

Introduzione steganografia

Nell'ambito della sicurezza informatica, la steganografia è spesso utilizzata per scopi legittimi, ma può anche essere sfruttata in modo malevolo e, a tal proposito, recentemente, abbiamo incontrato una situazione in cui è stato inviato un file malevolo contenente immagini steganografiche. L'incidente è iniziato con la ricezione di un file zip legato al contesto lavorativo ordinario da una fonte apparentemente legittima. Il file conteneva allegati includenti immagini che a prima vista sembravano normali e innocue. Tuttavia, dopo ulteriori indagini, è stato scoperto che contenevano dati steganografici nascosti che avevano lo scopo di eseguire codice maligno sul sistema. Questo evento mette in evidenza l'importanza di comprendere e difendersi dalle tecniche avanzate utilizzate dai cybercriminali per nascondere malware e altre minacce all'interno di file apparentemente innocui.

La steganografia è un insieme di tecniche che permettono di nascondere un messaggio, garantendo non tanto la confidenzialità della comunicazione, quanto la segretezza della stessa. Potremmo definire quest'ultima come un sistema per trasmettere messaggi segreti. Infatti, il messaggio da trasmettere viene inserito all'interno di un messaggio contenitore che viene inviato tramite il canale di comunicazione. Un osservatore che analizza la comunicazione non noterà quindi nulla di più del messaggio contenitore.

L'invenzione della steganografia è molto antica: se ne trovano tracce già nell'antica Grecia.

È infatti noto un racconto di Erodoto nel quale un messaggio di ribellione viene tatuato sulla testa di uno schiavo e "trasMESSO" una volta che i capelli dello schiavo sono ricresciuti. Sebbene questo racconto venga talvolta presentato come un primo caso d'uso della crittografia, si tratta senz'altro di steganografia. Infatti, la segretezza del messaggio del racconto risiede nella segretezza della comunicazione stessa, non nella distorsione del messaggio.

Tecniche più moderne di steganografia sono state ampiamente utilizzate in tempi recenti.

Basti pensare agli inchiostri invisibili, o alla tecnica delle cifre nulle, nella quale il messaggio viene desecretato unendo le lettere iniziali di ogni parola, periodo o capoverso. Questo tipo di tecniche, all'apparenza molto semplici e scenografiche, sono state utilizzate ad esempio durante la seconda guerra mondiale, per scambiare messaggi tra soldati o all'interno della resistenza.

La steganografia nell'informatica

La rapida evoluzione dell'informatica e delle telecomunicazioni ha portato alla definizione di nuove tecniche di steganografia, adatte ai contenuti digitali. Nel contesto della sicurezza informatica, la steganografia viene vista come una forma di security by obfuscation: la sicurezza della comunicazione è data dalla segretezza della stessa.

Nella steganografia digitale, i messaggi vengono nascosti all'interno di file multimediali: immagini, brani, video o altri messaggi. Nel tempo sono state sviluppate tecniche di steganografia che permettono di nascondere messaggi all'interno di svariati contesti:

- frame inseriti all'interno di video, con velocità elevata
- codifica del messaggio all'interno dell'eco di un file audio
- utilizzo di pacchetti corrotti o header inutilizzati nelle comunicazioni di rete
- introduzione di rumore di fondo

La tecnica più utilizzata nella steganografia digitale è quella del rumore di fondo o del bit meno significativo.

Least Significant Bit

La tecnica del bit meno significativo (LSB), è una delle tipologie più comuni di steganografia digitale, in quanto l'efficacia della tecnica sposa la semplicità dell'implementazione. Questo tipo di tecnica viene utilizzata soprattutto tramite l'uso di immagini e video: la rappresentazione digitale di questo tipo di contenuti multimediali utilizza i pixel, ciascuno rappresentato da un determinato colore, a sua volta definito da una serie di bit.

La modifica del bit meno significativo di un pixel non invalida il colore che rappresenta: la modifica del colore è impercettibile all'occhio umano.

Basandosi su ciò, la tecnica del LSB modifica il bit meno significativo di ogni pixel, codificando il messaggio all'interno del file multimediale senza compromettere l'aspetto del contenuto. Questo tipo di tecnica può essere utilizzata anche con file audio, sebbene il rumore di fondo introdotto potrebbe essere facilmente rilevabile. Per questo motivo si utilizzano altre tecniche per l'utilizzo di questo tipo di contenitori, come l'echo hiding, in cui si codifica il messaggio modificando l'eco all'interno del file.

Rafforzamento segretezza

Nel paragrafo precedente è stato fornito un semplice algoritmo steganografico per la codifica di un messaggio all'interno di un'immagine. Questo potrebbe tuttavia non essere sufficiente per garantire la segretezza del messaggio. Infatti, se un eventuale osservatore (o attaccante) fosse in possesso dell'immagine contenitore non modificata, risulterebbero evidenti le differenze.

Per questo motivo, alcuni algoritmi utilizzano una chiave, nota a mittente e destinatario, all'interno dell'immagine contenitore.

Per rafforzare ulteriormente questa tecnica e complicare la vita agli attaccanti, alcuni algoritmi utilizzano la chiave per decidere come suddividere il messaggio all'interno del file, distribuendo il messaggio tra i bit meno significativi. In ogni caso, quando la dimensione del file lo consente, è sempre consigliabile cifrare il messaggio prima di codificarlo all'interno del contenitore. In questo modo, qualora il messaggio risultasse sospetto ad un attaccante che tenta di analizzarlo, questi si scontrerebbe contro la crittografia.

Steganalisi e distorsione

Un attacco steganografico è un attacco ad una comunicazione che fa uso di steganografia come:

steganalisi

quando si tenta di rilevare la presenza di un messaggio segreto all'interno di un

contenitore

attacco a distorsione

quando vi è un tentativo di modifica del messaggio, in modo da eliminare l'informazione segreta o corromperla.

Nel caso di steganografia LSB, si può effettuare un attacco a distorsione in modo molto semplice, riducendo la definizione dell'immagine. Questo provocherà una modifica della codifica dei pixel dell'immagine, corrompendo il messaggio segreto.

Le tecniche di steganalisi dipendono invece dalle informazioni in possesso dell'attaccante: egli potrebbe conoscere il contenitore originale utilizzato, potrebbe essere entrato in possesso di altri messaggi trasmessi utilizzando lo stesso contenitore o potrebbe trovarsi nella condizione di poter analizzare esclusivamente il messaggio senza alcuna informazione aggiuntiva.

Nel caso di LSB vi sono due tecniche di steganalisi principali:

Attacchi visuali

vengono utilizzati algoritmi di filtering per evidenziare i bit meno significativi in cui si nasconde il messaggio. L'immagine viene poi osservata dall'attaccante per determinare la presenza di un eventuale messaggio.

Attacchi statistici

vengono effettuati confronti statistici sulla base di distribuzioni di frequenza dei colori.

Altri utilizzi della steganografia

La steganografia viene utilizzata in molti Paesi del mondo per aggirare la censura e introdurre una componente di anonimato al messaggio. Viene inoltre utilizzata, in combinazione con la crittografia, per la trasmissione di informazioni confidenziali in ambito politico e militare.

Utilizzando la tecnica del least significant bit, è possibile tramite la steganografia perseguire altri obiettivi. Il watermark è un'informazione che viene comunemente inserita all'interno di file multimediali, comunemente immagini, per evidenziare la paternità del file e preservarne i diritti d'autore. Non è semplice, infatti, controllare un'opera in rete e impedire che altre persone se ne appropriino. Il watermark svolge in questo caso il compito di "firma", contrassegnando in modo visibile o invisibile l'origine del file.

La steganografia può essere utilizzata anche per includere contenuti all'interno di un file, come nel caso di NFT, per nascondere attacchi e per l'esfiltrazione di dati.

Una delle applicazioni della steganografia più interessanti degli ultimi anni, è la codifica di codice all'interno di un'immagine. Un gruppo di ricercatori ha definito un tipo di attacco informatico noto come stegosplit, nel quale l'exploit utilizza come sorgente di codice un'immagine. L'inclusione di codice JS all'interno

dell'immagine passa inosservato all'utente che naviga sul web, ma non viene ignorato dal browser, che interpreta il codice client-side.

In conclusione, la steganografia si rivela un'arma sottile ma efficace che, sebbene abbia origini antichissime, continua ad evolversi e adattarsi alle tecnologie del tempo, con applicazioni nel contesto della cybersecurity che non è possibile ignorare. Il concetto di nascondere e codificare informazioni attraverso metodi inusuali, come avviene nella steganografia, ha paralleli interessanti con l'uso dei linguaggi esoterici. Entrambi i campi condividono un fascino per l'occultamento e la manipolazione creativa delle informazioni. Dove la steganografia si preoccupa di nascondere il contenuto di un messaggio all'interno di altri dati, i linguaggi esoterici spesso nascondono la logica del programma dietro sintassi e meccanismi non convenzionali che introdurrà e spiegherà in modo esaustivo il mio collega Adrian Mario.

Alcune considerazioni finali

La steganografia rappresenta una frontiera affascinante nel campo della cybersecurity. Nel contesto digitale odierno, caratterizzato da una crescente necessità di protezione dei dati, la steganografia emerge come un metodo per salvaguardare le informazioni. Differendo dalla crittografia, che nasconde il significato di un messaggio, la steganografia nasconde l'esistenza stessa del messaggio, offrendo un livello aggiuntivo di sicurezza. Questo approccio può essere particolarmente utile in scenari dove la trasmissione di messaggi criptati potrebbe suscitare sospetti. Nonostante ciò, la steganografia non è esente da vulnerabilità e richiede l'uso responsabile e consapevole per garantire che non sia sfruttata per scopi illeciti.

Linguaggi esoterici

Dall'analisi preliminare delle immagini steganografiche, abbiamo rilevato la presenza di linguaggi esoterici nascosti e protetti da password.

Un linguaggio di programmazione esoterico è caratterizzato da una complessità elevata e da una deliberata mancanza di chiarezza. All'interno del programma da lei inviato, abbiamo identificato l'uso di diversi di questi linguaggi, tra cui:

- ****Brainfuck****: uno dei linguaggi esoterici più estremi, noto per il suo minimalismo e la complessità del codice.
- ****Cow****: un linguaggio basato su comandi del tipo "moo" con variazioni nelle maiuscole e nell'ordine.

Questi linguaggi, sebbene non abbiano una reale utilità pratica, sono popolari tra hacker e programmatori esperti. Vengono spesso utilizzati per testare i limiti della programmazione, come proof of concept per dimostrare teorie, o per puro divertimento. Alcuni sono concepiti come esercizi per comprendere meglio il funzionamento di un calcolatore.

Inoltre, è stato rilevato l'uso del formato Base64 per la decodifica di testi, che sono stati successivamente tradotti in codice binario e anche in testo ASCII.

Questa scoperta evidenzia l'adozione di tecniche avanzate e l'uso di strumenti sofisticati all'interno del programma analizzato.

Un linguaggio di programmazione esoterico è una tipologia di linguaggi di programmazione particolarmente complessi e volutamente meno chiari possibile. Questi linguaggi, popolari fra gli hacker e gli utenti più che abili, non hanno una vera utilità nel mondo reale, ma sono generalmente concepiti per mettere alla prova i limiti della programmazione su computer, come proof of concept per dimostrare una teoria o per semplice divertimento. Alcuni, invece, sono concepiti come esercizio per comprendere meglio il funzionamento di un calcolatore.

Turing Tarpit

È un linguaggio di programmazione Turing-completo il cui numero di comandi, operatori o oggetti è molto piccolo.

Stateful encoding

In questo metodo di programmazione, le istruzioni che il linguaggio di programmazione mette a disposizione sono in un elenco predefinito, e per eseguire un comando sono necessari due passaggi:

- localizzarne la posizione nell'elenco (ad esempio in reMorse il salvataggio sullo stack è l'istruzione "I" dell'elenco)
- lanciarlo, in modo da applicarne gli effetti

L'elenco delle operazioni può essere sia statico – come in reMorse[3] e THRAT[4] o dinamico, come in reMorse4ever.

Ecco un esempio sulla base di reMorse o THRAT:

Select Next Operation in list

Perform Operation

Funge

Una funge è un programma le cui istruzioni sono disposte a formare una figura che si sviluppa bidimensionalmente, e la sequenza delle istruzioni è stabilita dalla direzione di movimento di un "puntatore" su tale figura. Il programmatore ha a disposizione, oltre che istruzioni sui dati, anche comandi per modificare posizione e direzione di movimento del puntatore.

Non determinismo

In un linguaggio non deterministico, l'esecuzione delle istruzioni non è garantita con certezza, ma solo con una certa probabilità. In linguaggi come questi, anche tentare di ottenere un risultato è un compito arduo, dato che non si ha la certezza che un'istruzione sia eseguita o meno, e quindi non è possibile prevederne il funzionamento.

Di seguito alcuni esempi del classico programma hello world o di singole istruzioni scritte con linguaggi esoterici:

LOLCODE

è stato creato per imitare il linguaggio dei "lolcat"

Befunge

è un linguaggio nel quale i programmi sono "distesi" in un array bidimensionale, nel quale il puntatore viene fatto scorrere.

Brainfuck

è uno dei linguaggi esoterici più estremi, nel quale minimalismo e offuscazione del codice sono portati a livelli molto elevati.

Chef

è stato sviluppato in modo che i suoi programmi sembrano ricette di cucina; per esempio, il seguente costrutto inserisce un valore in uno stack:

```
Put cinnamon into 2nd mixing bowl
```

Cow

è un linguaggio basato su comandi del tipo "moo" variando le maiuscole e l'ordine.

FALSE

è un linguaggio basato sugli stack, dotato di comandi a singolo carattere e variabili.

Shakespeare

le istruzioni ricalcano gli scritti dell'omonimo drammaturgo. Ad esempio, la frase seguente indica un punto nel listato che può essere raggiunto tramite un'istruzione simile a GOTO:

```
Act I: Hamlet's insults and flattery.
```

Malbolge

è talmente complesso che il primo programma "Hello world" funzionante è arrivato due anni dopo il suo rilascio:

```
(=<`:9876Z4321UT.-Q+*)M'&%$H"!~}|Bzy?=|{z]KwZY44Eq0/{mIk**  
hKs_dG5[m_BA{?-Y;;Vb'rR543IM}/.zHGwEDCBA@98\6543W10/.R,+O<
```

Linguaggi esoterici: hanno qualche utilità o sono inutili?

I linguaggi di programmazione esoterici, spesso creati per scopi artistici, sperimentali o come puzzle mentali, non sono tipicamente utilizzati per la programmazione pratica. Tuttavia, possono avere alcuni vantaggi in contesti di cybersecurity, specialmente riguardo alla sicurezza dei dati. Ecco come potrebbero contribuire:

1. Oscurità del Codice

I linguaggi esoterici sono difficili da leggere e comprendere per chi non li conosce, il che può aggiungere un livello di offuscamento al codice. Questo rende più difficile per un attaccante capire e modificare il codice:

Protezione del Codice Sorgente

Scrivere parti critiche del software in un linguaggio esoterico può rendere più difficile la comprensione e l'analisi del codice da parte di un attaccante.

Esempio

Brainfuck o Malbolge sono così complessi da leggere e scrivere che anche un semplice programma può essere quasi indecifrabile senza una conoscenza approfondita.

2. Diversificazione del Stack Tecnologico

Utilizzare un linguaggio esoterico all'interno di un sistema più ampio può aumentare la complessità dell'attacco:

Barriere Aggiuntive

Un attaccante deve comprendere e saper operare con un linguaggio non convenzionale per compromettere l'intero sistema, aumentando il tempo e lo sforzo richiesto.

Esempio

Un sistema che utilizza linguaggi standard per la maggior parte del codice ma implementa funzioni critiche in un linguaggio esoterico crea una barriera significativa.

3. Riduzione delle Vulnerabilità Comunemente Conosciute

I linguaggi esoterici possono non essere soggetti alle stesse vulnerabilità dei linguaggi più comuni, come buffer overflow o SQL injection:

Incompatibilità con Exploit Noti

La mancanza di una base di conoscenza ampia sugli exploit specifici per un linguaggio esoterico riduce la probabilità di attacchi riusciti.

Esempio

Un linguaggio come Whitespace, che utilizza solo spazi bianchi per la sintassi, non ha le stesse vulnerabilità tipiche dei linguaggi a caratteri visibili.

4. Utilizzo come Strumento di Offuscamento

I linguaggi esoterici possono essere utilizzati per offuscare dati sensibili:

Offuscamento dei Dati: Codificare dati sensibili in un linguaggio esoterico può rendere più difficile per un attaccante estrarre informazioni utili.

Esempio: Utilizzare un linguaggio esoterico per crittografare o offuscare chiavi di accesso o altre informazioni sensibili.

5. Formazione e Ricerca

L'utilizzo di linguaggi esoterici può aiutare nella formazione e nella ricerca in cybersecurity:

Esercizi di Pensiero Laterale: Studiare e lavorare con linguaggi esoterici può migliorare le capacità di problem-solving e di pensiero laterale, utili per identificare e risolvere problemi di sicurezza.

Esempio: Progetti accademici e competizioni di hacking che utilizzano linguaggi esoterici possono fornire una formazione unica e stimolante.

Limitazioni e Considerazioni

Nonostante i vantaggi sopra elencati, ci sono delle limitazioni e considerazioni da tenere presente:

Manutenibilità

Il codice scritto in linguaggi esoterici può essere estremamente difficile da mantenere e aggiornare.

Prestazioni

Molti linguaggi esoterici non sono ottimizzati per l'efficienza e possono avere prestazioni molto inferiori rispetto ai linguaggi convenzionali.

Adattabilità

L'integrazione di linguaggi esoterici in sistemi esistenti può essere complessa e richiedere un notevole sforzo.

In sintesi, mentre i linguaggi di programmazione esoterici non sono una panacea per la sicurezza dei dati, possono aggiungere un ulteriore livello di protezione attraverso l'offuscamento, la diversificazione tecnologica e la riduzione delle vulnerabilità comunemente conosciute. Tuttavia, è importante bilanciare questi benefici con le sfide pratiche legate alla loro implementazione e manutenzione.

Funzionamento brainfuck

Esempio di codice e della sua struttura

Codice

```
+++++ [ > ++++++ < - ] > +++++.
```

+++++ -> Incrementa il valore del primo byte nella memoria di 6. Ora la cella 0 contiene 6.

[-> Inizia un ciclo che continua finché il valore della cella corrente (cella 0) è diverso da zero.

> ++++++ -> Si sposta alla cella successiva (cella 1) e incrementa il suo valore di 10. Ora la cella 1 contiene 10.

< - -> Torna alla cella 0 e decrementa il suo valore di 1. Ora la cella 0 contiene 5.

] -> Fine del ciclo. Torna all'inizio del ciclo perché la cella 0 non è zero. Ripete i passi dal 2 al 4. Ogni volta che il ciclo viene eseguito, il valore della cella 1 viene incrementato di 10 e il valore della cella 0 viene decrementato di 1.

Dopo 6 iterazioni, il ciclo si interrompe perché la cella 0 raggiunge il valore 0. A questo punto, il valore della cella 1 è 60 ($10 * 6$).

> +++++ -> Si sposta alla cella 1 e incrementa il suo valore di 5. Ora la cella 1 contiene 65.

. -> Stampa il carattere corrispondente al valore ASCII della cella corrente (cella 1). Il valore 65 corrisponde alla lettera "A" in ASCII.

Spiegazione del codice datashield

Incrementi di valori e movimenti tra celle:

- **>**: si sposta alla cella successiva.
- **<**: si sposta alla cella precedente.
- **+**: incrementa il valore della cella corrente.
- **-**: decrementa il valore della cella corrente.

Cicli per creare valori specifici:

- **[...]**: crea un ciclo che continua finché il valore della cella corrente è diverso da zero. Questo è usato per creare valori più grandi rapidamente.

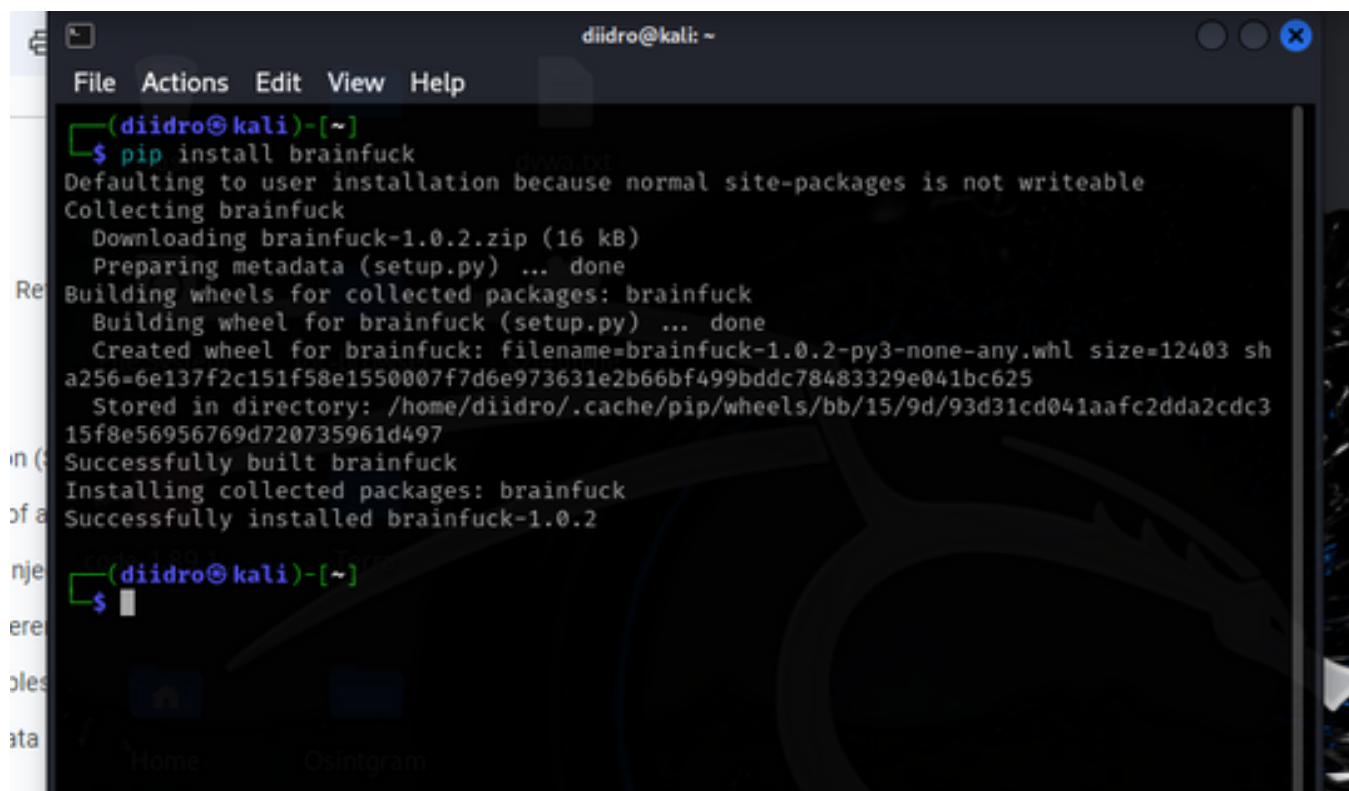
Output di caratteri:

- **.**: stampa il carattere corrispondente al valore ASCII della cella corrente.

bisogna ricordare che: ogni gruppo di istruzioni calcola e stampa UN CARATTERE della stringa output.

Dove lo lanciamo?

Ovviamente dalla kali, basta installare l'interprete, proseguiamo all'installazione.



```
(diidro@kali)-[~]
$ pip install brainfuck
Defaulting to user installation because normal site-packages is not writeable
Collecting brainfuck
  Downloading brainfuck-1.0.2.zip (16 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: brainfuck
  Building wheel for brainfuck (setup.py) ... done
  Created wheel for brainfuck: filename=brainfuck-1.0.2-py3-none-any.whl size=12403 sha256=6e137f2c151f58e1550007f7d6e973631e2b66bf499bddc78483329e041bc625
  Stored in directory: /home/diidro/.cache/pip/wheels/bb/15/9d/93d31cd041aafc2dda2cdc315f8e56956769d720735961d497
Successfully built brainfuck
Installing collected packages: brainfuck
Successfully installed brainfuck-1.0.2

(diidro@kali)-[~]
$
```

In alternativa si possono usare interpreti online come:

Dcode (<https://www.dcode.fr/reversefuck-language>)

Paiza.io

copy.sh

ReverseFuck looks like **BrainFuck** but has the particularity of inverting the **BrainFuck** language operators to make it even more incomprehensible: the + becomes a -, an opening bracket [becomes a closing bracket], and so on. Here is the table to convert BF to **BrainFuck** and conversely :

ReverseFuck	<u>Brainfuck</u>	Operation
+	-	Pointer value - 1
-	+	Pointer value + 1
>	<	Pointer position - 1
<	>	Pointer position + 1
]	[Loop while pointer > 0
[]	End loop
.	,	Store input in pointer position
,	.	Output <u>ASCII</u> value of pointer

Semplici

Loop semplice

,[.]

Un ciclo continuo che prende del testo in input dalla tastiera e lo scrive sullo schermo. Da notare che si assume che la cella sia impostata a 0 quando un comando ';' viene eseguito dopo la fine dell'input (alle volte chiamata end-of-file o "EOF"); le implementazioni differiscono in questo punto. Per implementazioni che impostano la cella a -1 o EOF, oppure che non modificano il valore della cella, questo programma andrebbe scritto rispettivamente "++[-,]+" o "[.[-],]"

Manipolazione dei puntatori

>,[.>]

Una versione dell'ultimo esempio che salva anche tutto l'input in un array per uso futuro, muovendo il puntatore ogni volta.

Sommare

[->+<]

Questo somma la locazione corrente (distruttivamente, essa viene messa a zero) alla locazione

successiva.

Istruzioni condizionali

```
,-----[-----,-----]
```

Questo esempio prenderà input scritti in minuscolo dalla tastiera e li farà diventare maiuscoli. Per uscire, premere il tasto invio.

In primo luogo, inseriamo il primo carattere usando il comando `,` e immediatamente sottraiamo 10 da esso. (Molti, ma non tutti, i programmi Brainfuck usano 10 per indicare il tasto di ritorno a capo.) Se l'utente preme invio, l'istruzione di ciclo (`[]`) salterà dopo la fine del ciclo, perché setteremo il primo byte a zero. Se il carattere inserito non era 10, assumeremo che esso fosse una lettera minuscola, ed entreremo nel ciclo, nel quale sottrarreemo un altro 22 da esso, per un totale di 32, il quale è la differenza tra una lettera ASCII minuscola e la corrispondente lettera maiuscola.

Successivamente lo visualizzeremo. Ora inseriamo il prossimo carattere, ed ancora sottraiamo 10. Se questo carattere fosse un [linefeed](#), usciamo dal ciclo; altrimenti, ritorneremo all'inizio del ciclo, sottrarreemo un altro 22, lo visualizzeremo, e così via. Quando usciamo dal ciclo, il programma termina, siccome non ci sono più comandi.

Copiare un [byte](#)

Brainfuck non include nessuna operazione per copiare byte. Questo deve essere fatto con i costrutti di ciclo e gli operatori aritmetici. Muovere un byte è abbastanza semplice; muovere il valore di `[0]` a `[1]` può essere fatto come segue:

```
[->+<]
```

Comunque, questo resetta il valore di `[0]` a 0. Possiamo ripristinare il valore di `[0]` dopo la copia prendendo vantaggio dell'abilità di copiare un valore in due posti alla volta. Per copiare il valore di `[0]` in entrambi `[1]` e `[2]` è semplice:

```
[->+>+<<]
```

Possiamo prendere vantaggio di questo per ripristinare il valore `[0]`. Quindi, possiamo non distruttivamente copiare `[0]` a `[1]` (usando `[2]` come variabile temporanea) così come segue:

```
[->+>+<<]>>[-<<+>>]
```

Complessi

Somma

```
,>++++++[<----->-],,[<+>-],<,>.
```

Questo programma aggiunge due numeri a singola cifra e visualizza il risultato correttamente se anch'esso è composto da una sola cifra:

4+3

(Ora le cose iniziano a diventare un po' più complicate. Noi possiamo riferirci ai byte nell'array come `[0]`, `[1]`, `[2]`, e così via.)

Il primo numero è inserito in `[0]`, e gli si sottrae 48 per ottenere la cifra corrispondente (i codici ASCII per i numeri da 0 a 9 sono infatti quelli da 48 a 57). Questo è fatto mettendo un 6 in `[1]` ed usando un ciclo per sottrarre 8 da `[0]` il numero corrispondente di volte. (Questo è un metodo comune di sommare o sottrarre grandi numeri) Successivamente, si inserisce il segno di somma in `[1]`; si inserisce infine il secondo numero, sovrascrivendo il simbolo di somma.

Il ciclo successivo `[<+>-]` fa il vero lavoro, muovendo il secondo numero sopra il primo, sommandoli insieme ed azzerando `[1]`. A ogni loop, esso aggiunge uno a `[0]` e sottrae uno da 1; così `[1]` viene alla fine azzerato, e la quantità aggiunta a `[0]` è esattamente quella rimossa da `[1]`. Viene quindi inserito un valore di ritorno in `[1]`. (Non stiamo controllando possibili errori sull'input.)

Poi il puntatore viene spostato indietro a `[0]`, che viene visualizzato. (`[0]` è ora $a + (b + 48)$, siccome non abbiamo corretto b ; questo valore è identico ad $(a + b) + 48$, che è ciò che vogliamo.) Ora il puntatore è spostato a `[1]`, il quale contiene il risultato; esso viene infine visualizzato, terminando l'algoritmo

Tool Kali per encode e decode: steghide

Per installazione tool usare -> `sudo apt-get install steghide`

`man steghide` -> comandi utili

password - "paolo"



CONCLUSIONE

In conclusione, il progetto di rete per l'azienda Theta offre una soluzione sicura e robusta per proteggere dati e risorse critiche. La suddivisione in zone ben definite permette di applicare misure di sicurezza specifiche, migliorando la protezione contro minacce interne ed esterne.

L'implementazione delle VLAN con privilegi minimi e la segregazione della rete interna riducono la superficie di attacco, applicando il principio del privilegio minimo. L'uso di un IPS (Intrusion Prevention System) fornisce protezione in tempo reale contro le minacce.

La DMZ, protetta da firewall e con l'uso di un reverse proxy, aggiunge un ulteriore livello di sicurezza, migliorando l'efficienza e il bilanciamento del carico. Questa architettura non solo soddisfa gli standard di sicurezza di Theta, ma offre una soluzione scalabile e gestibile per le future esigenze.

Grazie per l'attenzione.

Rimaniamo a disposizione per rispondere a qualsiasi domanda e discutere ulteriormente i dettagli del progetto.

