

Nascondere e proteggere: Steganografia e Linguaggi Esoterici per la protezione avanzata dei dati.

Nella crescente battaglia per la sicurezza delle informazioni sensibili, l'evoluzione della tecnologia ha reso indispensabile l'adozione di metodi sempre più sofisticati per proteggere e nascondere dati critici. In questo contesto, due approcci emergono: la steganografia, antica arte del nascondere messaggi all'interno di altri dati, e l'uso di linguaggi di programmazione esoterici, noti per la loro complessità e 'oscurità' sintattica.

Segreti celati: dall'antica arte della steganografia alle sue applicazioni digitali

Introduzione alla Steganografia

La steganografia, dal greco antico στεγανός, "coperto, nascosto", e γράφειν, "scrivere", è l'arte o, meglio, la scienza, di occultare informazioni all'interno di altri dati o supporti comunicativi.

Questa disciplina si distingue dalla crittografia: mentre la crittografia si focalizza sulla cifratura del messaggio per renderlo incomprensibile a chi non possiede la chiave di decifrazione, la steganografia nasconde l'esistenza stessa del messaggio. Il risultato è una comunicazione che, se ben eseguita, rimane invisibile e indiscernibile tranne che per il mittente e il destinatario.

Storia della Steganografia

La steganografia (o la semplice crittografia in epoca più antica) ha radici molto antiche e ha sempre giocato un ruolo significativo in vari contesti storici. La capacità di trasmettere messaggi in modo invisibile ha avuto, e continua ad avere, un impatto significativo nelle strategie di comunicazione, specialmente in contesti di conflitto e censura.

Uno dei primi esempi risale al 440 a.C., quando lo storico greco Erodoto documentò l'uso di messaggi nascosti scritti sulla pelle rasata di uno schiavo, oppure, grazie ai testi di Plutarco, sappiamo dell'uso della cosiddetta scitale, o scitala, ovvero un bastone attorno al quale veniva avvolto un nastro, solitamente in papiro o cuoio, contenente un messaggio, che poteva essere letto solo se si possedeva un bastone di uguale diametro, sul quale il destinatario avvolgeva il cartiglio.

Proseguendo l'exkursus e passando all'epoca romana, è sicuramente molto famoso il Cifrario di Cesare, ovvero un metodo di scrittura cifrata che Giulio Cesare era solito utilizzare durante le sue campagne di guerra per inviare messaggi ai suoi generali, che avevano quindi necessità di mantenere il contenuto segreto.

Nel Medioevo iniziò la “moda” degli inchiostri simpatici, ovvero inchiostri invisibili che, se scaldati o mescolati con sostanze chimiche, rivelavano un messaggio nascosto.

Iniziamo ad avere esempi più rilevanti e calzanti di steganografia sicuramente più vicino al nostro secolo, quindi durante la Prima e Seconda Guerra Mondiale, dove questa veniva applicata con la tecnica dei microdot, ovvero la compressione di informazioni importanti in un piccolo punto. I tedeschi erano particolarmente ferrati in materia e riuscivano a

ridurre intere pagine di informazioni in piccoli punti, che, a loro volta, potevano essere nascosti in un testo senza essere rilevati da un occhio non attento.

Durante la Guerra Fredda, sia gli USA che l'URSS utilizzarono tecniche steganografiche per nascondere documenti e fotografie in oggetti di uso quotidiano inviati attraverso le frontiere. A volte, queste informazioni venivano nascoste dentro regali diplomatici o oggetti d'arte.

Con l'avanzamento della tecnologia, anche le tecniche di steganografia si sono sviluppate, venendo utilizzate, quindi, nei contesti digitali. Ad esempio, durante le proteste post-elettorali in Iran nel 2009/2010, i manifestanti usavano la steganografia per nascondere e distribuire fotografie e video fuori dal Paese, aggirando la severa censura di internet.

Modelli Steganografici

Un modo per classificare la steganografia è quello di catalogarla in base al metodo impiegato per occultare il messaggio. Questi modelli non solo dimostrano la versatilità e l'efficacia della steganografia, ma anche come possano essere adattati a seconda delle necessità e dei contesti. I due principali approcci sono il modello di Cover-Modification e il modello di Cover-Selection.

Modello di Cover-Modification

Questo modello implica modifiche dirette al contenitore per incorporare il messaggio segreto e le modifiche devono essere indistinguibili all'osservatore.

La steganografia Iniettiva è la tecnica più utilizzata, e consiste nell'inserire il messaggio all'interno di un altro messaggio creandone un ultimo indistinguibile dall'originale.

Durante la Seconda Guerra Mondiale, i servizi segreti britannici svilupparono tecniche avanzate di Cover-Modification, tra cui la stampa di mappe segrete su seta che i piloti potevano nascondere facilmente. Queste mappe erano stampate in modo tale che, senza una luce specifica, apparivano come normali fazzoletti.

Nel contesto digitale, il modello di Cover-Modification viene applicato modificando i bit meno significativi (LSB - Least Significant Bit) delle immagini digitali per nascondere dati critici. Questo metodo è popolare per la sua semplicità e l'ampia applicabilità nei formati file comuni.

Altri esempi di Cover-Modification sono: la steganografia Sostitutiva che si basa sulla sostituzione di parti di rumore presenti nei canali di comunicazione con il messaggio segreto. La modifica è impercettibile agli esseri umani, ma può essere rilevata tramite appositi dispositivi; la steganografia Costruttiva, molto simile alla sostitutiva, ha come obiettivo quello di sostituire il rumore originale presente naturalmente nelle comunicazioni con il messaggio segreto; la steganografia Generativa consiste, invece, nel generare un contenitore proprio per il messaggio segreto, costruendone uno per nascondere nel miglior modo possibile.

Modello di Cover-Selection / Steganografia Selettiva:

Al contrario del precedente, che in qualche modo va ad alterare un oggetto esistente, il modello di Cover-Selection utilizza

oggetti che, per loro natura, contengono già elementi o pattern che possono essere interpretati come un messaggio codificato, o che meglio si prestano a nascondere.

Un aneddoto poco noto ma sicuramente affascinante proviene dall'Olanda del XVII secolo. I codici venivano nascosti nella selezione e nell'arrangiamento dei tulipani all'interno dei giardini. Poiché diversi colori e schemi di tulipani potevano essere associati a specifici messaggi, facendo una semplice passeggiata in un giardino si potevano trasmettere informazioni segrete.

In età moderna sono molteplici gli artisti che usano la Cover-Selection creando dipinti che, visti sotto angolazioni specifiche o con illuminazioni particolari, rivelano messaggi nascosti, anche di rilevanza politica e sociale.

È importante sottolineare che questo è un approccio per la maggior parte teorico, poiché nella pratica risulta essere dispendioso e non permette di nascondere grandi quantità di informazioni.

La Steganografia Digitale

La steganografia Digitale è l'applicazione delle tecniche steganografiche ai media digitali, come file audio, video, immagini e testi.

Con l'avanzamento della tecnologia, la steganografia digitale ha acquisito una rilevanza e una sofisticazione maggiore, diventando uno strumento importante sia per la sicurezza informatica sia per attività meno etiche.

LSB (Least Significant Bit)

Questa tecnica, sicuramente la più diffusa, prevede la modifica dei bit meno significativi nelle immagini digitali o nei file audio. Poiché questi cambiamenti sono minimi, sono impercettibili all'occhio o all'orecchio umano.

Nelle immagini, ad esempio, ogni pixel ha un colore differente e, cambiando un bit in ogni pixel, il colore risulterà variato in modo non significativo e il contenuto dell'immagine finale sarà identico all'originale, anche se essa è stata manipolata. In generale, questo vale per ogni media, quindi anche video o audio, poiché le piccole manipolazioni vengono tollerate e il contenuto può essere visualizzato.

Una delle applicazioni famose di LSB include la trasmissione di messaggi segreti in immagini pubblicate sui social media durante le proteste politiche. Molti attivisti usano questa tecnica per coordinare azioni senza attirare l'attenzione delle autorità.

Mascheramento con trasformazioni di Fourier

Questa particolare tecnica utilizza le trasformazioni matematiche di Fourier per nascondere informazioni nelle trasformazioni di frequenza di file audio e video.

Alcune band musicali hanno sperimentato con la steganografia audio per nascondere messaggi nei loro album. Questi messaggi potevano essere rivelati solo usando software specifici che analizzavano lo spettro del suono.

Steganografia nel Testo

Questa tecnica prevede metodi come l'alterazione degli spazi bianchi, l'uso di tabulazioni o la modifica della punteggiatura per

formare codici, quindi la creazione di file che contengono dati nascosti nelle piccole variazioni testuali.

Steganografia in Reti Neurali

È una innovazione molto recente e prevede l'uso di reti neurali, delle quali viene sfruttata la complessità e la versatilità per nascondere informazioni in modo sofisticato e difficile da rilevare.

Viene utilizzata per comunicazioni sicure, protezione della proprietà intellettuale, verifica dell'integrità dei modelli distribuiti, autenticazione, e tracciabilità per combattere la pirateria. Nascondendo informazioni nei parametri dei modelli, si garantisce che solo utenti autorizzati possano accedere o rilevare i dati nascosti, migliorando la sicurezza e la protezione dei dati.

CREAZIONE DI UNA IMMAGINE STEGANOGRAFICA

Ho voluto creare una immagine steganografica e inserire all'interno di essa un messaggio segreto.

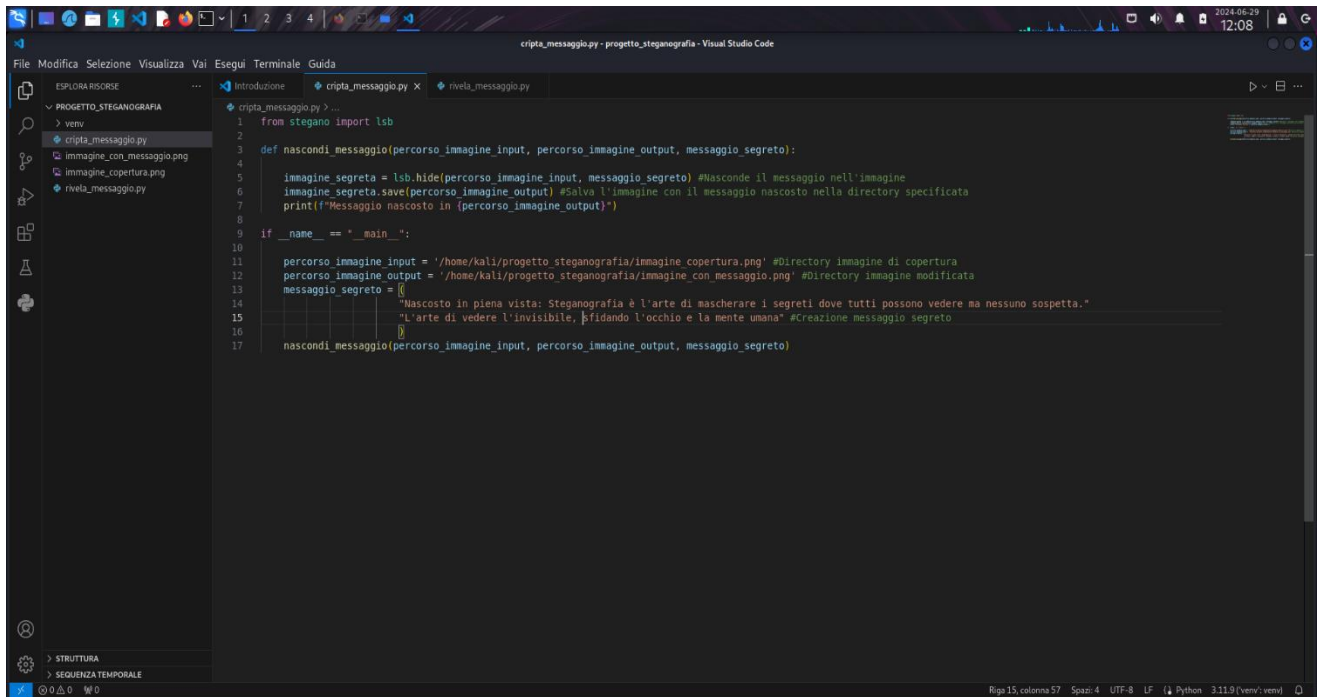
Ho pensato di scrivere un codice in Python e di usare, quindi, la libreria 'Stegano'. Il metodo che userò sarà quello del LSB, ovvero nasconderò un messaggio segreto all'interno di una immagine digitale, non alterando l'aspetto originale della stessa. Infine, sempre con l'ausilio della stessa libreria, si rivelerà il messaggio segreto per dimostrare che le informazioni possono essere occultate, ma anche recuperate, da immagini apparentemente innocue.

Le due funzioni principali che ho definito sono:
nascondi_messaggio, che utilizza 'lsb.hide' e inserisce il messaggio nell'immagine principale, di cui ho inserito la directory, e ne salva una risultante nella stessa cartella, e rivela_messaggio che, riprendendo il percorso dell'immagine con la modifica, usa 'lsb.reveal' per estrarre il messaggio e scriverlo sul terminale. Per mia scelta ho deciso di fare due script separati, ma poteva essere fatto anche uno unico.

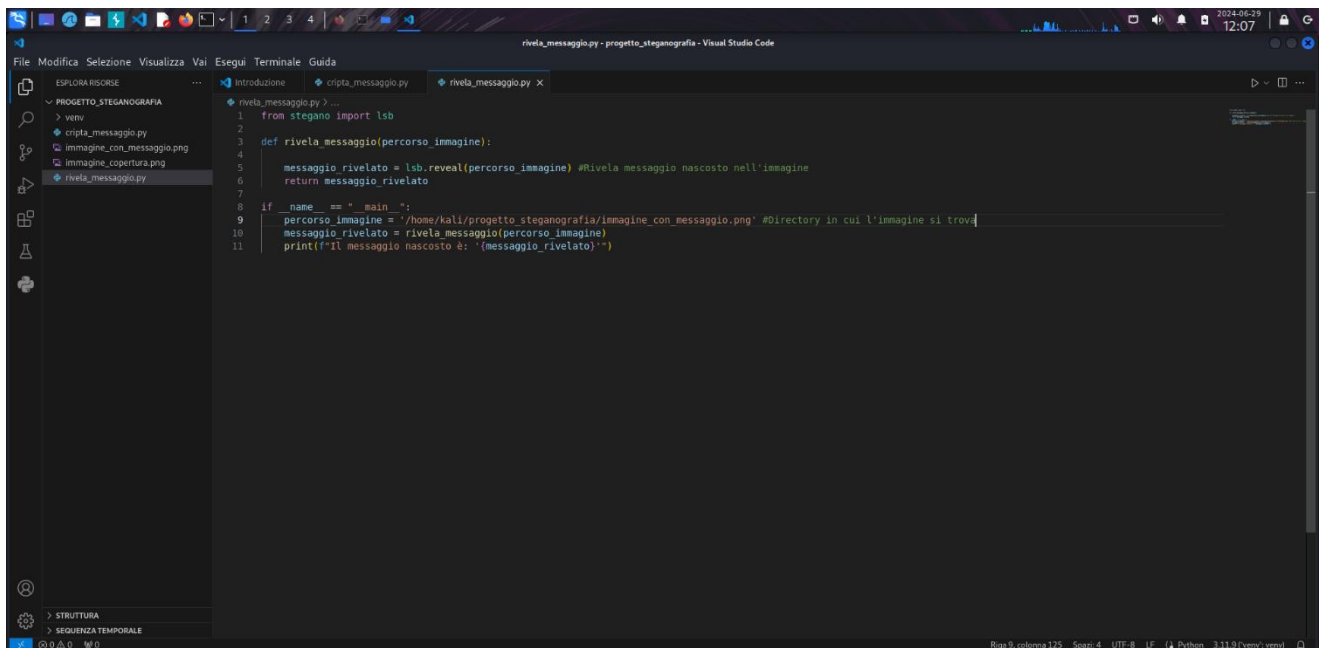
Parto dalla scelta dell'immagine di copertura, e ne scelgo una ad alta definizione, che abbia abbastanza pixel da essere modificati e che non risenta quindi delle manipolazioni. La inserisco all'interno di una cartella, dove verranno successivamente salvati anche i due script e l'immagine finale, in quanto mi servirà una directory da inserire nei codici per dare una posizione delle immagini.



Poi, come dicevo prima, ho creato due codici, uno per nascondere il messaggio e uno per rivelarlo, e in entrambi inserisco la libreria 'stegano' dalla quale importo 'lsb', di cui userò le funzioni.

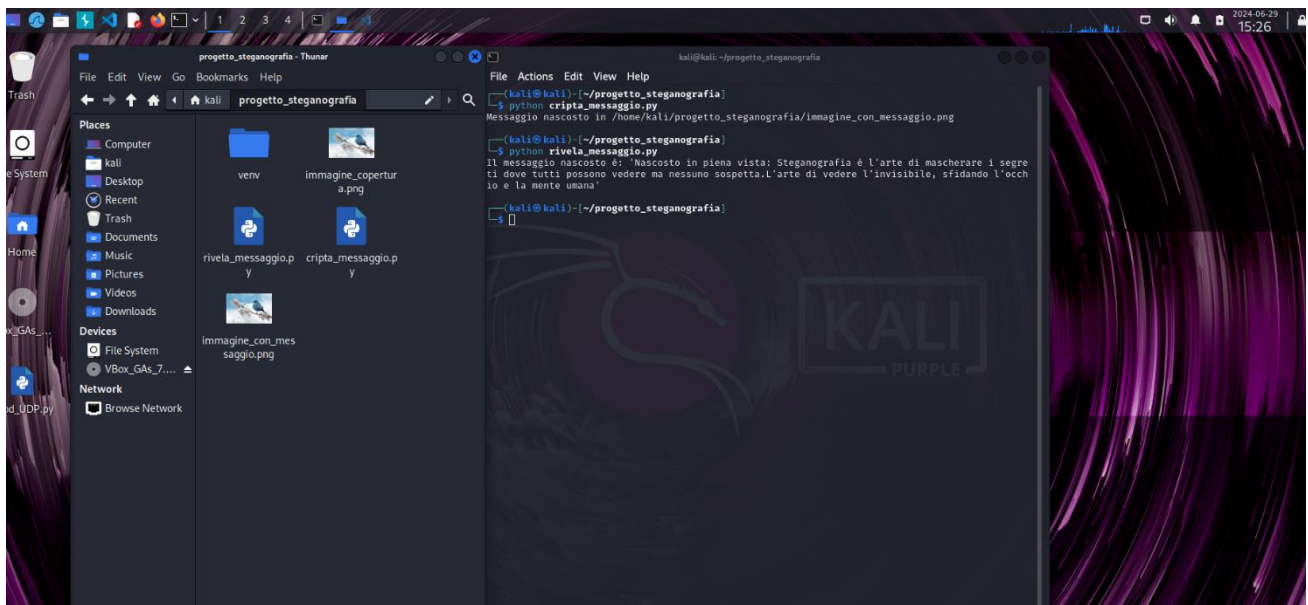


```
1 from stegano import lsb
2
3 def nascondi_messaggio(percorso_immagine_input, percorso_immagine_output, messaggio_segreto):
4
5     immagine_segreto = lsb.hide(percorso_immagine_input, messaggio_segreto) #Nasconde il messaggio nell'immagine
6     immagine_segreto.save(percorso_immagine_output) #Salva l'immagine con il messaggio nascosto nella directory specificata
7     print(f"Messaggio nascosto in {percorso_immagine_output}")
8
9 if __name__ == "__main__":
10
11     percorso_immagine_input = '/home/kali/progetto_steganografia/immagine_copertura.png' #Directory immagine di copertura
12     percorso_immagine_output = '/home/kali/progetto_steganografia/immagine_con_messaggio.png' #Directory immagine modificata
13     messaggio_segreto = (
14         "Nascosto in piena vista: Steganografia è l'arte di mascherare i segreti dove tutti possono vedere ma nessuno sospetta."
15         "L'arte di vedere l'invisibile, fidando l'occhio e la mente umana" #Creazione messaggio segreto
16     )
17     nascondi_messaggio(percorso_immagine_input, percorso_immagine_output, messaggio_segreto)
```



```
1 from stegano import lsb
2
3 def rivela_messaggio(percorso_immagine):
4
5     messaggio_rivelato = lsb.reveal(percorso_immagine) #Rivela messaggio nascosto nell'immagine
6     return messaggio_rivelato
7
8 if __name__ == "__main__":
9     percorso_immagine = '/home/kali/progetto_steganografia/immagine_con_messaggio.png' #Directory in cui l'immagine si trova
10    messaggio_rivelato = rivela_messaggio(percorso_immagine)
11    print(f"il messaggio nascosto è: {messaggio_rivelato}")
```

Infine, avvio l'esecuzione come da immagine sottostante. Infatti, il primo comando 'python cripta_messaggio.py' mi crea una immagine detta 'immagine_con_messaggio.png' e la inserisce nella directory che ho indicato, e, successivamente, con il comando 'python rivela_messaggio.py' si stamperà a schermo il messaggio nascosto nell'immagine.



Esolang e Sicurezza Informatica: un'analisi dei linguaggi di programmazione esoterici tra Professionalità e Gioco – ‘Security Through Obscurity’

I linguaggi di programmazione esoterici, spesso abbreviati come esolang, ci invitano a esplorare gli angoli più insoliti e creativi del mondo della programmazione.

Questi linguaggi si distinguono nettamente dai linguaggi convenzionali come Python o Java, che sono progettati per essere pratici, efficienti e comprensibili. Gli esolang, invece, si distanziano deliberatamente da queste precise caratteristiche, spesso con l'intento di divertire, sfidare intellettualmente, o esplorare i confini teorici dei linguaggi di programmazione.

La loro esistenza rivela non solo le infinite possibilità della programmazione, ma anche come il processo di programmazione stesso può essere reinterpretato e manipolato a nostro piacimento.

Sviluppandosi principalmente per esperimenti o come forma di espressione artistica, gli esolang offrono una prospettiva unica e, perché no, umoristica sull'informatica, dimostrando che la programmazione può trascendere la sua natura tecnica per diventare un vero e proprio esercizio creativo e un mezzo di espressione artistica.

Origini e sviluppo

La storia degli esolang è un racconto affascinante di ingegneria e creatività, che si intrecciano per superare i confini di ciò che è tecnicamente possibile e applicabile alla realtà, offrendo un approccio più giocoso alla programmazione.

In generale, il loro sviluppo è segnato da seri tentativi e sperimentazione ludica, dimostrando che la programmazione può essere sia scienza, sia arte.

L'introduzione ad un concetto vero e proprio di linguaggio di programmazione esoterico si ha nel 1972 con INTERCAL, acronimo di "Compiler Language With No Pronounceable Acronym", che fu progettato da due studenti di Princeton per sfidare le convenzioni sociali. Esso si diversificava totalmente dai linguaggi tradizionali, e inseriva nel codice una sintassi bizzarra e requisiti sicuramente non ortodossi, come l'utilizzo del comando 'PLEASE', una 'parola magica' che rendeva il programma più educato e quindi funzionante su ogni computer.

```
source code
1 DO ,1 <- #67
2 DO ,2 <- #105
3 DO ,3 <- #97
4 DO ,4 <- #111
5 DO ,5 <- #33
6 PLEASE READ OUT ,1
7 PLEASE READ OUT ,2
8 PLEASE READ OUT ,3
9 PLEASE READ OUT ,4
10 PLEASE READ OUT ,5
11 PLEASE GIVE UP
12
```

Questo nell'immagine è un semplice esempio di come INTERCAL stamperebbe la parola 'Ciao!'. Il 'DO' assegna una variabile ad un valore, che a sua volta corrisponde ad una lettera nel codice ASCII. Il 'PLEASE', come si vede, non aggiungere nulla se non un tocco umoristico.

Durante gli anni '90, si ebbe una crescita significativa dell'interesse per questo tipo di linguaggi. Proprio in questo decennio, nel 1993, nacque uno dei più famosi esolang: Brainfuck.

L'obiettivo del suo creatore era quello di sviluppare un linguaggio per un compilatore da meno di 200 byte. Infatti, questo linguaggio è noto proprio per il suo minimalismo: usa solo otto comandi e opera su un solo array di memoria. Nonostante tutto, però, si può ritenere Turing-Completo e quindi può eseguire qualsiasi tipo di programma computabile.

```
1 ++++++[>++++++>+++++++>++++>+<<<-]>+.,+.,+++++.,.+++.,>+.,<<+++++++>+.,
2 >.,+++.,
3 -----.,
4 -----.,
5 >+.,>+.,
6 <<+++++++>+.,>+.,.-----.,-----.,>+.,>+++++++>+.,
7
```

In questa immagine è stata scritta la frase ‘Ciao a tutti quanti!’ in Brainfuck. Può essere scritto anche in un'unica linea, ma in questo caso, con questa formattazione, ad ogni riga è presente una parola. Il codice non è programmato per stampare parole o frasi, piuttosto si occupa di stampare i numeri che rappresentano i caratteri secondo la tabella ASCII, e sta a noi programmare il computer per stampare un carattere alla volta basandoci sul codice corrispondente.

Sempre in questo decennio vengono introdotti anche Befunge, un linguaggio bidimensionale che permette ad un cursore di muoversi in quattro direzioni nello spazio di lavoro, e Piet, che utilizza immagini bitmap come sorgente: i programmi vengono ‘disegnati’ come opere astratte e il controllo è dato dal cambiamento nei colori.

Befunge-93

- ▶ Header
- ▼ Code

```
"!dlroW ,olleH">: #, _@
```
- ▶ Footer
- ▶ Input
- ▶ Arguments
- ▼ Output

```
Hello, World!
```
- ▶ Debug

Piet

- ▶ Header
- ▼ Code

```
0000000: 47 49 46 38 37 61 1c 00 03 00 84 14 00 ff 00 00 GIF87a.....
0000010: 00 00 00 ff c0 c0 ff 00 00 c0 00 00 ff ff c0 ff .....
0000020: ff 00 c0 c0 00 c0 ff c0 00 ff 00 00 c0 00 c0 ff .....
0000030: ff 00 ff ff 00 c0 c0 c0 ff 00 00 ff 00 00 c0 .....
0000040: ff c0 ff ff 00 ff c0 00 c0 ff ff ff ff ff ff ff .....
0000050: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff .....
0000060: ff ff ff ff ff ff ff ff ff ff ff ff ff ff 2c 00 00 .....
0000070: 00 00 1c 00 03 00 00 05 3c e0 23 42 4a 73 24 c3 .....<.#BJs$.
0000080: 23 39 ec 33 35 f0 02 39 d1 c2 3c 81 11 44 82 63 #9.35..9..<..D.c
0000090: 2a 40 05 42 70 98 10 24 13 c1 44 f4 80 2c 17 0a *@.Bp..$.D....
00000a0: 43 21 32 a9 56 0f 0d 87 80 3a 39 20 1c 12 c3 20 C!2.V....:9 ...
00000b0: 5c 28 17 72 21 00 3b \(.r!.;
```
- ▶ Footer
- ▶ Input
- ▶ Arguments
- ▼ Output

```
Hello, World!
```
- ▶ Debug



‘Hello, World!’ in Piet

Turing completezza

La Turing completezza è un criterio importante per valutare la potenza di un linguaggio. Deriva il suo nome dalla macchina di Turing, un concetto astratto proposto dal matematico Turing nei primi del Novecento.

Se un linguaggio può eseguire ogni algoritmo computazionale è Turing completo e, anche nel caso degli esolang, è importante sottolineare come essi lo siano, nonostante la loro natura e la

loro sintassi stravagante, e quindi, con la giusta disponibilità di memoria e di spazio, possono fare tutto ciò che fa un linguaggio più convenzionale.

Altri esempi di esolang

Oltre i già citati linguaggi, ne esistono molti altri, che sono stati sviluppati per le più disparate motivazioni.

Malbolge: è un linguaggio appositamente creato per essere difficile da programmare, molto criptico e complesso nella sintassi. Siccome il suo utilizzo è quasi impossibile, non vi è alcuna garanzia che un codice scritto in Malbolge funzioni in ogni compilatore.

Qui sotto metterò un esempio di codice che dovrebbe stampare la frase 'Ciao a tutti!' e si può subito notare la sua complessità.

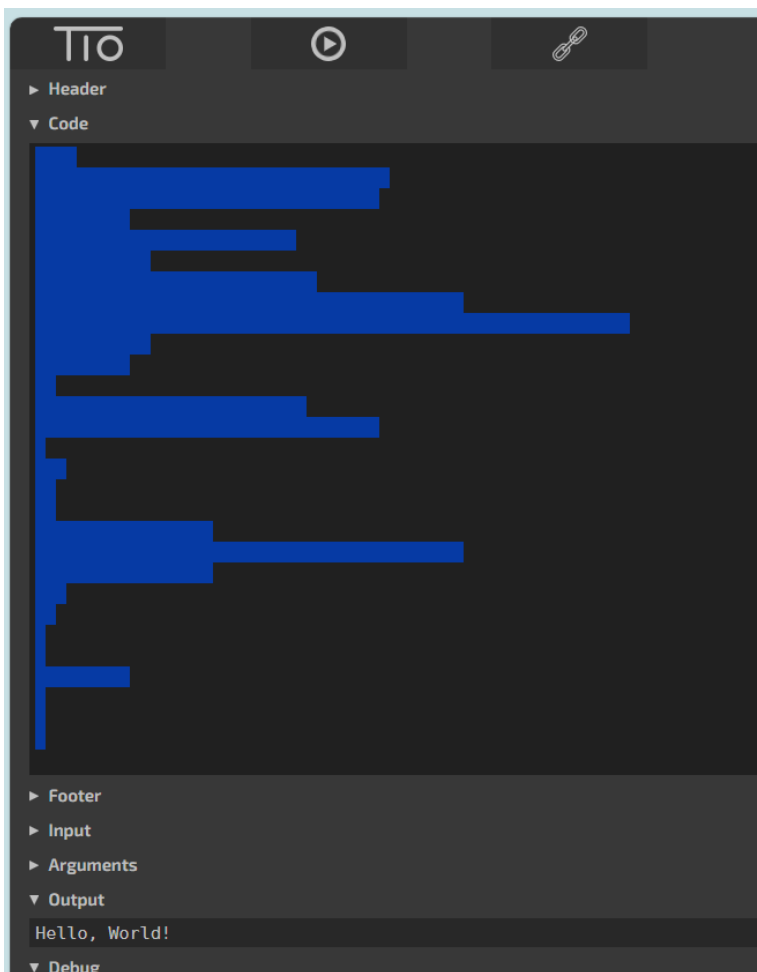
```
1 - ('&:9]!~}|z2Vxwv-,POqponl$Hjig%eB@@>}=<M:9wv6WsU2T|nm-,jcL(I&$$#"
2 - `CB]V?Tx<uVtT`Rpo3NlF.Jh++FdbCBA@?]|~|4XzyTT43Qsqq(Lnmkj"Fhg${z@>|
```

COW: è un linguaggio basato su comandi dati dalle variazioni della parola 'moo'. Si basa sulla struttura chiamata "memory tape", come per il Brainfuck, che consiste in una serie di 'celle' che contengono valori numerici. I comandi sono dati dalle variazioni delle capitalizzazioni di 'moo' e sono presenti 4 comandi eccezionali: MMM, OOO, oom, OOM.

```
1  MoO moO MoO mOo MOO OOM MMM moO moO
2  MMM mOo mOo moO MMM mOo MMM moO moO
3  MOO MOo mOo MoO moO moo mOo mOo moo
```

Questo nell'immagine è un esempio di codice COW che stampa la famosa sequenza di Fibonacci e si può vedere come ogni 'moo' viene modificato con maiuscole e minuscole.

Whitespace: questo esolang utilizza solo spazi, tabulazioni e avanzamenti di riga come comandi, non utilizzando quindi tutti gli altri caratteri, che, se inseriti, vengono considerati commenti.



Questo nell'immagine è un esempio di codice Whitespace che stampa la frase 'Hello, World!'. Ho evidenziato il codice per metterlo in risalto, ma si possono utilizzare anche delle lettere per commentare e quindi far capire dove finisce la linea.

Chef: è un linguaggio progettato per far sembrare i programmi ricette di cucina. Il principio fondamentale resta sempre quello di far generare alle ricette un output valido. Famoso è il codice scritto dal creatore del linguaggio stesso, ovvero il classico programma ‘Hello, World!’, e che ad un primo sguardo sembra una ricetta per un soufflé.

Hello World Souffle.

This recipe prints the immortal words "Hello world!", in a basically brute force way. It also makes a lot of food for one person.

Ingredients.

72 g haricot beans
101 eggs
108 g lard
111 cups oil
32 zucchinis
119 ml water
114 g red salmon
100 g dijon mustard
33 potatoes

Method.

Put potatoes into the mixing bowl. Put dijon mustard into the mixing bowl. Put lard into the mixing bowl. Put red salmon into the mixing bowl. Put oil into the mixing bowl. Put water into the mixing bowl. Put zucchinis into the mixing bowl. Put oil into the mixing bowl. Put lard into the mixing bowl. Put lard into the mixing bowl. Put eggs into the mixing bowl. Put haricot beans into the mixing bowl. Liquefy contents of the mixing bowl. Pour contents of the mixing bowl into the baking dish.

Serves 1.

Shakespeare: è un esolang che fa assomigliare i programmi alle opere letterarie di Shakespeare. Usa i personaggi delle opere come variabili e i dialoghi e le azioni svolte da loro come comandi.

```
Hamlet, a program.  
Juliet, a variable.  
  
[Hamlet]:  
You are as lovely as a beautiful rose!  
Speak your mind!  
  
[Juliet]:  
Listen to your heart!
```

FALSE: è un linguaggio minimalista e stack based, quindi segue il principio LIFO. Usa una sintassi molto semplice e breve, in quanto venne creato con l'obiettivo di essere usato su un compilatore di 1024 byte.

È importante sottolineare che fu preso ad esempio per la creazione di Befunge e Brainfuck.

```
1 [1 2 3 4 5] {+} % 5, $;
```

Questo è un piccolo programma che va a calcolare e stampa la somma dei primi cinque numeri positivi e interi.

LOLCODE: è stato ideato quando diventarono virali i memes di LOLCATS nel 2007.



```
HAI 1.3  
VISIBLE "Hai world!"  
KTHXBYE
```

Questi memes presentano gatti in posa con didascalie divertenti, scritte in inglese molto semplificato e a volte alterato, quasi a farlo sembrare come se scritto da un bambino.

In realtà negli anni questo linguaggio è stato criticato poiché non presenta le stesse caratteristiche degli altri esolang ma è un linguaggio semplice che presenta una

sintassi 'strana'.

Per questo, per alcuni, viene classificato come 'Weirdlang'.

Monicelli: è un linguaggio esoterico italiano che prende il nome dal famoso regista Mario Monicelli, famoso per i film divertenti e sarcastici, e in particolare è stato ispirato dal film 'Amici miei' e dalle 'supercazze' presenti nello stesso. È stato inventato, quindi, più per divertimento che per un uso pratico vero e proprio. Caratterizzato da una sintassi molto bizzarra e umoristica, con chiari riferimenti alla cultura e alla commedia italiana.

Le sue regole sono volutamente illogiche se messe a paragone con quelle di altri tipi di linguaggi di programmazione.

‘Lei ha clacsonato’ è l’apertura della funzione main, le operazioni si esprimono con le parole al posto dei simboli e dei confronti. I vari personaggi del film sono le variabili (Necchi – int, Perozzi – float, Sassaroli – double), l’input viene scritto ‘porge’ mentre l’output con ‘posterdati’. Infine, il programma viene concluso con il comando ‘vaffazum!’.

Questi sotto nell’esempio sono due script in Monicelli: a destra è presente il programma ‘Hello, World!’, mentre a sinistra il calcolo della successione di Fibonacci.

```
# ciao-mondo.mc
#
# Author: Cristian Consonni <cristian@balist.es>
# Released under MIT
#
# This program prints the string 'Ciao, mondo!\n' - which is Italian for
# 'Hello, world!\n'.
#
# The program declares a `char` variable called `tarapiatapioco` and then
# puts in the values need for each letter and prints them right away.
# These are the char values:
#
# C i a o , . m o n d o ! \n
# 67 105 97 111 44 32 109 111 110 100 111 33 10

Lei ha clacsonato
voglio la tarapiatapioco, Mascetti
tarapiatapioco come se fosse 67, tarapiatapioco a posterdati # C
tarapiatapioco come se fosse 105, tarapiatapioco a posterdati # i
tarapiatapioco come se fosse 97, tarapiatapioco a posterdati # a
tarapiatapioco come se fosse 111, tarapiatapioco a posterdati # o
tarapiatapioco come se fosse 44, tarapiatapioco a posterdati # ,
tarapiatapioco come se fosse 32, tarapiatapioco a posterdati #
tarapiatapioco come se fosse 109, tarapiatapioco a posterdati # m
tarapiatapioco come se fosse 111, tarapiatapioco a posterdati # o
tarapiatapioco come se fosse 110, tarapiatapioco a posterdati # n
tarapiatapioco come se fosse 100, tarapiatapioco a posterdati # d
tarapiatapioco come se fosse 111, tarapiatapioco a posterdati # o
tarapiatapioco come se fosse 33, tarapiatapioco a posterdati # !
tarapiatapioco come se fosse 10, tarapiatapioco a posterdati # \n
vaffanzum!
```

1. blinda la supercazzola Necchi velocizzata con la cosa Necchi o scherziamo?
2. che cos'è la cosa? bituma, mi ascolti, presti attenzione che il Perozzi è
3. minore di 2: vaffanzum una cosa!
4. o tarapia tapioco: voglio l'amore, Necchi
5. l'amore come se fosse brematurata la supercazzola velocizzata con una cosa meno 1 o scherziamo? più brematurata la supercazzola velocizzata con una cosa meno 2 o scherziamo?
6. che cos'è l'amore? bituma, che domande, vale nulla, non si perde tempo!
7. minore di 0: avvertite don ulrico e velocità di esecuzione
8. vaffanzum l'amore! bituma lasciamo perdere
9. e velocità di esecuzione
10. Lei ha clacsonato bituma, quanto chiasso per nulla
11. voglio un numero, Necchi mi porga il numero bituma, si sbrighi
12. brematurata la supercazzola velocizzata con un numero o scherziamo? a posterdati.

Correlazione tra Steganografia e Esolang - Conclusioni

In conclusione, la steganografia e i linguaggi di programmazione esoterici, pur avendo obiettivi e applicazioni apparentemente distinti, presentano numerose correlazioni significative.

Entrambi, infatti, condividono l'idea fondamentale di nascondere e offuscare l'informazione, presentando sfide uniche che richiedono creatività e ingegno per essere risolte.

La steganografia, come discusso, è l'arte di nascondere informazioni all'interno di altri dati. Questo è evidente negli esempi in cui messaggi sono nascosti in immagini o file audio, proteggendo così la privacy e la sicurezza delle comunicazioni. Analogamente, i linguaggi di programmazione esoterici, come Brainfuck e Malbolge, offuscano intenzionalmente la logica del codice, rendendolo complesso da comprendere per chi non è esperto. Questa caratteristica li rende strumenti potenti per aumentare la sicurezza del codice contro tentativi di reverse engineering.

La creatività e la sfida intellettuale sono altri elementi comuni. La steganografia richiede soluzioni innovative per nascondere efficacemente le informazioni e superare le tecniche di rilevamento. Allo stesso modo, i linguaggi esoterici rappresentano una sfida per i programmatori, offrendo nuovi paradigmi e complessità nella scrittura del codice.

Entrambi i campi trovano applicazione nella comunicazione nascosta e nel testare i limiti delle tecniche esistenti. La steganografia permette comunicazioni segrete, mentre i linguaggi esoterici possono essere utilizzati per codificare messaggi in modi non immediatamente riconoscibili.

Infine, abbiamo visto come la flessibilità e l'adattabilità siano caratteristiche chiave di entrambe le discipline. Le tecniche steganografiche possono essere applicate a vari formati di dati, mentre i linguaggi esoterici, pur complessi, offrono un'ampia gamma di applicazioni creative.

In sintesi, la steganografia e i linguaggi di programmazione esoterici non solo migliorano la sicurezza e la privacy, ma stimolano anche l'innovazione e la creatività nel campo della programmazione e della teoria della computazione. Questi strumenti ci ricordano l'importanza di guardare oltre l'apparente complessità per scoprire nuove possibilità e soluzioni nascoste.