control}")

The test returns *true;* that’s actually useful for the exercise logic and solution.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

Aiuti:

73

1) There are two cookies:

1. FLAG: which contains an incomplete flag;

2. UID: user ID

What we can think is that by giving the correct user-ID, the page will return the flag.

The value contained in UID “f899139df5e1059396431415e770c6dd” seems a hashed value. By using a password cracker, we can see that the value is an MD5 hash, where the ciphertext is 100. This means that the UID= MD5(100) is a wrong UID (since we don’t have a correct flag). UID seems the MD5 of an integer: what if we bruteforce them? We can start from 1 to 100. 2) A brute-force among several integer numbers seems the right approach.

Quindi, per avviare l’esercizio (spostandosi (cd) in preventivo nella cartella che contiene tutto *Sweeeeeet*): - *chmod -R +rx ./*

- *sudo dockerd* (In una finestra di terminale a parte da quella dei comandi successivi; basta farlo una volta e occorre lasciare la finestra aperta)

- *sudo docker-compose up* (lasciare aperta questa finestra)

- Aprire un browser all’indirizzo http://127.0.0.1:8080/

Soluzione

Abbiamo la seguente pagina Web:



Non c’è molto da guardare qui. Ispezionando i cookie si può notare qualcosa di interessante:

Ci sono due cookies:

1. FLAG: che contiene un flag incompleto;

2. UID: ID utente

Possiamo pensare che, fornendo l’ID utente corretto, la pagina restituirà la flag.

Il valore contenuto in UID “*f899139df5e1059396431415e770c6dd*” sembra un valore hash. Utilizzando un cracker di password (crackstation.net) possiamo vedere che il valore è un hash MD5, dove, codificando con MD5 con Crackstation, si ricava 100.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

74 Letteralmente, modifico nei Cookie lo UID inserendo 100 e la flag viene visualizzata. 

La flag è formattata come URL; ecco che, modificando il tutto opportunamente, diventa: *encryptCTF{4lwa4y5\_Ch3ck\_7h3\_c00ki3s}*

Alternativamente [cfr: empirectf/README.md at master · EmpireCTF/empirectf (github.com)] Visitando il sito la prima volta, viene dato un cookie UID:

*$ curl -vv http://104.154.106.182:8080/ 2>&1 | grep "Set-Cookie"*

*< Set-Cookie: UID=f899139df5e1059396431415e770c6dd; expires=Sat, 06-Apr-2019 14:20:00 GMT; Max Age=172800*

Visitando il sito la seconda volta con il cookie UID impostato, otteniamo un altro cookie: $ *curl -b "UID=f899139df5e1059396431415e770c6dd" -vv http://104.154.106.182:8080/ 2>&1 | grep "Set Cookie"*

*< Set-Cookie: FLAG=encryptCTF%7By0u\_c4nt\_U53\_m3%7D*

Ma questa non è la bandiera.

È interessante notare che l'UID è sempre lo stesso, anche quando si visita con browser diversi o da IP diversi. Infatti, l'hash è un hash noto ed è *md5("100") == "f899139df5e1059396431415e770c6dd".* Possiamo cambiare il nostro UID in *md5("0") == "cfcd208495d565ef66e7dff9f98764da",* che ci fornisce il flag effettivo:

*$ curl -b "UID=cfcd208495d565ef66e7dff9f98764da" -vv http://104.154.106.182:8080/ 2>&1 | grep "Set Cookie"*

*< Set-Cookie: FLAG=encryptCTF%7B4lwa4y5\_Ch3ck\_7h3\_c00ki3s%7D%0A*

Da questo, si capiva il senso del codice di confronto che lo SPRITZ ha dato sopra.

Inoltre, se si provassero ad inviare in bruteforce MD5 da 0 a 101, l’unica risposta diversa è proprio quella data da *md5(0)*.

Il tutto è risolvibile con una sola linea di codice:

curl http://104.154.106.182:8080/ -H "Cookie: UID=$(printf %s '0' | md5sum | cut -c 1-32)" --head -s | grep -oP 'FLAG=\K([a-zA-Z0-9\{\}%\_]+)' | perl -pe 's/\%(\w\w)/chr hex $1/ge'

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

75 3) Vault: Testo, aiuti e soluzione Testo

*You do not have access to any file.*

*The challenge can be resolved only browser side.*

*To launch the app, run the following*

*- docker-compose up*

Aiuto:

1) If we try to insert some random values, e.g., “test”, the application responds with a “Denied Access” page.

We need to guess correct credentials to reach the flag. A fair hypothesis is

that the APP is using a database like system that handles the password. If we inspect the page we do not see any useful information that we can exploit.

As aforementioned, a fair hypothesis is that the system relies on a database, so what about SQL injection?

Search on the web how simple SQL injection works.

Quindi, per avviare l’esercizio (spostandosi (cd) in preventivo nella cartella che contiene tutto *Vault*): - *chmod -R +rx ./*

- *sudo dockerd* (In una finestra di terminale a parte da quella dei comandi successivi; basta farlo una volta e occorre lasciare la finestra aperta)

- *sudo docker-compose up*

- Aprire un browser all’indirizzo http://127.0.0.1:9090/

Soluzione

Ci viene fornita la seguente pagina web:



Se proviamo a inserire alcuni valori casuali, ad esempio “test”, l’applicazione risponde con una pagina “Accesso negato”. Sembra che sia necessario indovinare le credenziali corrette per raggiungere la flag. Un’ipotesi corretta è che l’applicazione utilizzi un sistema simile a un database per la gestione delle password. Se ispezioniamo la pagina non vediamo alcuna informazione utile che possiamo sfruttare.

Proviamo ad usare una SQL Injection.

(Una SQL Injection si verifica di solito quando si chiede a un utente un input, come il suo nome utente/id, e invece di un nome/id, l’utente fornisce un’istruzione SQL che verrà inconsapevolmente eseguita sul database.

- SQL Injection basata sull’inserimento di 1=1 è sempre vera

- SQL Injection basata sull’inserimento “”=”” è sempre vera

- SQL Injection basata sull’inserimento di statement SQL (es. stringa ed altra query SQL) Un modo per proteggersi è parametrizzare le query, tali da essere preparati a sanitizzare ogni stringa prevenendo caratteri speciali oppure inserimento di keyword SQL che validano sempre l’accesso Di più al link: https://www.w3schools.com/sql/sql\_injection.asp)

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

76 Possiamo inserire l’attacco SQL Injection inserendo *‘ or 1=1--* come username e password

Attenzione: Ad alcuni sembra andare con 🡪 *' or '='*

**

Con il seguente risultato:



Non c’è molto da vedere: se seguiamo il codice QR non troviamo nessuna flag ispezionando il codice HTML. Tuttavia, andando dentro ai cookies, l’unico di sessione presente è la seguente stringa: *ZW5jcnlwdENURntpX0g0dDNfaW5KM2M3aTBuNX0%3D*

Si tratta di una stringa base64, ma dobbiamo togliere il *%3D*, in quanto attualmente si tratta di un URL. Possiamo quindi provare a decodificarla, ottenendo correttamente la flag:

*encryptCTF{i\_H4t3\_inJ3c7i0n5*

*Altra possibile soluzione* 🡪 comporta l’uso di *sqlmap*, strumento open source che esegue in automatico le iniezioni. Qui si può usare il comando:

*sqlmap -u 127.0.0.1:9090/login.php --data="username=admin&password=pass&submit=submit" -p username --dump --time-sec 1 --batch --answer="crack=n*

Lezione 9: Language Vulnerabilities/Vulnerabilità del linguaggio (Conti)

Oltre alle possibili vulnerabilità del sistema in uso, bisogna decisamente considerare anche le possibili vulnerabilità del linguaggio che si sta usando. Alcune funzioni possono esporre l’applicazione a minacce. È una buona pratica essere consapevoli di questi rischi per prevenire gli attacchi.

Per esempio, il C è una sorta di padre di tutti i linguaggi di programmazione ed è considerato di alto livello. Un linguaggio di alto livello è un linguaggio di programmazione che consente di sviluppare un programma in un contesto di programmazione molto più semplice e generalmente indipendente dall’architettura hardware del computer, in cui molte cose sono lasciate al programmatore (ad esempio, la gestione della memoria (allocazione/deallocazione di variabili)).

Si possono trovare diverse minacce (link di riferimento: https://int0x33.medium.com/day-49-common-c code-vulnerabilities-and-mitigations-7eded437ca4a)

Molte vulnerabilità del C riguardano i buffer overflow.

Buffer overflow (o buffer overrun), in informatica, è una condizione di errore che si verifica a runtime quando in un buffer di una data dimensione vengono scritti dati di dimensioni maggiori. Le aree di memoria riservate ai buffer per contenere i dati, quando viene scritto codice vulnerabile, consentono a un exploit di scrivere su altri valori importanti in memoria, come le istruzioni che la CPU deve eseguire successivamente. C e C++ sono suscettibili di buffer overflow perché definiscono le stringhe come

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

77 array di caratteri a terminazione nulla, non controllano implicitamente i limiti e forniscono chiamate di libreria standard per le stringhe che non applicano il controllo dei limiti.

La gestione della memoria, nel caso di C, è completamente data al programmatore; ciò non accade nel caso di linguaggi come Java, dove la *garbage collection* (gestione della memoria e deallocazione delle variabili) è automaticamente gestita.

A tal proposito, ad esempio, abbiamo:

- La funzione *gets()* non può essere utilizzata in modo sicuro. A causa della mancanza di controllo dei limiti e dell’impossibilità per il programma chiamante di determinare in modo affidabile la lunghezza della prossima riga in arrivo, l’uso di questa funzione consente agli utenti malintenzionati di modificare arbitrariamente la funzionalità di un programma in esecuzione attraverso un attacco di buffer overflow. Essa legge infiniti caratteri dati da un flusso e li memorizza nella stringa *str* Nell’esempio, cosa succede se inseriamo più di 15 caratteri? 🡪 corruzione della memoria 

- La funzione *strcpy()* copia i caratteri contenuti in *src* in *trg* e può essere facilmente utilizzata in modo improprio, consentendo agli utenti malintenzionati di modificare arbitrariamente la funzionalità di un programma in esecuzione attraverso un attacco di buffer overflow.

In questo esempio, cosa succede se copiamo più di 10 caratteri? 🡪 corruzione della memoria 

Ora prendiamo l’esempio di altri linguaggi, come ad esempio PHP, usato spesso nel contesto web, definito come *dynamically typed language*, il che significa che il tipo è associato ai valori di run-time e non alle variabili nominate, ai campi e così via. Questo significa che il programmatore può scrivere un po’ più velocemente, perché non deve specificare i tipi ogni volta (a meno che non usi un linguaggio a tipizzazione statica con inferenza dei tipi). Questo a volte può essere un problema.

(Link di riferimento:

https://medium.com/swlh/php-type-juggling-vulnerabilities-3e28c4ed5c09

https://www.php.net/manual/en/language.types.type-juggling.php)

PHP ha una funzione chiamata “type juggling” o “type coercion”. Ciò significa che durante il confronto di variabili di tipo diverso, PHP le converte prima in un tipo comune e comparabile.

Ad esempio, quando il programma confronta la stringa “7” e il numero intero 7 nello scenario seguente: 

Il codice verrà eseguito senza errori e mostrerà “PHP può confrontare interi e stringhe”. Questo comportamento è molto utile quando si vuole che il programma sia flessibile nel gestire diversi tipi di input dell’utente.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

78 Tuttavia, è importante notare che questo comportamento è anche una fonte importante di bug e vulnerabilità di sicurezza. Ad esempio, quando PHP deve confrontare la stringa “7 puppies” con il numero intero 7, PHP cercherà di estrarre il numero intero dalla stringa. Quindi il confronto sarà valutato come Vero.



Il modo più comune in cui questa particolarità di PHP viene sfruttata è l’utilizzo per aggirare l’autenticazione. Quindi, inviando semplicemente un input intero pari a 0, si accederà con successo come amministratore, poiché la valutazione sarà True:



Come farlo in pratica:

1. Identificare il linguaggio di programmazione utilizzato nell’applicazione

2. Identificare la versione

3. Identificare le eventuali librerie utilizzate

4. Verificare su Google la presenza di eventuali vulnerabilità

Esercizi Lezione 9

1. You are given the code of a webapp. What is it hiding?

2. Because creating real pwn challs was to mainstream, we decided to focus on the development of our equation solver using OCR

3. The authors tried to protect their JS code … is that enough to scare an attacker?

1) 50\_Slash\_Slash: Testo, aiuti e soluzione

Testo

*You own the application.*

*Free to use any resource you are given (e.g., you can have a look at the files contained in the 7z file).*

In questo contesto, dettaglierò tutti i passi a differenza di chi dovrebbe farlo, ma non lo fa: - Unzippare tutta la cartella in formato 7z

- Dentro, seguire il link: https://www.youtube.com/watch?v=N5vscPTWKOk&list=PL osiE80TeTt66h8cVpmbayBKlMTuS55y&index=7 per imparare ad usare i *virtual environment* di Python. Un ambiente virtuale in Python è uno strumento che aiuta a mantenere separate le dipendenze richieste da diversi progetti creando per loro ambienti virtuali isolati.

- Le applicazioni sviluppate con Python useranno spesso pacchetti e moduli che non fanno parte della libreria standard. A volte le applicazioni necessitano di una versione specifica di una libreria, poiché l’applicazione potrebbe richiedere la correzione di un determinato bug o l’applicazione potrebbe essere scritta utilizzando una versione obsoleta dell’interfaccia della libreria.

Ciò significa che potrebbe non essere possibile per un’installazione Python soddisfare i requisiti di ogni applicazione. Se l’applicazione A richiede la versione 1.0 di un particolare modulo ma l’applicazione B richiede la versione 2.0, i requisiti sono in conflitto e l’installazione della versione 1.0 o 2.0 non consente l’esecuzione di un’applicazione.

La soluzione a questo problema è quella di creare un ambiente virtuale, un albero di directory autonomo che contiene un’installazione Python per una particolare versione di Python, oltre a una serie di pacchetti aggiuntivi.

*Scritto da Gabriel*

*Cybersecurity semplice (per davvero)*

79 - Quindi, occorre (i primi quattro passaggi da saltare se tutto installato): o Installare Python 🡪 *sudo apt install python3* (per Arch: *sudo pacman -S python*)

o Installare PIP 🡪 *sudo apt install python3-pip* (per Arch: *sudo pacman -S python-pip)* o Installare Virtualenv 🡪 *sudo pip install virtualenv* (per Arch: *sudo pacman -S python virtualenv*)

o Installare Flask 🡪 *sudo pip install flask*

o Spostarsi all’interno della cartella *app* una volta unzippata la cartella

o Attivare l’environment presente 🡪 *source env/bin/activate*

o Installare i pacchetti con *pip* inclusi nel file *requirements* 🡪 *pip install -r requirements.txt* o Una volta finito di usare il virtualenv, si usa *deactivate* per uscire

Aiuti:

1) If you run the application with python application.py you see a meme telling you are in the wrong path. If you inspect the python code, you can see that the flag is contained inside a virtual environment call. Let’s go to inspect the activation file of the environment, which is contained at “./env/bin/activate”. There, you might find the FLAG.

Attenzione

Potrebbe esserci un errore del tipo:

*Cannot import name Container from collections*

Ciò è dovuto, nel caso si usi Python 3.10, all’uso di un pezzo deprecato all’interno della libreria. Sarà quindi necessario Python 3.9.

Su Arch Linux si fa così:

- Installare *yay* 🡪 https://www.lffl.org/2021/01/guida-yay-arch-linux-manjaro.html - Installare Python 3.9 🡪 *yay python39*

Su Ubuntu si fa così:

*sudo apt update*

*sudo apt install software-properties-common*

*sudo add-apt-repository ppa:deadsnakes/ppa*

*sudo apt install python3.9*

Installare *pip* in versione 3.9

- *curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py*

- *python3.9 get-pip.py*

Usare i comandi per avviare l’esercizio come segue (in versione 3.9 e attenzione a mettere sempre *-m*) - *python3.9 -m pip install -r requirements.txt*

- *python3.9 application.py*

Soluzione

Ci viene fornito un file "7Z". Per unzipparlo, seguire questi comandi:

- *sudo apt-get install p7zip-full* (se non già presente ovviamente)

- *7za x myfile.tar.7z*

- *tar -xvf myfile.tar*

La flag è contenuta all'interno, quindi dobbiamo guardarlo all'interno. Se decomprimiamo la cartella, otteniamo una cartella chiamata "app". Se ispezioniamo il suo interno, possiamo vedere un file chiamato "application.py", un'applicazione Flask (uno dei pacchetti installati con Pip). Possiamo ad esempio eseguirla; apriamo un terminale e digitiamo: *python application.py*

*Scritto da Gabriel*