MINISTERO DELLO SVILUPPO ECONOMICO
DIREZIONE GENERALE PER LA LOTTA ALLA CONTRAFFAZIONE
UFFICIO ITALIANO BREVETTI E MARCHI

UIBM

DOMANDA DI INVENZIONE NUMERO	**102021000012488**
Data Deposito	**14/05/2021**
Data Pubblicazione	**14/11/2022**

Classifiche IPC

Sezione	Classe	Sottoclasse	Gruppo	Sottogruppo
G	06	F	21	12

Sezione	Classe	Sottoclasse	Gruppo	Sottogruppo
G	06	F	21	14

Sezione	Classe	Sottoclasse	Gruppo	Sottogruppo
G	06	N	3	04

Sezione	Classe	Sottoclasse	Gruppo	Sottogruppo
G	06	N	3	08

Titolo

Metodo di configurazione di reti neurali e metodo di elaborazione di file binari

"Metodo di configurazione di reti neurali e metodo di elaborazione di file binari"

DESCRIZIONE

CAMPO DELLA TECNICA

La presente invenzione riguarda il campo della sicurezza del software e le protezioni ad esso applicate.

STATO DELLA TECNICA

Il software, a causa delle sue caratteristiche intrinseche, è altamente sensibile sotto l'aspetto della sicurezza.

Si pensi ad esempio che il software sovente incorpora o gestisce dati confidenziali, privati o proprietà intellettuale di terze parti. Esso rende inoltre esplicito il know-how che ha reso possibile la sua creazione. Tali informazioni sono disponibili sotto forma di istruzioni e strutture dati che compongono un dato software o che vengono da questo trattate. Sono quindi potenzialmente disponibile a chiunque abbia una copia del software. Pertanto, il software può anche essere visto come un contenitore di assets fondamentali per il business dell'azienda che l'ha sviluppato.

Il software è esposto a numerose minacce e rischi. In uno scenario di tipo MATE (Man-At-The-End), un attaccante, avendo a disposizione un determinato software e controllando l'ambiente in cui esso verrà eseguito, può analizzare il comportamento di detto software, ad esempio mediante un debugger, o può disassemblarlo o decompilarlo per estrarne la struttura logica. Queste operazioni di *reverse engineering* possono permettere quindi di ottenere alcuni degli asset presenti nel software. Gli attacchi MATE sono frequenti e possono consistere nell'individuazione ed eventuale riutilizzo di funzionalità ritenute strategiche, la compromissione dei controlli di licenza, l'identificazione di falle o contesti di utilizzo che possono essere sfruttati per compromettere le funzionalità del software stesso o degli

ambienti in cui esso viene eseguito, etc.

Data la severità dei rischi cui il software è esposto, non si può prescindere dall'adottare opportune mitigazioni. In questo contesto le mitigazioni sono i metodi e tecnologie di protezione del software, adottati sia in fase di sviluppo del software che immediatamente prima della sua distribuzione. Il processo di protezione del software combina l'uso di funzioni crittografiche, trasformazioni del software stesso e tecniche di ingegneria del software al fine di mitigare i rischi. Le protezioni del software possono fare affidamento su funzionalità di sicurezza disponibili nell'ambiente in cui il software è eseguito, ma possono anche essere integrate nel software stesso, utilizzando tecnologie di protezione appositamente progettate.

Pur non esistendo soluzioni definitive che possano impedire ad attaccanti di ottenere gli asset nel software, sono note nell'arte numerose soluzioni di protezione adottate, volte a ritardare il più possibile l'attaccante e quindi preservare il business model adottato dal proprietario di tale software.

Si pensi ad esempio allo scenario di una software house - le software house rappresentano una parte importante del mercato delle protezioni del software - nel momento in cui intende commercializzare un nuovo videogame. Una larga fetta delle vendite di questo tipo di prodotto si concentra nei primi giorni successivi all'uscita sul mercato. Pertanto, diventa fondamentale ritardare il più possibile il momento un attaccante riesca a violare con successo le protezioni software contenute nell'opera, ad esempio i controlli di licenza, rilasciando *crack* o facendo circolare copie illegali del videogame.

L'efficacia delle tecniche di protezione applicate al software sono sovente valutate, prima che un dato software venga rilasciato, da personale specializzato interno o esterno alla azienda che sviluppa il software. Tuttavia, questo processo di valutazione è spesso eseguito in modo manuale o per mezzo di strumenti semi-automatizzati, richiedendo pertanto il

dispendio di una notevole quantità di tempo e risorse. Inoltre, il tempo molto spesso è limitato a causa dei modelli di business adottati che impongono un time-to-market stringente per riuscire ad imporsi sul mercato prima dei concorrenti.

Una prima fase che attaccanti prima devono svolgere di estrarre asset dal software consiste nell'identificare le tecniche di protezione che sono state applicate a specifiche porzioni del software, al fine di disabilitarle o eliminarle. Una valutazione dell'efficacia delle protezioni non può prescindere dallo stimare quanto facile è riconoscere quali protezioni sono state applicate.

SOMMARIO DELL'INVENZIONE

Attualmente gli strumenti noti nello stato dell'arte volti ad individuare le tecniche di protezione applicate ad un software non sono ancora soddisfacenti poiché offrono, quando presenti, un limitato livello di automazione e di conseguenza dipendono fortemente dall'attività manuale, in un contesto in cui il tempo gioca un ruolo fondamentale per il successo o il fallimento di un prodotto software sul mercato.

Inoltre, detti strumenti non sono ottimizzati per lo svolgimento di attività di individuazione delle aree protette all'interno del software stesso.

Lo scopo della presente invenzione è quello di migliorare il grado di automazione nella fase di individuazione delle tecniche di protezione applicate al software.

È oggetto della presente invenzione un metodo per l'automazione del processo di individuazione delle protezioni software applicate ad un file binario e dell'individuazione delle aree protette all'interno di detto file come descritto dalla rivendicazione 1 e da sue forme di realizzazione preferite descritte dalle rivendicazioni 2-12.

È inoltre oggetto della presente invenzione un metodo di elaborazione di file come descritto dalla rivendicazione 13 e da sue forme di realizzazione preferite descritte dalle

rivendicazioni 14-16.

BREVE DESCRIZIONE DEI DISEGNI

Forme di realizzazione preferite della presente invenzione verranno descritte nel seguito, a puro titolo esemplificativo, con riferimento ai disegni allegati, in cui:

- La Fig.1 mostra un esempio di un sistema informatico utilizzabile ai fini della presente invenzione;
- La Fig.2 mostra un esempio schematico di un file binario in cui sono evidenziate le funzioni e l'eventuale presenza di protezioni;
- La Fig.3 mostra, mediante blocchi funzionali, un esempio di un metodo di configurazione di una rete neurale in grado di ottenere informazioni circa le tecniche di protezione eventualmente presenti in un file da analizzare;
- La Fig.4 mostra un esempio di funzione codificata;
- La Fig.5 mostra, mediante blocchi funzionali, un'architettura semplificata di una rete neurale basata su celle LSTM (Long Short Term Memory);
- La Fig.6 mostra, mediante blocchi funzionali, un'architettura semplificata di una rete neurale di tipo transformer BERT (Bidirectional Encoder Representations from Transformers).

DESCRIZIONE DETTAGLIATA DELL'INVENZIONE

La seguente descrizione dettagliata delle forme di realizzazione preferite si riferisce ai disegni allegati che ne costituiscono una parte e mostrano, a titolo esemplificativo, specifiche forme di realizzazione della presente invenzione. La seguente descrizione non è pertanto da intendersi in senso limitativo, e la portata delle invenzioni è definita solo dalle rivendicazioni allegate.

La Fig. 1 mostra un esempio di un sistema informatico 10 configurato per fornire informazioni circa eventuali protezioni del software contenute in un file da analizzare.

Il sistema 10 comprende, per esempio, un dispositivo di elaborazione general-purpose 20, sotto forma di un convenzionale personal computer, che include un'unità di processamento 21, una memoria di sistema 22 ed un bus di sistema 23 che accoppia la memoria di sistema 22 e altri componenti di sistema all'unità di processamento 21.

Il bus di sistema 23 può essere uno qualsiasi dei differenti tipi di bus in grado di consentire un canale di comunicazione tra differenti dispositivi hardware per lo scambio di informazioni. La memoria di sistema 22 comprende, per esempio, una memoria di sola lettura (ROM) 24 e una memoria ad accesso casuale (RAM) 25. Un sistema di input/output di base (BIOS) 26, memorizzato nella ROM 24, contiene le routine di base che trasferiscono le informazioni tra i componenti del personal computer 20. Il BIOS 24 contiene inoltre le routine di avvio del sistema. Il personal computer 20 comprende inoltre un drive per l'hard disk 27 per leggere da e scrivere su almeno un hard disk 29. Il drive per l'hard disk 27 è collegato al bus di sistema 23 tramite un'interfaccia per unità hard disk 32.

Per esempio, il sistema 10 include l'hard disk 29 ma potrebbe comprendere altri tipi di supporti, quali schede di memoria, hard disk esterni, RAM, ROM e simili.

I moduli di programma possono essere memorizzati su hard disk 29 sulla ROM 24 e sulla RAM 25.

Un utente può inserire comandi e informazioni nel personal computer 20 attraverso uno o più dispositivi di input come, per esempio, una tastiera 40 e un dispositivo di puntamento ottico 42. Questi e altri dispositivi di input sono spesso collegati all'unità di processamento 21 attraverso una specifica interfaccia di input 46 che dipende dal tipo di porta utilizzata come ad esempio una porta seriale, una porta parallela, una porta USB, etc. accoppiata al bus di sistema 23. Un monitor 47 o un altro dispositivo di visualizzazione si collega anche al bus di sistema 23 tramite un'interfaccia come un adattatore video 48. Oltre

al monitor, i personal computer possono comprendere inoltre altre periferiche di output (non mostrate) come ad esempio una stampante.

Il personal computer 20 può operare, in una rete di scambio di dati, utilizzando connessioni logiche a uno o più computer remoti come il computer remoto 49. Il computer remoto 49 può essere un altro personal computer, un server, un router, un PC di rete o un altro nodo della rete. Comprende tipicamente molti o tutti i componenti descritti sopra in relazione al personal computer 20. Tuttavia, nell'esempio in Fig. 1 è mostrato per semplicità solo un dispositivo di archiviazione 50. Le connessioni logiche mostrate in Fig. 1 possono comprendere una rete di tipo LAN e/o WAN 51 comuni negli uffici, nelle reti di computer aziendali, nelle intranet e in Internet.

Quando si trova in un ambiente di rete LAN/WAN, il PC 20 si collega ad una rete 51 attraverso un'interfaccia di rete o un adattatore 53 che può essere una scheda di rete cablata o wireless. In un ambiente di rete, i moduli di programma rappresentati come residenti all'interno del personal computer 20 o porzioni di essi possono essere memorizzati in un dispositivo di archiviazione remoto 50.

I moduli di programma possono comprendere: il sistema operativo 35, uno più programmi applicativi 36, almeno una rete neurale NN(P_i) (modulo di processamento 33) e un modulo di addestramento MOD_TRAIN 34. In particolare, è prevista una pluralità di reti neurali NN(Pi), ciascuna associata ad una particolare protezione del software.

Ognuna delle reti neurali NN(P_i) può essere implementata in hardware, in software o in una loro combinazione. Il modulo di addestramento MOD_TRAIN 34 ha il compito di addestrare ogni rete neurale NN(P_i) per mezzo di data set, ovvero una collezione di dati utilizzati come campioni allo scopo di "insegnare" alla rete neurale NN(P_i) come reagire a fronte di specifici dati in ingresso.

Come verrà descritto in modo più dettagliato più avanti, ciascuna rete neurale

$NN(P_i)$ è addestrata per ottenere informazioni circa una specifica tecnica di protezione eventualmente presente in un file da analizzare.

La figura 2 mostra un esempio schematizzato di file binario che può appartenere, a titolo esemplificativo, ad un'applicazione o ad una libreria software. Il file binario è formato da una pluralità di funzioni FNZ 1 – FNZ n, ognuna delle quali è composta da una sequenza di istruzioni assembly, dette anche righe di codice o più in generale codice. Una o più di detta pluralità di funzioni contenute nel file binario potrebbero necessitare di protezione software qualora il loro contenuto rappresenti un asset, ovvero costituisca un valore in termini economici e/o di know-how.

Gli assets che possono costituire un'area critica all'interno del file binario possono essere ad esempio, ma non solo, gli algoritmi proprietari (o altra proprietà intellettuale), segreti crittografici oppure i controlli di sicurezza come ad esempio i controlli di licenza di un software commerciale.

In figura 2, la funzione FNZ 1 e la funzione FNZ 6 sono mostrate con il simbolo di uno scudo a voler specificare che queste due funzioni sono il risultato dell'applicazione di protezioni del software poiché contenenti almeno un asset. Si noti che particolari righe di codice o funzioni potrebbero essere state protette al fine di confondere gli attaccanti pur non essendo degli asset.

Al contrario, la funzione FNZ 4 è invece mostrata con un simbolo di una X a voler indicare che non si tratta di una funzione protetta con alcun tipo di protezione software.

Le protezioni software, dopo essere state applicate al codice lasciano una loro impronta digitale, o fingerprint, anomala rispetto al codice non protetto.

Esempi di impronta digitale presente nel codice a seguito dell'applicazione di una protezione software potrebbero essere flussi di controllo o condizioni logiche

particolarmente complessi.

Ogni protezione software possiede un'impronta digitale caratteristica che potrebbe permettere di dedurre alcune informazioni come ad esempio quali peculiarità possiedono gli asset protetti e quali proprietà di sicurezza si è deciso di applicare. Esempi di protezioni software, possono essere: il control flow flattening, i predicati opachi, le branch functions, l'encode arithmetic, la conversione di dati in funzioni (ad esempio con macchine di Mealy), la fusione o suddivisione di variabili, la ricodifica di variabili (ad esempio, l'xor masking, il residue number encoding, ...), la white-box cryptography, la virtualizzazione mediante l'uso di virtual machine o di compilazione JIT, i controlli sul call stack, le code guards, il control flow tagging, l'anti-debugging, la code mobility, il client/server code splitting, l'anti-cloning e la software attestation.

La figura 3 mostra, mediante un diagramma di flusso, una forma di realizzazione preferita di un metodo di configurazione di reti neurali 100 implementabile, per esempio, mediante il sistema 10.

Il metodo 100 consente di configurare una o più reti neurali NN(Pi), ciascuna da impiegare per ottenere informazioni circa una specifica tecnica di protezione eventualmente presente in un file da analizzare.

Dopo una fase di inizio, il metodo 100 prevede una prima fase 110 in cui sono forniti uno o più file sorgenti (cioè un file espresso con un linguaggio ad alto livello) impiegati ai fini di un addestramento di una rete neurale. Si consideri, per brevità, l'impiego di un singolo file sorgente. Secondo l'esempio descritto, tale file sorgente è, inizialmente, privo di protezioni software.

Successivamente, secondo un esempio, a tale file sorgente vengono applicate una o più protezioni software. Nell'esempio considerato ci si riferisce al caso in cui sono applicate più protezioni software $P_1,\ldots,P_n$. In particolare, tali protezioni $P_1,\ldots,P_n$ sono

applicate ad una o più funzioni del file sorgente.

Come è anche ribadito più avanti, non a tutte le funzioni del file sorgente si applicano le protezioni, ma alcune di esse rimangono prive di protezione. Ciò permetterà di addestrare le reti neurale NN(Pi) anche al riconoscimento di funzioni non protette.

Secondo l'esempio qui descritto, il file sorgente provvisto delle protezioni software $P_1,\ldots,P_n$ viene quindi compilato, ottenendo, un file binario.

Si noti che è possibile prevedere che sia direttamente fornito un file binario al quale sono state già applicate le protezioni $P_1,\ldots,P_n$, evitando cioè di effettuare la prima fase 110. Si osservi che alcune protezioni sono applicate al file sorgente mentre altre sono applicate direttamente sul file binario.

In una seconda fase 120 si effettua il disassemblaggio del file binario ottenibile, per esempio, dalla precedente compilazione allo scopo di estrarne la pluralità di funzioni in esso contenute (cioè, delle sue porzioni di codice).

L'operazione di disassemblaggio permette di ottenere il file precedentemente compilato sotto forma di codice assembly andando a sostituire ogni codice operativo del linguaggio macchina con una sequenza di caratteri che lo rappresenta in forma mnemonica, cioè in un modo facilmente interpretabile da un operatore. Anche i dati e gli indirizzi di memoria possono essere riscritti in assembly secondo una base numerica, ad esempio esadecimale, oppure in forma simbolica utilizzando stringhe di testo (identificatori). Il programma in formato assembly risulterà quindi relativamente più leggibile rispetto al binario corrispondente.

Esempi di codici operativi in formato mnemonico sono la ADD per l'operazione di somma o la MOV ad indicare un'operazione di copia.

Inoltre, è effettuata una terza fase 130 che ha lo scopo di raccogliere in un data set, o collezione di dati, la pluralità di funzioni protette estratte dal file assembly

contestualmente all'indicazione delle protezioni presenti per ogni funzione. Tale indicazione è un identificativo del tipo di protezione applicata $P_1,\ldots,P_n$. Il data set può avere una struttura matriciale.

Si noti che il data set può essere anche ottenuto mediante una libreria di funzioni protette e associate ad una libreria di protezioni, senza effettuare l'applicazione delle protezioni alle funzioni del file binario e l'estrazione delle funzioni dal file binario indicata nella seconda fase 120.

L'insieme delle informazioni riguardanti la funzione protetta e l'indicazione delle protezioni applicate alla medesima funzione definisce un campione CHMP. Tale campione CHMP è, per esempio, una riga del data set nel seguente formato:

$$(asm_j, P_1,\ldots,P_n)$$

dove asm_j è la j-esima funzione protetta estratta dal file assembly e gli identificativi $P_1,\ldots,P_n$ rappresentano le specifiche protezioni software applicate alla funzione asm_j.

In particolare gli identificativi $P_1,\ldots,P_n$ possono essere delle variabili booleane che indicano se la protezione P_i è stata applicata alla funzione asm_j. Per funzioni alle quali non è stata applicata alcuna protezione gli identificativi $P_1,\ldots,P_n$ assumono il valore "falso".

I campioni CHMP saranno impiegati nelle successive fasi del metodo 100. Si noti che per effettuare una buona fase di addestramento di una rete neurale è necessario che il data set contenga un numero sufficientemente elevato di campioni CHMP.

Secondo una forma preferita del metodo 100, la prima fase 110 (fase di compilazione) e la seconda fase 120 (fase di disassemblaggio) possono essere eseguite più volte utilizzando differenti combinazioni di protezioni che prevedano l'applicazione di una o più sequenze di protezioni a ciascuna funzione.

Come già accennato, vantaggiosamente, il data set è costruito anche utilizzando funzioni compilate alle quali non è stata applicata alcuna protezione software. Le funzioni

compilate senza alcuna protezione software vengono chiamate funzioni vanilla ed hanno lo scopo di bilanciare il data set. Lo scopo di tale bilanciamento è quello di evitare che un data set sbilanciato influisca negativamente sul processo di apprendimento di una rete neurale, descritto nel seguito del presente documento, portandola a focalizzarsi sugli eventi prevalenti, trascurando quelli rari. In particolare, le funzioni vanilla servono a far apprendere alla rete neurale come sono fatte funzioni non protette, in modo che possa distinguere queste dalle funzioni protette con la specifica tecnica di protezione che la rete neurale è addestrata ad identificare.

In una quarta fase 140 si esegue la codifica di ogni funzione (asm_i) appartenente alla pluralità di funzioni convertendole in funzioni codificate CHMP_COD. Questa operazione ha lo scopo di trasformare in una sequenza di valori numerici le istruzioni contenute nelle funzioni di ogni campione CHMP espresse in linguaggio assembly, in particolare, le istruzioni contenenti codici operativi, dati ed indirizzi quando espressi in formato mnemonico.

Al termine di detta fase di codifica 140 ogni funzione codificata CHMP_COD appartenente al data set sarà espressa come sequenza di valori numerici risultando pertanto adeguata ad essere utilizzata in una fase di addestramento di una rete neurale. La fase di codifica 140 può opzionalmente comprendere due ulteriori sotto-fasi che permettono ad una rete neurale di raggiungere più velocemente la convergenza: la sotto fase di mascheratura e la sotto fase di scalamento.

In una quinta fase 150, si esegue un addestramento di una delle reti neurali NN(Pi). Per esempio, è addestrata una prima rete neurale $NN(P_1)$ associata ad una prima protezione del software P_1.

In particolare, la prima rete neurale $NN(P_1)$ è addestrata per fornire un primo indice di probabilità PIi (in questo caso i = 1) che indica la probabilità che una data

funzione j-esima (asm_j) sia protetta dalla prima protezione P_1.

Inoltre, nel caso in cui si sia individuata una funzione alla quale risulta applicata la prima protezione P1 con un primo indice di probabilità PIi,j superiore ad una determinata soglia, la prima rete neurale NN(P_1) può fornire anche un secondo indice FAi,j,k. Tale secondo indice FAi,j,k rappresenta la possibilità che alle istruzioni di una specifica area (indicata genericamente con l'indice k) di tale funzione (asm_j) sia stata applicata la prima protezione P_1 (Pi con i= 1). Per esempio, dove è più alto il valore del secondo indice più l'area è "sospetta".

Ad esempio, per ogni istruzione di una funzione, la prima rete neurale NN(P_1) indica la probabilità che questa presenti la prima protezione P1. Ciò permette di identificare le istruzioni della funzione alle quali è stata applicata una determinata protezione o che in alternativa sono state introdotte dall'applicazione della protezione, o in altre parole, di individuare la posizione di una protezione all'interno di una funzione.

L'addestramento della prima rete neurale NN(P_1) è effettuato utilizzando il data set che comprende le funzioni codificate CHMP_COD relative alla prima protezione P_1 e le funzioni codificate CHMP_COD protette con ogni possibile coppia di protezioni Pi (per esempio, P1+P2, P1+P3, ... , P1+Pn). Si noti che anche è possibile utilizzare combinazioni più lunghe di quelle sopra indicate, per esempio triple (e.g. P1+P2+P3), quadruple (e.g. P1+P2+P3+P4), ecc. In tal modo si potrebbe migliorare l'accuratezza in casi in cui si vogliano identificare particolari combinazioni di protezioni più lunghe (triple, quadruple ecc.), ad esempio perché è noto che sono utilizzate nello stato dell'arte.

Secondo l'esempio descritto, l'addestramento è effettuato mediante il modulo di addestramento MOD_TRAIN 34.

L'addestramento può essere ripetuto per ogni rete neurale NN(Pi) relativa anche alle altre protezioni di interesse P_2-P_n.

Si osservi che la rete neurale NN(Pi) può essere scelta tra le reti neurali in grado di gestire sequenze e/o le reti neurali aventi un meccanismo di attenzione.

Una tipologia di rete neurale in grado di gestire le sequenze è, ad esempio, una rete neurale di tipo ricorrente, ovvero una rete in cui sono presenti connessioni di retroazione. Tale retroazione crea una sorta di "memoria" di quanto accaduto nel passato recente rendendo disponibile al tempo T un'informazione processata al tempo T-1 o T-2 facendo pertanto dipendere il valore dell'uscita corrente non solo dai valori di ingresso corrente, ma anche dagli ingressi precedenti. Un esempio di rete neurale ricorrente è la rete LSTM (Long Short-Term Memory).

L'idea alla base del meccanismo di attenzione è quella di poter definire su quali parti del vettore di input la rete neurale deve concentrarsi per generare l'output appropriato. In altri termini, un meccanismo di attenzione consente di elaborare dei dati in ingresso mentre si occupa anche delle informazioni rilevanti contenute in altri dati di ingresso. Il meccanismo di attenzione consente inoltre di mascherare quei dati che non contengono informazioni rilevanti. Esempi di reti neurali che utilizzano il meccanismo di attenzione possono essere ad esempio le reti neurali ricorrenti, come ad esempio la già citata LSTM, oppure le reti neurali come ad esempio BERT (Bidirectional Encoder Representations from Transform).

Le reti neurali NN(Pi), addestrate come sopra descritto, possono essere impiegate in un metodo di classificazione applicato ad un file binario da analizzare (cioè un file distinto da quello usato per l'addestramento nel metodo di configurazione 100).

In questo caso, il file binario da analizzare viene disassemblato e dal file assembly così ottenuto si estraggono le relative funzioni (asm) che si vogliono analizzare. Ciò può essere ottenuto mediante un convenzionale disassemblatore.

Ciascuna funzione (asm) è quindi processata da ogni rete neurale NN(Pi).

Ciascuna di tali reti neurali NN(Pi) restituirà un relativo primo indice di probabilità PIi,j associato ad una specifica protezione Pi e anche, preferibilmente, il secondo indice di FAi,j,k per ogni funzione.

L'insieme dei valori del primo indice di probabilità PIi,j consente di effettuare una classificazione delle protezioni $P_1,\ldots, P_n$ eventualmente presenti nel file binario analizzato.

I valori dei secondi indici FAi,j,k sono associati ad ulteriori indicazioni che identificano la posizione delle istruzioni all'interno di ogni funzione aventi tali valori del secondo indice.

Quindi il metodo di classificazione permetterà di valutare la qualità della sicurezza delle protezioni applicate al file binario analizzate perché l'individuazione di tali protezioni da parte delle reti neurali NN(Pi) indica che la protezione è identificabile rapidamente, quindi un attaccante viene 'ritardato' di meno.

Con riferimento ad esempi di applicazioni pratiche, si osservi che le società specializzate nella protezione del software tipicamente operano utilizzando due team distinti: il primo (team di protezione) si occupa di proteggere effettivamente il software, mentre il secondo team (team di reverse engineering), emula il comportamento dei possibili attaccanti, cercando di identificare gli asset presenti all'interno dell'applicazione e le protezioni utilizzate, per poi rimuovere/aggirare queste ultime compromettendo la sicurezza degli asset. Il team di protezione propone una soluzione iniziale, la cui bontà è valutata dal team di reverse engineering. Queste operazioni sono quindi eseguite iterativamente finché non è stato raggiunto un livello di protezione sufficiente (oppure è terminato il tempo a disposizione).

Il metodo di classificazione descritto, basato sul metodo di configurazione 100, può essere quindi utilizzato dalle società specializzate nella protezione del software in due modalità differenti. Il team di protezione può ottenere una valutazione rapida della

identificabilità delle protezioni scelte (senza aspettare i risultati delle attività di reverse engineering). Contestualmente, il metodo di classificazione descritto può anche essere utilizzato dal team di reverse engineering per automatizzare e velocizzare l'identificazione degli asset, un primo passaggio imprescindibile nelle loro attività.

I metodi di configurazione e classificazione sopra descritti sono applicabili ad ogni tipo di protezione sopra elencata e, con particolare efficacia, alle seguenti tipologie di protezione, come emerso da test effettuati dalla Richiedente: control flow flattening, predicati opachi, branch functions, l'encode arithmetic.

Nel seguito sono descritte forme di attuazione particolari di alcune delle fasi del metodo 100 sopra descritto.

Codifica

La fase di codifica 140, come detto in precedenza, ha lo scopo di trasformare le istruzioni contenute nelle funzioni di ogni campione CHMP in sequenze di valori numerici. Questa trasformazione ha l'obiettivo di codificare ogni funzione (asm_i) dei campioni CHMP appartenenti al data set come una matrice di valori numerici le cui righe sono le codifiche delle istruzioni componenti la funzione stessa.

La funzione codificata, come mostrato nella figura 4a, presenta due pseudo-istruzioni fittizie (anch'esse codificate assieme al resto della funzione) di <begin> ed <end> aggiunte rispettivamente all'inizio ed alla fine della funzione (asm_i) per esplicitarne i confini. Tali pseudo-istruzioni fittizie sono inserite in una fase preliminare della fase di codifica 140.

Opzionalmente, sempre in detta fase preliminare, il numero di istruzioni che costituiscono la funzione può essere troncata alle prime n istruzioni. A solo scopo esemplificativo in figura 4c è mostrata la funzione codificata troncata ad una dimensione massima prefissata pari a 4 rispetto alla funzione in figura 4a. L'operazione di troncamento

si rende necessaria nel momento in cui il tipo di rete neurale scelta per essere addestrata ad operare come un classificatore prevede che la sequenza in ingresso abbia una dimensione massima che non va superata.

La figura 4b mostra invece il dettaglio di una i-esima istruzione appartenente alla funzione di figura 4a dopo essere stata codificata. Secondo l'esempio, l'istruzione codificata è composta da:

- un opcode rappresentante la codifica numerica dell'opcode dell'istruzione;
- un indirizzo rappresentante l'indirizzo dell'istruzione;
- una serie di parametri op1-op6 rappresentanti la codifica degli eventuali operandi dell'istruzione.

Il numero dei parametri può variare sulla base dell'architettura hardware scelta.

Nella figura 4b è rappresentata la generalizzazione di un'istruzione basata su architettura di tipo ARM dove le istruzioni più complesse possono avere fino a sei operandi. Tuttavia, il numero di operandi gestibile dalla fase di codifica 140 non ha un limite al numero massimo di operandi in modo da potersi agevolmente adattare alle istruzioni di differenti architetture hardware.

Viene di seguito descritto un esempio di codifica di un'istruzione effettuata dal presente metodo tenendo conto che i valori numerici riportati relativi alla codifica sono da intendersi a puro scopo descrittivo, dato che tali valori possono cambiare in base all'architettura hardware e alla rete neurale scelti.

Un esempio di istruzione assembly è "0x1234 add r0, r2, 5" che esprime l'assegnazione r0 = r2 + 5 e può essere suddivisa in cinque differenti parti:

- 0x1234: è il valore numerico che indica l'indirizzo dell'istruzione che nell'esempio considerato è un intero espresso in esadecimale, che equivale al numero 4660 in base decimale, ed indica la posizione dell'istruzione in memoria;

- add: è il tipo di operazione (opcode) dell'istruzione, che in questo caso è una somma;
- r0: è il primo operando dell'istruzione e fa riferimento al registro di memoria r0;
- r2: è il secondo operando dell'istruzione e fa riferimento al registro di memoria r2;
- 5: è il terzo operando dell'istruzione e indica il numero intero 5.

In questo esempio, l'istruzione utilizza solo tre operandi, il quarto, il quinto e il sesto operando saranno assenti.

Per esempio, la fase di codifica 140 trasforma ogni riga di istruzione delle funzioni di ogni campione CHMP in una sequenza di 1250 valori così suddivisi:

- 236 dedicati alla codifica dell'opcode;
- 4 dedicati alla codifica dell'indirizzo dell'istruzione;
- 273 dedicati alla codifica del primo operando;
- 239 dedicati alla codifica del secondo operando;
- 239 dedicati alla codifica del terzo operando;
- 239 dedicati alla codifica del quarto operando;
- 16 dedicati alla codifica del quinto operando;
- 4 dedicati alla codifica del sesto operando.

Si consideri che la codifica dell'istruzione deve essere sempre una sequenza lunga 1250 valori anche nel caso di un numero inferiore di operandi, nell'esempio pari a 6. Dato che l'istruzione di ADD considerata ha solo tre operandi, anche l'assenza degli operandi mancanti è codificata per non alterare la lunghezza della sequenza di codifica di valori.

La codifica dell'opcode avviene su una sequenza di 236 valori numerici e rappresenta l'embedding dell'opcode stesso. Per embedding si intende una tecnica standard di

modellazione dove le parole o i numeri vengono mappati in sequenze numeriche. Detti valori sono stati pre-calcolati per mezzo di una rete neurale utilizzante algoritmi standard, quali ad esempio gli algoritmi CBOW (Continuous Bag Of Words) e Skip-gram, descritti nel documento Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean, "*Distributed Representations of Words and Phrases and their Compositionality*", in Proceedings of the 26th International Conference on Neural Information Processing Systems, Volume 2 (NIPS'13), Curran Associates Inc., Red Hook, NY, USA, pp. 3111-3119, disponibile al seguente indirizzo: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

La codifica dell'indirizzo avviene su una sequenza di 4 valori numerici dove ognuno dei 4 valori è determinato secondo un preciso criterio.

Il primo valore della sequenza è valorizzato a 1 quando l'istruzione contiene un indirizzo, o a 0 quando mancante, come nel caso di alcune istruzioni speciali. Le istruzioni <begin> e <end> sono esempi di istruzioni speciali.

Il secondo valore della sequenza è valorizzato a 0 quando l'istruzione non ha un indirizzo oppure se il risultato dell'operazione matematica è stata eseguita su operandi non validi (NaN) oppure vale infinito. In tutti gli altri casi il secondo valore della sequenza è valorizzato a 1.

Il terzo valore della sequenza contiene il valore dell'indirizzo espresso in base decimale quando presente, altrimenti è valorizzato a 0.

Il quarto valore della sequenza è valorizzato a 1 quando l'indirizzo dell'istruzione contiene il punto esclamativo (!) come accade per alcuni indirizzi speciali, altrimenti è valorizzato a 0. Il punto esclamativo è usato in certi (rarissimi) casi in ARM per indicare l'operazione di write-back, ovvero che il risultato di un'operazione deve essere scritto dentro

un certo indirizzo. Per esempio, se in un'istruzione è scritto l'indirizzo "1234!" significa che il risultato dell'istruzione deve essere scritto nella cella di memoria 1234. Se non è presente il punto esclamativo, allora il risultato dell'operazione non viene scritto nella cella di memoria 1234 (che tipicamente verrà solo letta).

Secondo detti criteri, l'indirizzo 0x1234 contenuto nell'istruzione di esempio è codificato come 1, 1, 4660, 0.

La codifica del primo operando, come detto in precedenza, avviene su una sequenza di 273 valori che sono suddivisi in 9 sezioni per poter rappresentare i diversi tipi di operando.

Una prima sezione denominata "stringa" è utilizzata per codificare un operando di tipo stringa su un valore della sequenza. Una seconda sezione denominata "numero" è utilizzata per codificare un operando di tipo numerico su una sequenza di 4 valori. Una terza, quarta e quinta sequenza denominate "endian", "cond" e "stato CPU" sono utilizzate per codificare gli stati interni del processore rispettivamente su sequenze di 2, 15 e 4 valori. La sesta sezione denominata "registri" è utilizzata per codificare i registri di memoria su una sequenza di 154 valori. Una settima sezione denominata "barriera" è utilizzata per codificare le barriere di memoria su una sequenza di 12 valori. Una barriera di memoria è un tipo di operazione che permette alla CPU di imporre un vincolo sull'ordinamento delle operazioni evitando un'esecuzione fuori ordine a causa delle ottimizzazioni delle prestazioni delle CPU moderne.

Un'ottava sezione denominata "indirizzo" è utilizzata per codificare gli indirizzi di memoria su una sequenza di 65 valori. Infine, la nona sezione denominata "coproc" è utilizzata per codificare un coprocessore matematico su una sequenza di 16 valori.

Nell'istruzione d'esempio il primo operando è un registro, r0, e pertanto sarà codificato nella sesta sezione "registri" mentre le sequenze di valori delle altre sezioni saranno tutte valorizzare a 0.

Dei 154 valori di detta sezione, i primi 123 valori sono associati ai registri dove il primo valore è associato al registro r0, il secondo valore è associato al registro r1, il terzo valore è associato al registro r2, etc. In fase di codifica, il valore associato al registro da codificare sarà valorizzato a 1 mentre gli altri valori associati agli altri registri saranno valorizzati a 0. I restanti 31 valori sono utilizzati per codificare le operazioni matematiche speciali sui registri come ad esempio gli scalamenti. Dato che il primo operando, r0, del nostro esempio non necessita di operazioni matematiche speciali, la sequenza di 31 valori sarà tutta valorizzata a 0.

La codifica del secondo operando, similmente a quanto descritto per il primo operando, avviene su una sequenza di 239 valori che sono suddivisi in 4 sezioni per poter rappresentare i diversi tipi di operando.

Una prima sezione denominata "numero" è utilizzata per codificare un operando di tipo numerico su una sequenza di 4 valori. Una seconda sezione denominata "registri" è utilizzata per codificare i registri di memoria su una sequenza di 154 valori. Una terza sezione denominata "indirizzo" è utilizzata per codificare gli indirizzi di memoria su una sequenza di 65 valori. Una quarta sezione denominata "reg coproc" è utilizzata per codificare un registro di coprocessore matematico su una sequenza di 16 valori.

Nell'istruzione d'esempio il secondo operando è nuovamente un registro, r2, e pertanto sarà codificato nella seconda sezione "registri" mentre le sequenze di valori delle altre sezioni saranno tutte valorizzare a 0.

Dei 154 valori di detta sezione, i primi 123 valori sono associati ai registri dove il primo valore è associato al registro r0, il secondo valore è associato al registro r1, il terzo valore è associato al registro r2, etc. In fase di codifica, il valore associato al registro da codificare sarà valorizzato a 1 mentre gli altri valori associati agli altri registri saranno valorizzati a 0. I restanti 31 valori sono utilizzati per codificare le operazioni matematiche speciali sui registri

come ad esempio gli scalamenti. Dato che il secondo operando, r2, del nostro esempio non necessita di operazioni matematiche speciali, la sequenza di 31 valori sarà tutta valorizzata a 0.

La codifica del terzo operando avviene su una sequenza di 239 valori che sono suddivisi in 4 sezioni, come per il secondo operando, per poter rappresentare i diversi tipi di operando. Una prima sezione denominata "numero" è utilizzata per codificare un operando di tipo numerico su una sequenza di 4 valori. Una seconda sezione denominata "registri" è utilizzata per codificare i registri di memoria su una sequenza di 154 valori. Una terza sezione denominata "indirizzo" è utilizzata per codificare gli indirizzi di memoria su una sequenza di 65 valori. Una quarta sezione denominata "reg coproc" è utilizzata per codificare un registro di un coprocessore matematico su una sequenza di 16 valori.

Nell'istruzione d'esempio il terzo operando è un numero, 5, pertanto sarà codificato nella prima sezione "numero" mentre le sequenze di valori delle altre sezioni saranno tutte valorizzare a 0.

Il primo valore della sezione "numero" è valorizzato a 1 quando l'operando contiene un valore numerico, altrimenti è valorizzato a 0. Il secondo valore della sezione "numero" è valorizzato a 1 quando il valore numerico non è NaN o infinito, altrimenti è valorizzato a 0. Il terzo valore della sezione "numero" è valorizzato con il valore numerico dell'operando, che nell'esempio è uguale a 5. Il quarto valore della sezione "numero" è valorizzato a 1 quando viene usata una notazione con il punto esclamativo (!), altrimenti è valorizzato a 0.

Secondo i criteri definiti per la sezione numero, il terzo operando dell'esempio, il numero 5, è codificato come 1150.

La codifica del quarto operando avviene su una sequenza di 239 valori che sono suddivisi in 4 sezioni, come per il secondo ed il terzo operando, per poter rappresentare i diversi tipi di operando. Una prima sezione denominata "numero" è utilizzata per codificare un

operando di tipo numerico su una sequenza di 4 valori. Una seconda sezione denominata "registri" è utilizzata per codificare i registri di memoria su una sequenza di 154 valori. Una terza sezione denominata "indirizzo" è utilizzata per codificare gli indirizzi di memoria su una sequenza di 65 valori. Una quarta sezione denominata "reg coproc" è utilizzata per codificare registro di un coprocessore matematico su una sequenza di 16 valori.

L'istruzione d'esempio non possiede un quarto operando ma dovrà comunque essere codificato per mantenere coerente la lunghezza della sequenza di valori. In questo caso la codifica i valori di tutte le sezioni saranno valorizzati a 0.

La codifica del quinto operando avviene su una sequenza di 16 valori compresi in un'unica sezione denominata "reg coproc" ed è utilizzata per codificare un registro di un coprocessore matematico su una sequenza di 16 valori.

L'istruzione d'esempio anche in questo caso non possiede un quinto operando ma dovrà comunque essere codificato per mantenere coerente la lunghezza della sequenza di valori. Come per il caso precedente, in mancanza dell'operando, i valori della sezione saranno valorizzati tutti a 0.

La codifica del sesto operando avviene su una sequenza di 4 valori compresi in un'unica sezione denominata "numero" è utilizzata per codificare un operando di tipo numerico su una sequenza di 4 valori.

L'istruzione d'esempio anche in quest'ultimo caso non possiede il sesto operando ma dovrà comunque essere codificato per mantenere coerente la lunghezza della sequenza di valori. Come per i casi precedenti, in mancanza dell'operando, i valori della sezione saranno valorizzati tutti a 0.

In conclusione, l'istruzione di esempio "0x1234 add r0, r2, 5" al termine della fase di codifica sarà rappresentata su una sequenza numerica di 1250 valori così composta:

- I primi 236 valori rappresentano l'opcode add e sono codificati con la sequenza (0.5741776823997498, 0.5895169377326965, 0.44707465171813965, 0.5283305644989014, …);
- i successivi 4 valori rappresentano l'indirizzo 0x1234 e sono codificati con la sequenza (1, 1, 4660, 0);
- i successivi 273 valori rappresentano il primo operando r0 e sono codificati con la sequenza (0, …, 0, 1, 0, …);
- i successivi 239 valori rappresentano il secondo operando r2 e sono codificati con la sequenza (0, …, 0, 0, 0, 1, 0, …);
- i successivi 239 valori rappresentano il terzo operando 5 e sono codificati con la sequenza (1, 1, 5, 0, …);
- i successivi 239 valori rappresentano il quarto operando assente e sono codificati con la sequenza (0, 0, 0, 0, 0, 0, 0, 0, 0, …);
- i successivi 16 valori rappresentano il quinto operando assente e sono codificati con la sequenza (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
- i successivi 4 valori rappresentano il sesto operando assente e sono codificati con la sequenza (0, 0, 0, 0).

Come detto in precedenza, la fase di codifica 140 può opzionalmente comprendere due ulteriori sotto-fasi che consentono ad una rete neurale di raggiungere più velocemente la convergenza: la sotto fase di mascheratura e la sotto fase di scalamento.

La sotto-fase di mascheratura ha il compito di eliminare ogni colonna dai campioni codificati appartenenti al dataset che presenta lo stesso valore in tutti i campioni. L'eliminazione di dette colonne è possibile in quanto la fase di codifica 140 è in grado di rappresentare più istruzioni di quelle che un processore sarebbe in grado di fare e inoltre alcuni dei valori non vengono mai utilizzati nella realtà. La sotto-fase di mascheratura

consente di ridurre la lunghezza delle sequenze codificate e pertanto la rete neurale in fase di addestramento non utilizza del tempo per soffermarsi su dati che non sono realmente importanti. Nel caso d'esempio la sotto-fase di mascheratura è in grado di ridurre la lunghezza della sequenza di valori da 1250 a 768.

La sotto-fase di riscalamento effettua un riscalamento dei valori all'interno di un range compreso tra 0 e 1. Sebbene i campioni codificati appartenenti al data set contengono per la maggior parte delle volte i valori 0 e 1, in alcuni casi contengono valori molto più grandi. Ad esempio, nel caso della codifica dell'indirizzo espresso con una sequenza lunga 4, il terzo valore della sequenza contiene l'indirizzo espresso in base decimale che nell'esempio descritto precedente valeva 4660. Altro esempio di valori presenti nella decodifica diverso da 0 e da 1 è ad esempio presente nella sezione "numeri" della codifica del primo, secondo, terzo, quarto e sesto operando. Nell'esempio precedente il terzo operando era il numero 5, codificato come terzo valore della sezione numero del terzo operando. L'operazione di riscalamento evita che ai numeri più grandi sia attribuito un peso maggiore in fase di addestramento a cui non sempre corrisponde maggiore importanza. Il peso maggiore attribuito ad un determinato valore nella fase iniziale dell'addestramento di una rete neurale rallenta il processo di apprendimento dato che la rete impiegherebbe più tempo per capire che il maggior peso attribuito a tali valori era privo di fondamento.

Addestramento

Quando la fase di codifica 140 è si è conclusa, i campioni codificati CHMP_COD possono essere utilizzati nella fase di addestramento 150 inviandoli alla rete neurale $NN(P_i)$.

Come detto in precedenza, in questa fase è possibile utilizzare qualsiasi rete neurale in grado di gestire le sequenze e con meccanismo di attenzione. In particolare, verranno descritti due possibili fasi di addestramento alternative, utilizzando due diverse architetture di dette reti neurali.

La figura 5 mostra una prima architettura semplificata basata su celle LSTM, una tipologia di rete neurale ricorrente con un meccanismo di memoria a lungo termine che consente l'elaborazione di sequenze di dati. Le informazioni di dette sequenze vengono memorizzate di modo che, grazie alla presenza di cicli, procedendo nella sequenza le informazioni memorizzate nelle celle siano di ausilio all'elaborazione dei nuovi dati. In questo modo la rete neurale è in grado di interpretare in ordine le istruzioni assembly contenute nei campioni codificati CHMP_COD. Una codifica semantica della funzione contenuta nel campione codificato CHMP_COD è utilizzata come input della rete neurale dove un primo strato 1S_LSTM di celle LSTM analizza le istruzioni assembly contenute nel campione codificato CHMP_COD. Dette celle LSTM sono celle di tipo bidirezionale in grado di analizzare le istruzioni assembly dalla prima all'ultima e in direzione opposta dall'ultima verso la prima.

Un meccanismo di attenzione ATT riceve i dati di output dal primo strato di celle 1S_LSTM da cui è in grado di estrarre i livelli di attenzione LIV_ATT delle singole istruzioni dando indicazione di quali istruzioni assembly contengono la protezione. Maggiore è il livello di attenzione, più è probabile che l'istruzione assembly faccia parte di una protezione. Tale livello di attenzione LIV_ATT corrisponde al secondo indice FAi,j,k, sopra descritto.

L'output del meccanismo di attenzione ATT è sommato all'output del primo strato di celle 1S_LSTM in modo che siano aggiunte le informazioni dei livelli di attenzione LIV_ATT ed entrano successivamente verso un secondo strato 2S_LSTM di celle LSTM, che analizza le istruzioni assembly contenute nel campione codificato.

L'output del secondo strato di celle 2S_LSTM entra in ingresso ad un ultimo strato TRAS_LIN dove una trasformazione lineare ed una funzione sigmoide sono utilizzati per calcolare i valori del primo indice PIi,j per ogni istruzione assembly in un intervallo compreso tra 0 e 1. Il valore finale (o punteggio) CLASS è dato dal punteggio ottenuto dall'ultima istruzione contenuta nel campione codificato CHMP_COD, ovvero la pseudo-istruzione

<end>. Detto punteggio più è prossimo al valore 1 e più è probabile la presenza di una protezione. Detto punteggio CLASS corrisponde al primo indice di probabilità PIi,j.

Una seconda architettura, come mostrato in figura 6, è basata sul sistema di tipo trasformer BERT (rappresentazioni dell'encoder bidirezionale da transformer), ovvero un'architettura non ricorrente con meccanismo di attenzione. Dato che questo tipo di rete neurale necessita che la funzione in ingresso abbia una lunghezza massima nota a priori, la precedente fase di codifica 140 effettua l'operazione di troncamento della funzione come mostrato in figura 4c. Per tale motivo in questo frangente l'ultima istruzione della funzione potrebbe non essere la pseudo-istruzione <end>.

Al tecnico del ramo sono note le particolarità di questa architettura di rete neurale ed i limiti superati presenti in architetture di reti neurali di tipo ricorrente.

La figura 6 mostra un'architettura semplificata basata su BERT dove una codifica semantica della funzione contenuta nel campione codificato CHMP_COD viene sommata ad una codifica posizionale della funzione per aggiungere un'informazione di posizione dalle istruzioni assembly contenute nel campione codificato CHMP_COD. Questa prima operazione si rende necessaria per esplicitare l'informazione di posizione dell'istruzione dato che la rete neurale di tipo transformer, non avendo ricorrenza, non possiede la nozione di posizione di un elemento all'interno di una sequenza. La codifica posizionale è un prerequisito alla fase di addestramento e sono dei coefficienti calcolati secondo formule standard memorizzati all'interno di una matrice.

Il risultato in output della fase precedente entra come input ad una serie di strati di encoding S_ENCOD che ne analizzano il contenuto, dove ogni strato di encoding possiede un meccanismo di attenzione che è utilizzato per valutare i livelli di attenzione LIV_ATT delle istruzioni assembly in modo analogo a quanto detto nell'architettura di rete neurale di tipo LSTM. Tali livelli di attenzione LIV_ATT corrispondono a valori del secondo indice

FAi,j,k, sopra descritto.

L'output dagli strati di encoding S_ENCOD entra in ingresso ad un ultimo strato TRAS_LIN dove una trasformazione lineare ed una funzione sigmoide sono utilizzati per calcolare i punteggi di classificazione per ogni istruzione assembly in un intervallo compreso tra 0 e 1. A differenza di quanto accade nello strato TRAS_LIN di una rete LSTM, il valore del punteggio finale CLASS è relativo alla prima istruzione contenuta nel campione codificato CHMP_COD, ovvero la pseudo-istruzione <begin>. Anche in questo caso il punteggio finale CLASS corrisponde al primo indice di probabilità PIi,j, sopra introdotto. Si noti che oltre ai due esempi descritti è possibile procedere all'addestramento di una qualsiasi architettura, tra quelle in grado di gestire sequenze e/o aventi un meccanismo di attenzione utilizzando un qualsiasi algoritmo di addestramento standard.

La soluzione descritta produce una metrica qualitativa dell'identificazione delle tecniche di protezione utilizzabile come stima della qualità della soluzione di protezione scelta, introducendo un elevato grado di automazione nell'individuazione delle tecniche di protezione applicate ad un file e delle aree protette all'interno del file stesso.

Si noti che i risultati ottenuti mediante l'applicazione del metodo della presente invenzione possono essere utilizzati come strumento di validazione dell'esposizione al rischio degli asset, come validazione dell'invisibilità di tecniche di protezione sviluppate e come identificazione dei metodi con cui vengono offuscati virus e malware per contribuire all'aggiornamento di strumenti di antivirus.

La soluzione descritta permette quindi raggiungere un livello di protezione del software più elevato a parità di tempo impiegato o un livello di protezione equivalente a quella ottenibile dalle tecniche note in un intervallo di tempo sensibilmente inferiore.

RIVENDICAZIONI

1. Un metodo (100) di configurazione di reti neurali, comprendente le fasi di:
   a) definire (110; 120) una pluralità di funzioni (asm) e applicare una pluralità di protezioni software (P_1,..,P_n) a dette funzioni;
   b) costruire (130) un data set comprendente una pluralità di campioni ciascuno includente una funzione (asm_i) della pluralità e almeno una di dette protezioni software (P_1,..,P_n) applicata alla rispettiva funzione;
   c) codificare (140) ogni funzione (asm_i) del data set per ottenere una pluralità di campioni codificati (CHMP_COD) ciascuno espresso come sequenza di valori numerici;
   d) addestrare (150) una rete neurale (NN(Pi)) mediante la pluralità di campioni codificati (CHMP_COD) in modo che sia in grado di processare un file da analizzare e fornire informazioni relative a protezioni del software applicate a detto file da analizzare.
2. Il metodo (100) della rivendicazione 1, in cui:

   detta fase di definire la pluralità di funzioni (110, 120) inoltre comprende la definizione di una pluralità di funzioni vanilla alle quali non sono applicate protezioni software;

   detto data set comprende un'ulteriore pluralità di campioni ciascuno includente una funzione (asm_i) della pluralità di funzioni vanilla.
3. Il metodo (100) della rivendicazione 1, in cui la fase di definire (110, 120) la pluralità di funzioni comprende le fasi di:

   fornire un file sorgente comprendente detta pluralità di funzioni (asm);

applicare (110) la pluralità di protezioni software (P_1,..,P_n) alla pluralità di funzioni (asm) del file sorgente e compilare il file sorgente provvisto delle protezioni software ottenendo un file compilato binario;

disassemblare (120) il file compilato binario ottenendo un file in formato assembly ed estrarre la pluralità di funzioni (asm) dal file in formato assembly.

4. Il metodo (100) della rivendicazione 1, in cui la fase di definire (110, 120) la pluralità di funzioni comprende le fasi di:

fornire un file binario comprendente detta pluralità di funzioni (asm) alle quali sono applicate una pluralità di protezioni software (P_1,..,P_n);

disassemblare (120) il file binario ottenendo un file in formato assembly ed estrarre la pluralità di funzioni (asm) dal file in formato assembly.

5. Il metodo (100) della rivendicazione 1, in cui detta rete neurale ($NN(P_1)$) è associata ad una singola tipologia di protezione del software (P_1).

6. Il metodo (100) della rivendicazione 2, in cui detta rete neurale ($NN(P_1)$) è configurata in modo che dette informazioni comprendano un primo indice di probabilità ($PI_{1,j}$) indicativo di una probabilità che una prima funzione (asm_j) della pluralità di funzioni sia stata protetta dalla prima protezione (P_1).

7. Il metodo (100) della rivendicazione 3, in cui detta rete neurale ($NN(P_1)$) è configurata in modo che dette informazioni comprendano un secondo indice (FAi,j,k) che rappresenta una possibilità che ad istruzioni di una specifica area della prima funzione sia stata applicata la prima protezione (P_1).

8. Il metodo (100) della rivendicazione 3, in cui la pluralità di protezioni software (P1,..,Pn) comprende almeno una delle seguenti protezioni: control flow flattening, predicati opachi, branch functions, encode arithmetic, conversione di dati in funzioni, fusione o suddivisione di variabili, ricodifica di variabili, white-box cryptography,

virtualizzazione mediante l'uso di virtual machine o di compilazione JIT, controlli sul call stack, code guards, control flow tagging, anti-debugging, code mobility, client/server code splitting, anti-cloning e software attestation.

9. Il metodo della rivendicazione 1, in cui detto metodo è effettuato per configurare una pluralità di reti neurali (NN(Pi) ciascuna associata ad una relativa protezione del software.

10. Il metodo (100) della rivendicazione 1, in cui detta rete neurale ($NN(P_i)$) è realizzata secondo almeno una delle seguenti tipologie di rete neurale: rete in grado di gestire sequenze, rete avente un meccanismo di attenzione.

11. Il metodo (100) della rivendicazione 9, in cui detta rete neurale ($NN(P_i)$) è realizzata secondo almeno una delle seguenti tipologie di rete neurale: rete LSTM, rete BERT, rete GRU, rete transformer-XL.

12. Il metodo (100) della rivendicazione 1, in cui la pluralità di campioni codificati include una pluralità di sequenze di valori numerici e detta fase di codifica comprende inoltre:

una fase di mascheratura in cui si eliminano dalla pluralità di sequenze di valori numerici valori ripetuti in ogni sequenza della pluralità;

una fase di riscalamento di detti valori numerici all'interno di un intervallo prestabilito.

13. Un metodo di elaborazione di file, comprendente le fasi di:

- fornire un file binario da analizzare includente una pluralità di funzioni da analizzare;
- disassemblare il file binario da analizzare per ottenere un file assembly;
- estrarre dal file assembly la pluralità di funzioni da analizzare(asm);
- codificare ogni funzione da analizzare esprimendola come relativa sequenza di valori numerici;

- fornire una pluralità di reti neurali ($NN(P_i)$), ciascuna associata ad una relativa protezione del software ($P_1,\ldots,P_n$), configurate secondo il metodo di configurazione (100) di almeno una delle rivendicazioni precedenti;
- processare la pluralità di funzioni da analizzare (asm) mediante la pluralità di reti neurali ($NN(P_i)$) per ricercare informazioni relative a protezioni del software all'interno della pluralità di funzioni da analizzare.

14. Il metodo della rivendicazione 13, in cui ciascuna di dette reti neurali (NN(Pi)) è associata ad una rispettiva tipologia di protezione del software (Pi).

15. Il metodo della rivendicazione 14, in cui processare la pluralità di funzioni da analizzare (asm) mediante la pluralità di reti neurali (NN(Pi)) restituisce informazioni di classificazione includenti una pluralità di indici di probabilità (IPi) ciascuno indicativo di una probabilità che una relativa funzione (asm_i) della pluralità di funzioni sia protetta da una di dette protezioni (P_i).

16. Il metodo della rivendicazione 14, in cui processare la pluralità di funzioni da analizzare (asm) mediante la pluralità di reti neurali (NN(Pi)) restituisce informazioni di posizione includenti una pluralità di secondi indici (FAi,j,k) ciascuno indicativo di una possibilità che ad istruzioni di una specifica area di una funzione sia stata applicata una corrispondente protezione (Pi).

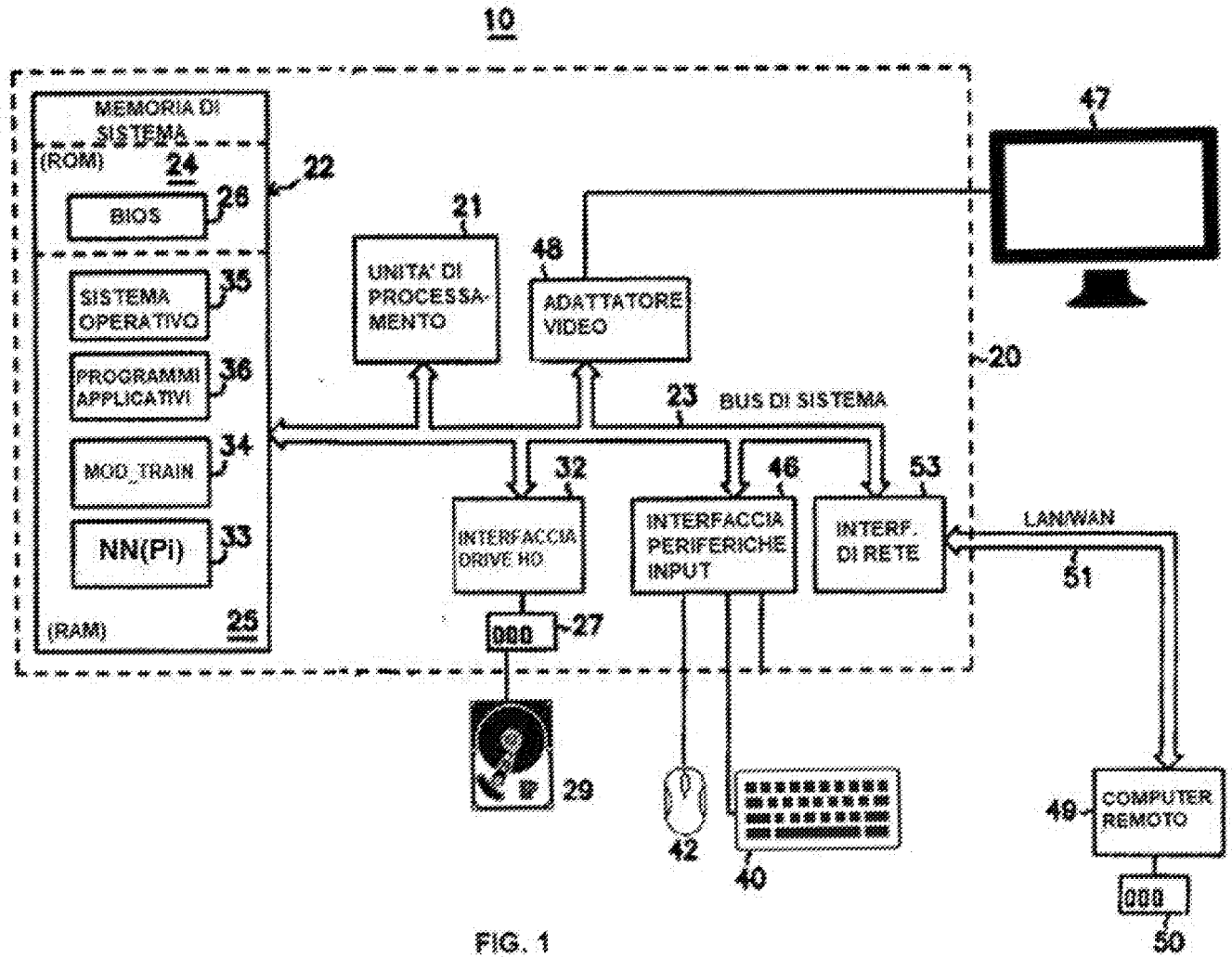

10
MEMORIA DI SISTEMA
(ROM)
24
22
BIOS
28
SISTEMA OPERATIVO
35
PROGRAMMI APPLICATIVI
36
MOD_TRAIN
34
NN(Pi)
33
(RAM)
25
21
UNITA' DI PROCESSA-MENTO
48
ADATTATORE VIDEO
47
20
23
BUS DI SISTEMA
32
INTERFACCIA DRIVE HD
27
46
INTERFACCIA PERIFERICHE INPUT
53
INTERF. DI RETE
LAN/WAN
51
29
42
40
49
COMPUTER REMOTO
50

FIG. 1

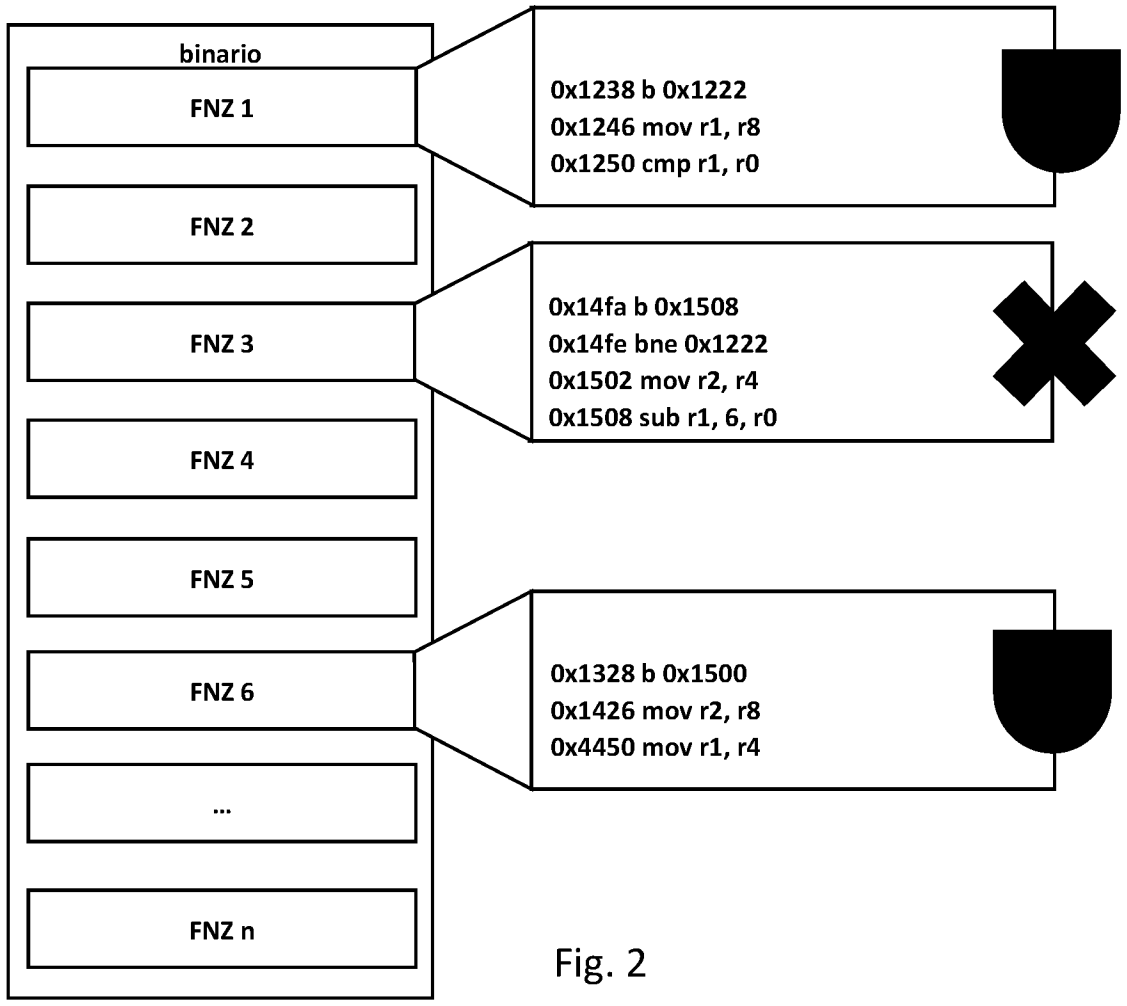

binario
FNZ 1
FNZ 2
FNZ 3
FNZ 4
FNZ 5
FNZ 6
...
FNZ n
0x1238 b 0x1222
0x1246 mov r1, r8
0x1250 cmp r1, r0
0x14fa b 0x1508
0x14fe bne 0x1222
0x1502 mov r2, r4
0x1508 sub r1, 6, r0
0x1328 b 0x1500
0x1426 mov r2, r8
0x4450 mov r1, r4

Fig. 2

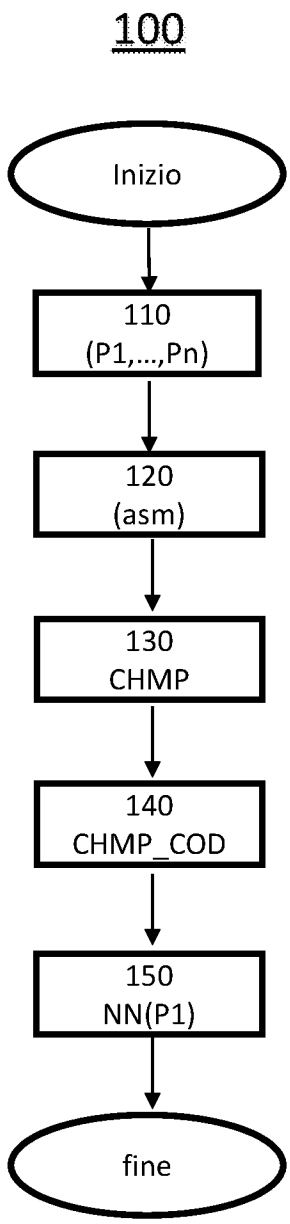
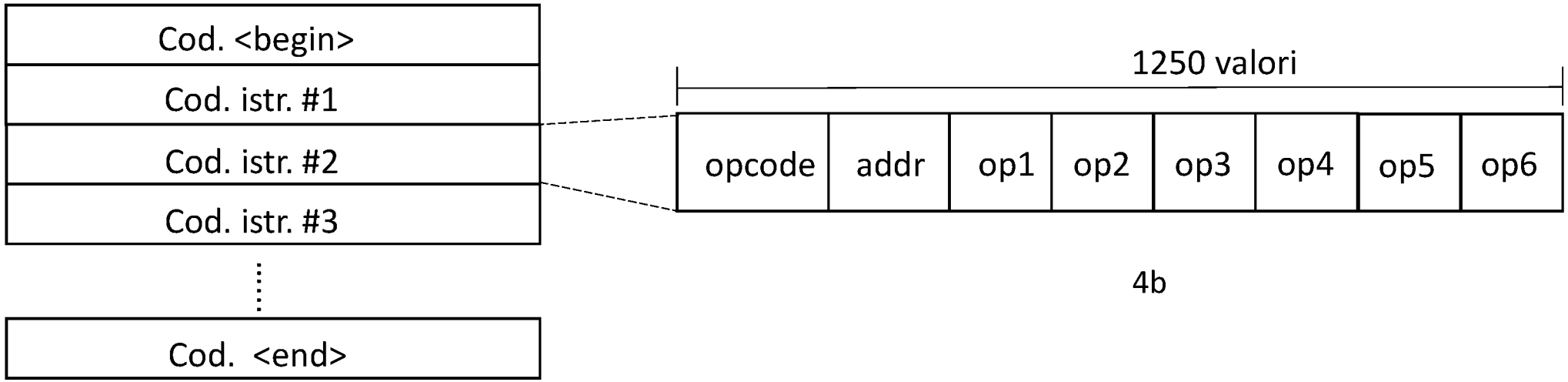

100
Inizio
110
(P1,…,Pn)
120
(asm)
130
CHMP
140
CHMP_COD
150
NN(P1)
fine

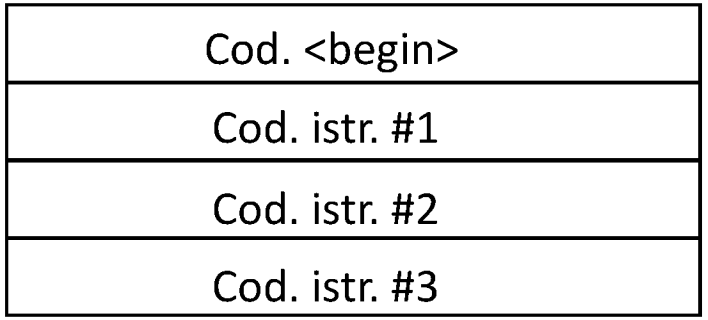
Fig. 3


Cod. <begin>
Cod. istr. #1
Cod. istr. #2
Cod. istr. #3
Cod. <end>
1250 valori
opcode
addr
op1
op2
op3
op4
op5
op6

4a

4b

Fig. 4


Cod. <begin>
Cod. istr. #1
Cod. istr. #2
Cod. istr. #3

4c

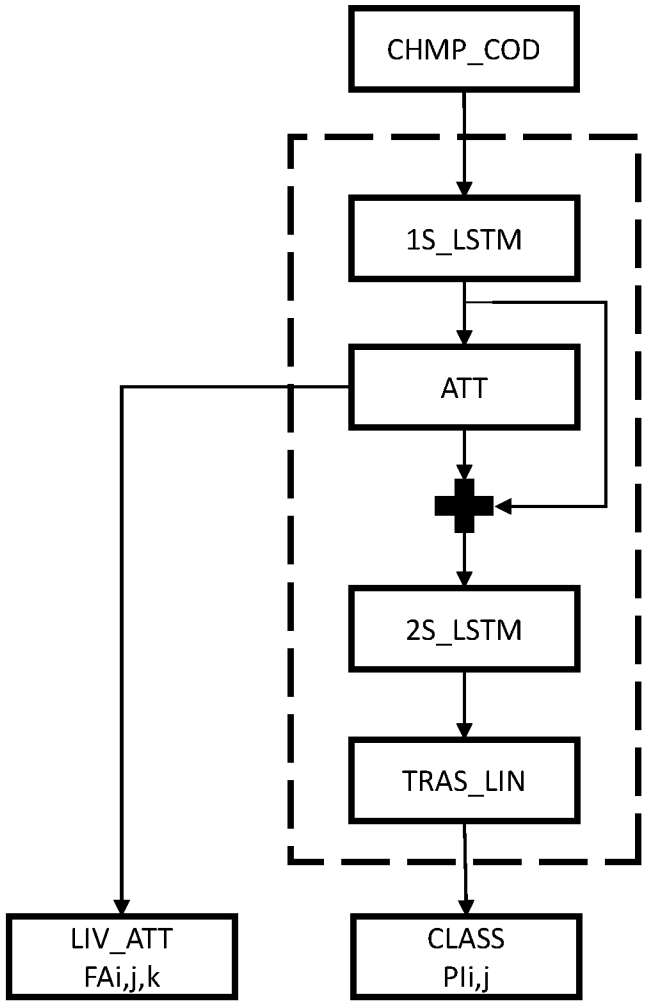


Fig. 5

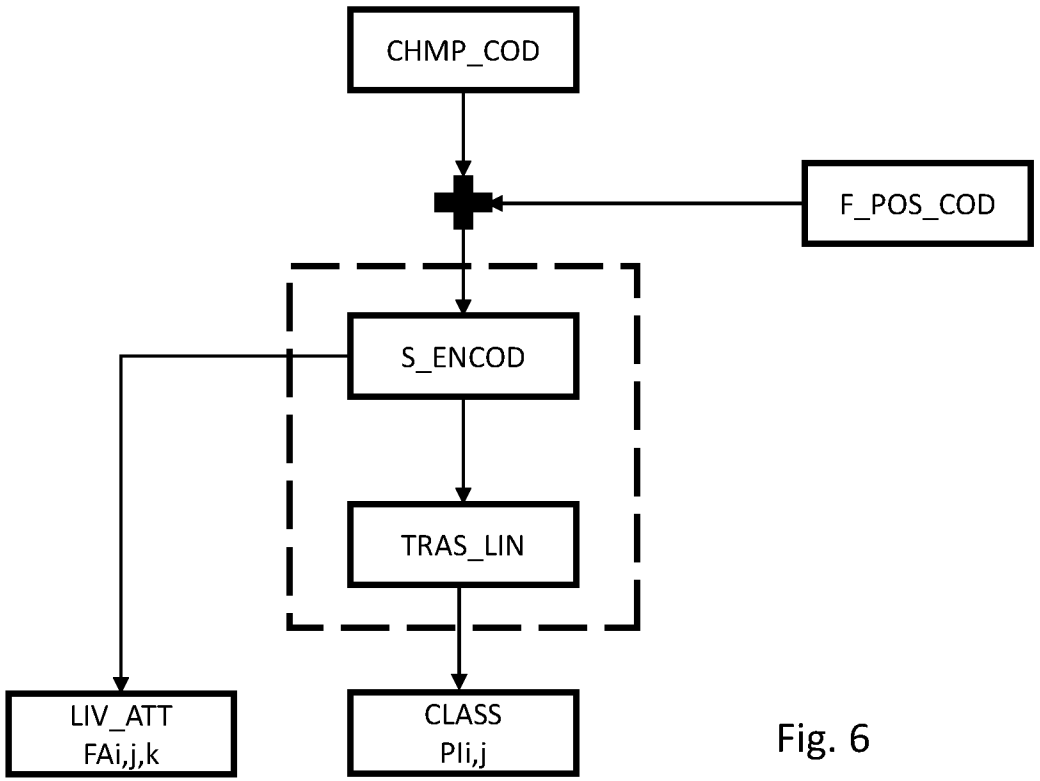


Fig. 6