sudo dockerd

chmod -R +rx ./

sudo ./docker\_build.sh

sudo ./docker\_run.sh

sudo docker ps

'or''='

‘or 1=1--

Quello che accade è un’interpolazione (cioè, interpretare uno o più segnaposto con % delle stringhe) della

stringa non sicura, in quanto viene controllato in chiaro allowed\_ip. Si nota da come è stato scritto

allowed\_ip nel ramo else come poter scrivere la nostra stringa; in questo modo, non verrà controllata.

Infatti, questa cosa permette di accedere ai valori del dizionario allowed (quello con le graffe); infatti, il

segno di % permette anche il mapping posizionale all’interno di questo.

Con un po’ di attenzione, si può notare che la flag viene stampata in corrispondenza di “allowed\_flag”.

La stampa utilizza il formato "%(allowed\_flag)s" per stampare il valore contenuto nel dizionario (questo

serve a formattarlo come IP).

Usiamo questa informazione per stampare la flag come si vede qui:

La formattazione %(allowed\_flag)s serve a sfruttare il mapping dizionario e quindi a beccare allowed\_flag

dentro al dizionario.

Questo restituisce un messaggio (come prima), ma questa volta l'IP inserito ci mostra la flag:

You have choose IP INSA{Y0u\_C@n\_H@v3\_fUN\_W1Th\_pYth0n}, but only 8.8.8.8 will receive the key

In generale, inviando all’host ‘%s’ e qualsiasi intero valido come numero di porta, otteniamo tutto il

dizionario ‘allowed’.

CONSOLE:

function x(str, key) {

var map = [];

var res = "";

for (var z=1; z<=255; z++) {

map[String.fromCharCode(z)] = z;

}

for (var j=0, z=0; z<str.length; z++) {

res += String.fromCharCode(

map[str.substr(z,1)] ^ map[key.substr(j,1)]

);

j = (j < key.length) ? j+1 : 0;

}

return res;

}

console.log( x("\_NeAM+bh\_saaES\_mFlSYYu}nYw}", "6") );

OPPURE DA GABRIEL: (pag 85)

A differenza della crittografia, dove è necessario fornire una password per la decifrazione,

nell'offuscamento di JavaScript non esiste una chiave di decifrazione. In effetti, se si cripta JavaScript sul

lato client, sarebbe uno sforzo inutile: se avessimo una chiave di decodifica da fornire al browser, questa

potrebbe essere compromessa e il codice potrebbe essere facilmente accessibile.

Con l'offuscamento, invece, il browser può accedere, leggere e interpretare il codice JavaScript offuscato con la stessa facilità del codice originale non offuscato. Anche se il codice offuscato ha un aspettocompletamente diverso, genererà esattamente lo stesso output nel browser.

Usiamo uno strumento online **(**[**http://jsnice.org/**](http://jsnice.org/)**)**per ottenere una versione più chiara.

Come si vede, però, quella che descrivono loro non è la soluzione completa; manca infatti un pezzo, in quanto il codice è ancora in esadecimale completo e parecchio non chiaro.

Io consiglio [**https://deobfuscate.io/**](https://deobfuscate.io/) andando a fare il check su “Rename Hex Identifiers”, in

maniera tale da ripulire il codice al completo. In questo modo, si può vedere subito che è in atto una funzione di XOR (da cui il nome dell’esercizio).

function x(davesha, jesstin) {

var quatavious = [];

var quiriat = "";

for (z = 1; z <= 255; z++) {

quatavious[String.fromCharCode(z)] = z; //indicizza ogni elemento di quatavious a z (z per 255 volte)

}

;

for (j = z = 0; z < davesha.length; z++) {

quiriat += String.fromCharCode(quatavious[davesha.substr(z, 1)] ^ quatavious[jesstin.substr(j, 1)]);

//esegue lo XOR delle sottostringhe “davesha” e “jesstin” in quatavious (tutta con z) di lunghezza 1

j = j < jesstin.length ? j + 1 : 0; //se j è minore di jesstin.length allora vale “j+1” altrimenti “0”

}

;

return quiriat; //ritorna la stringa quiriat

}

fromCharCode() converte i caratteri Unicode a caratteri

Se utilizziamo il valore admin come nome utente, il codice controlla la password inserita con una password

specifica (cifrata). Tuttavia, la nostra password inserita passa prima attraverso una funzione chiamata xx,

con valore 6. Sembra un algoritmo di crittografia. Abbiamo due possibilità:

1. Sperare che si tratti di una crittografia simmetrica;

Cybersecurity semplice (per davvero)

Scritto da Gabriel

85 2. Definire un algoritmo di inversione.

La prima opzione sembra più veloce, quindi possiamo provarla. L'idea è che, dato che conosciamo la chiave

di crittografia (6), se è simmetrica possiamo usare la stessa funzione con la password cifrata.

cifrata.

Si vada quindi sulla console da tasto destro scrivendo: x("\_NeAM+bh\_saaES\_mFlSYYu}nYw\u001d}", "6")

L'output è: "iNSA{+ThisWasSimpleYouKnow+}".

Riferimento altra soluzione (aggiustata e usata perché dà l’idea chiave di ragionamento)

https://st98.github.io/diary/posts/2017-04-10-inshack-2017.html

La chiave in effetti dell’esercizio è la coppia di controlli:

if(form.id.value=="admin"){if(x(form.pass.value,"6")

Se si sfrutta la vulnerabilità di XOR di usare il valore della stringa \_NeAM+bh\_saaES\_mFlSYYu}nYw\u001d}

sulla base del valore 6 (con /0 che rappresenta il valore nullo), convertiamo encrypted in UTF-8 da

esadecimale (sapendo che la stringa sotto rappresenta il suo valore in esadecimale ed eseguiamo lo XOR

diretto con la stringa, sfruttando il fatto che la chiave sia ripetuta essendo simmetrica):

from pwn import \*

from codecs import \*

encrypted =

bytes.fromhex('7F4E65414D2B62685F73616145535F6D466C535959757D6E59771D7D').decode('utf-8')

print (xor(encrypted, '6\0'))

x("\_NeAM+bh\_saaES\_mFlSYYu}nYw}", "6")

→ iNSA{+ThisWasSimpleYouKnow+}

SHA256