1 Auto format string tool

Auto format string tool è un tool scritto in python che, sfruttando una vulnerabilità format string presente nel programma target, calcola la stringa con padding necessaria per la scrittura di un valore ad uno specifico indirizzo di memoria.
Per il corretto funzionamento del tool è richiesto che meccanismi di sicurezza come stack protector e ASLR siano disabilitati, così che sia possibile calcolare il padding corretto attraverso più esecuzioni del programma target.

1.1 Format string exploit

La vulnerabilità di format string sfrutta un cattivo uso, da parte del programmatore, delle funzioni di output o stampa che consentono la formattazione di valori salvati nella memoria del programma.
Per semplicità nei seguenti esempi prenderemo in considerazione, in particolare, il caso della funzione printf in C, ma questo exploit è applicabile anche in altri linguaggi di programmazione e con diverse funzioni di stampa.
La funzione printf, quando viene specificato un elemento di formattazione come *%s*, effettua un pop dello stack, interpreta il valore ottenuto come stringa e lo sostituisce a tempo di compilazione a *%s* per stamparlo. In un caso come *printf("val1: %d, val2: %d, val3: %d", n1, n2, n3)* i valori n1, n2 ed n3 vengono aggiunti nello stack sopra la funzione di stampa, e printf effettua 3 pop per recuperarli, interpretarli come decimali e stamparli a video. In un caso invece come *printf("val1: %d")* in cui non è stato specificato alcun argomento, printf effettua comunque un pop recuperando il valore successivo presente nello stack, leggendo memoria che non era in origine destinata ad essere stampata.
Supponiamo ora che all'utente venga consentito di assegnare un valore a una stringa di nome *userinput* attraverso una funzione scanf, e che questa stringa venga poi fornita come input diretto per una printf con *printf(userinput)*, nel caso in cui l'utente inserisca *"%d %d %d"* la funzione printf interpreterà i 3 %d come formattatori, stampando a video i 3 elementi successivi dello stack. In una situazione come questa l'utente ha completa libertà di leggere il contenuto della memoria del programma aggiungendo un quantitativo di %d a piacere.

1.1.1 Leggere la memoria di un programma

Come abbiamo visto nell'esempio precedente, è possibile leggere la memoria di un programma sfruttando una sequenza di formattatori, ma è anche possibile utilizzare la sintassi di direct parameter access per accedere direttamente alla posizione desiderata dello stack: usando $n%y, dove n è l'offset rispetto alla posizione attuale e y è il tipo in cui formattare il valore ottenuto, riusciremo a leggere un indirizzo di memoria senza dover prima effettuare pop degli elementi che lo precedono; ad esempio $5%x stamperà il valore trovato a distanza 5 sullo stack come valore esadecimale.
Questo tipo di accesso risulta possibile con la maggior parte delle librerie C e semplifica notevolmente l'exploit: questo tool utilizza il direct parameter access, poiché la scrittura automatizzata di un valore arbitrario a un determinato indirizzo senza di esso risulterebbe molto complessa e più frequentemente impossibile.

1.1.2 Scrivere valori in memoria usando printf

Fra i formattatori di stampa ne esiste uno molto particolare: %n. Esso recupera il prossimo argomento dello stack, lo interpreta come indirizzo di memoria e scrive a quell'indirizzo il numero di caratteri stampati a video fino a quel momento. Questo formattatore permette quindi di effettuare un exploit che scriva valori arbitrari nella memoria del programma a condizione che l'indirizzo a cui scrivere sia presente sullo stack e sia raggiungibile con una sequenza di pop o con direct parameter access.
Nel caso in cui l'indirizzo a cui si desidera scrivere non sia presente in memoria è comunque possibile specificarlo: potremmo ad esempio usare la stessa scanf con cui inseriamo formattatori per inserire l'indirizzo a cui desideriamo scrivere; specificando *"AAAA $32%p \x25\x25\x25\x25"*, supponendo che 32 sia l'offset necessario a raggiungere i byte dell'indirizzo contenuti nella stringa che abbiamo specificato, stamperemo 0x25252525. Scrivendo invece *"AAAA $32%n \x25\x25\x25\x25"* scriveremo il valore 5 (poiché prima di %x abbiamo stampato *"AAAA "* di lunghezza 5) all'indirizzo di memoria 25252525.
Un aspetto importante da ricordare è che nel caso di architettura a 64 bit gli indirizzi conterranno spesso uno o più null byte *\x00*, che verranno interpretati dalla printf come caratteri di terminazione della stringa ed interromperanno la stampa: per evitare di incorrere in questo problema è sufficiente appendere gli indirizzi a cui scrivere al termine della stringa di exploit, così che non impediscano la stampa dei caratteri necessari a raggiungere il valore che si desidera scrivere.

Un altro aspetto da considerare è il fatto che il programmatore potrebbe aggiungere un limite massimo al numero di caratteri accettati dalla funzione di input: è possibile ridurre la probabilità di incorrere in questo problema specificando, ad esempio, un *%300d* piuttosto che passando 300 caratteri di padding.
Infine è importante ricordare che la printf non garantisce la stampa di più di 4095 caratteri: per non incorrere in problemi di questo tipo il tool spezza la singola scrittura in più sottoscritture (come specificato di seguito) fino a quando la somma dei singoli sottovalori non risulta minore di 4000.

1.2 Analisi del tool

Il tool si compone di quattro classi principali:

- **Opt**, in **main.py**, che si occupa di verificare ed effettuare il parsing degli input del tool;
- **Exploit**, in **exploit.py**, che si occupa di verificare se il programma target è vulnerabile all'exploit delle format string e prova a metterlo in pratica;
- **DirectFmtGenerator**, in **format_string_generator.py**, che si occupa di generare i diversi tipi di stringhe di exploit usate da **Exploit**;
- **Logger**, in **logger.py**, che si occupa di creare il file di log per il programma in test, che andrà a contenere i risultai dei test effettuati sul programma target.

1.2.1 Parametri accettati e file di configurazione

Il tool accetta quattro parametri da linea di comando:

- **target**: path relativo del programma target su cui effettuare l'exploit;
- **target_address**: indirizzo target del programma su cui effettuare la scrittura, è interpretato come valore decimale o esadecimale se preceduto da *0x*;
- **target_value**: valore da scrivere al target_address, è interpretato come valore decimale o esadecimale se preceduto da *0x*;
- **input_file**: path relativo del file json contenente gli input da passare al programma target.

Il file di input json si compone di un array salvato con chiave *input*, ed è strutturato nel seguente modo:

```
{
  "input": [
    {
      "type": ...,
      "marker": ...,
      "value": ...
    },
    ...
  ]
}
```

dove:

- **type** specifica il tipo dell'input e può assumere valore *"command_line_input"* se passato da linea di comando o *"execution_input"* se passato a tempo di esecuzione (ad esempio quando l'input è richiesto da una *scanf()*);
- **marker** specifica la risposta in stampa che il tool si aspetta di ricevere dal programma target prima di inviare l'input associato. È presente solo se **type** è "command_line_input" e può assumere due valore:
  - una stringa non vuota, se esiste e si conosce la risposta;
  - *null* se non esiste o non si conosce la risposta.
- **value**: input effettivo da passare al programma target. Può assumere due valori:

- una stringa vuota, nel caso in cui il programma target non imponga vincoli sull'input per proseguire con la sua esecuzione;
- il valore specifico atteso dal programma target per quell'input.

Nel caso in cui venga assegnata una stringa vuota, il tool considera l'input come testabile per l'exploit.

Supponiamo che il nostro programma target richieda due input da linea di comando, *argv[1]* e *argv[2]*, e che l'exploit sia attuabile tramite una *printf()* su *argv[2]*. Supponiamo inoltre che tale funzione sia raggiungibile solo se *argv[1]* è uguale alla stringa *"test"*. In questo caso, il programma impone un vincolo sul valore di *argv[1]* (lasciando "libero" *argv[2]*), e quindi il file di input sarà così strutturato:

```
{
  "input": [
    {
      "type": "command_line_input",
      "value": "test"
    },
    {
      "type": "command_line_input",
      "value": ""
    }
  ]
}
```

Ovviamente gli input specificati all'interno dell'array devono essere ordinati secondo l'ordine con cui vengono richiesti dal programma target. In questo caso, il primo elemento rappresenta *argv[1]*, mentre il secondo *argv[2]*.

Il tool è ulteriormente configurabile tramite il file **options.ini**, dove:

- **bytes_to_write** specifica il numero di byte da scrivere al target_address. Può assumere valore *1*, *2*, *4* o *8*;
- **wait_time_marker** specifica il tempo di attesa in caso di presenza di **marker** quando l'input è passato a tempo di esecuzione. Può assumere come valore un numero positivo o *-1*;
- **wait_time_no_marker** specifica il tempo di attesa in caso di assenza di **marker** prima di inviare il prossimo input. Può assumere come valore un numero positivo;
- **log_dir** specifica il path relativo alla cartella che andrà a contenere i file di log;
- **input_len_control** specifica se permettere al controllo sul numero di caratteri stampati dell'input di bloccare il processo o meno. Può assumere valore *True* o *False*.

1.2.2 Inizializzazione

Una volta verificati gli input passati da linea di comando e i valori assegnati alle proprietà del file di configurazione (es: la dimensione del valore, in byte, è conforme al numero specificato in **byte_to_write**), il tool procede ad estrarre gli input specificati nel file di input json sotto forma di dizionario e separarli in due liste in base al valore dell'etichetta **type**.

Per evitare di passare al programma target delle stringhe vuote come input, il tool sostituisce a ciascun elemento con **value** una stringa vuota la stringa *"PPPP"* e associa ad ogni input una nuova etichetta, chiamata **free**, che specifica se il **value** dell'input era una stringa vuota (*True*) o meno (*False*).

1.2.3 Formattazione dei valori in byte e decisione della write size

Prima di iniziare a testare gli input, il tool esegue una serie di operazioni sul valore da scrivere e l'indirizzo target:

1. Decide i parametri con cui il valore (o i valori, come si vedrà in seguito) da scrivere verrà formattato in byte;

2. Spezza il valore da scrivere in più sottovalori, in modo che la loro somma sia inferiore a 4000, per evitare di superare il limite superiore di caratteri di stampa garantiti dalla *printf()*: partendo dalla dimensione di scrittura specificata nella proprietà **bytes_to_write** del file di configurazione, il tool itera più volte spezzando ad ogni ciclo il sottovalore massimo nella dimensione di scrittura successiva (in ordine decrescente 8, 4, 2 e 1).

   Ad esempio, un valore esadecimale come *0x003f0046* pari a 4128838 verrebbe spezzato in *0x003f* pari a 63 e *0x0046* pari a 70, entrambi con dimensione di scrittura 4, e l'iterazione terminerebbe poiché la loro somma, 133, risulta minore di 4095. Il tool ordina salva poi i nuovi sottovalori generati e le relative dimensioni di scrittura;

3. Determina la serie di indirizzi target a cui scrivere, nel caso in cui il valore non sia più singolo. Per ciascun sottovalore, l'indirizzo in cui dovrebbe essere scritto corrisponde a quello del sottovalore precedente incrementato della dimensione di scrittura di quest'ultimo. Nel caso del primo sottovalore, l'indirizzo corrisponderà all'originale indirizzo target.

   Ad esempio, supponendo che i valori valori da scrivere siano *a*, *b*, *c*, con rispettivamente 4, 4, 2 come dimensione di scrittura, e che l'indirizzo target fornito dall'utente sia *0x0*, allora avremo *0x0* per il valore *a*, *0x4* per il valore *b* e *0x8* per il valore *c*.

   Le coppie sottovalore-indirizzo vengono ordinate, in ordine crescente, in base al sottovalore.

4. Crea una serie di "scrittori", uno per ogni sottovalore, che verrà utilizzato durante la creazione della stringa finale di exploit. Uno scrittore è una lista di tre elementi, dove:

   - Nella prima posizione viene salvato il padding necessario a raggiungere il sottovalore da scrivere. Il padding per il primo sottovalore corrisponderà esattamente al sottovalore, per il secondo alla differenza tra il primo sottovalore e il secondo, per il terzo alla differenza per il terzo sottovalore e i primi due e così via;
   - Nella seconda posizione viene salvata la posizione, nello stack, dell'indirizzo in cui si vuole scrivere il sottovalore. In questa fase viene lasciata vuota, in quanto viene determinata durante la creazione della stringa finale;
   - Il formattatore delle *printf()* utilizzato per scrivere valori in un indirizzo. Il formattatore può essere *"ln, "n", "hn, "hnn"* in base alla dimensione di scrittura del sottovalore (rispettivamente 8, 4, 2, 1 byte).

   La stringa rappresentata il padding per il sottovalore può assumere due forma differenti:

   - Se il padding da scrivere rientra nell'intervallo 1 e 7 compresi, la stringa sarà formata da esattamente quel numero di caratteri. Quindi, ad esempio, se il padding è 6, la stringa sarà *"AAAAAA"*;
   - Se il valore da scrivere è maggiore o uguale a 8, la stringa usata sarà *"%Nx"*, dove *N* rappresenta il padding da scrivere.

   Si è deciso di usare il *%x* in quanto è uno dei pochi formattatori che non hanno generato problemi durante i test del tool. In quanto non è possibile troncare il numero di cifre stampate dal formattatore, ma solo anteporvi del "padding", il limite è stato impostato ad 8 in quanto è il numero massimo di cifre che un valore stampato dal formattatore può avere.

5. Calcola la dimensione in caratteri della serie di indirizzi e scrittori creati. Il valore ottenuto rappresenta il numero minimo di caratteri dell'input che il programma target dovrebbe stampare affinché l'exploit possa essere eseguito.

   Supponiamo che, in totale, scrittori e indirizzi siano lunghi 70 e che l'exploit del programma target avvenga tramite *printf(var)*. Se la *printf()* stampasse, ad esempio, solo 60 caratteri, significherebbe che *var* conterrebbe solo una parte della stringa di exploit, rendendolo quindi, teoricamente, impossibile.

   Ovviamente questo numero non rappresenta la dimensione totale della stringa di exploit, ma solo quella calcolabile fino a questo momento, in quanto sono assenti le posizioni nello stack degli indirizzi cui scrivere nonché l'eventuale padding di allineamento da anteporre loro (che verranno discussi più avanti).

1.2.4 Passaggio degli input ed estrazione della risposta dal programma target

Vediamo ora come il tool esegue il programma target, gli passa gli input ed estrae una risposta:

1. Il programma target viene lanciato passandogli, come argomenti da linea di comando, tutti gli input con **type** *"command_line_input"*;
2. Il tool verifica se esistono input con **type** *"execution_input"*:
   - Se non ne esistono, raccoglie i dati stampati dal programma target finché non termina la sua esecuzione (raggiunge il suo End Of File);
   - Se ne esistono, verifica se l'input da inviare ha specificata l'etichetta **marker**:
     - Se è specificata (il suo valore è una stringa non vuota), il tool raccoglie i dati fino al raggiungimento di tale valore ed invia il valore contenuto di **value** concatenato ad un "a capo". Se **wait_marker** ha valore *-1*, l'attesa del raggiungimento del valore di **marker** non ha limiti temporali (può essere pressoché infinita), in caso contrario il tool attende un numero di secondi pari al valore specificato. Se **marker** non è stato raccolto entro il limite di tempo, il tool scarta i dati raccolti in questo punto e invia il **value**;
     - Se non è specificata, il tool raccoglie i dati per **wait_no_marker** secondi ed invia il **value** concatenato ad un "a capo";

   Questo fase di "raccolta-invio" è ripetuta per tutti gli input individuati. Inviato l'ultimo **value**, il tool semplicemente raccoglie i dati fino al termine di target.
3. L'insieme di dati raccolti viene usato come risposta del processo.

Nel caso in cui durante (1) o (2) il programma target generi errori di Broken Pipe, EOF o Segmentation Fault, la risposta viene scartata. Generalmente questi errori, eccetto dove specificato diversamente, portano a scartare l'input in test e passare al successivo.

1.2.5 Test degli input

Ottenuta la dimensione minima di caratteri, il tool procede con i test per verificare se il programma target è exploitabile o meno, considerando solo gli input che avevano, nel file json, **value** uguale ad una stringa vuota (e ora **free** uguale a *True*), ovvero solo quegli input che non hanno vincoli inposti dal programma target e che quindi possono essere modificati.

L'algoritmo di test procede come segue:

1. Il tool verifica se il **type** dell'input in test è *"command_line_input"* e se almeno uno degli indirizzi a cui scrivere contiene un null byte. In caso affermativo, il tool scarta l'input, altrimenti prosegue. Questo controllo viene eseguito in quanto, essendo il null byte usato per separare gli input da linea di comando, l'exploit risulterebbe impossibile;
2. Il tool esegue il programma target e, nella risposta ottenuta, conta il numero di ripetizioni delle stringhe *"0x"*, *(nil)* e *"%p"*;
3. Il tool verifica se il programma target è effettivamente vulnerabile all'exploit in oggetto, associando al **value** dell'input in test la stringa *"%p"*. Dalla nuova risposta ottenuta ricalcola le ripetizioni delle stringhe specificate al punto (2) e le confronta con quelle precedentemente ottenute. Il tool scarterà l'input se si verifica uno di questi quattro casi:
   - Il numero di *"%p"* differisce;
   - La differenza tra il numero di *"(nil)"* precedente e corrente è diversa da 1;
   - La differenza tra il numero di *"0x"* precedente e corrente è diversa da 1;
   - La differenza tra il numero di *"(nil)"* precedente e corrente e la differenza tra il numero di *"0x"* precedente e corrente sono entrambe 1.

   Quindi, attualmente, il tool scarterà l'input se questo viene "stampato" nella risposta più di una volta o se effettivamente non è vulnerabile;

4. Il tool verifica se l'input è "stampato" nella risposta per la dimensione minima di caratteri richiesta. Per fare ciò genera una stringa randomica lunga, in caratteri, quanto la dimensione minima, la assegna all'input, esegue il programma target e procede a contare il numero di volte che si ripete nella risposta ottenuta. Se il numero ottenuto è uguale al numero di ripetizioni della stessa rispetto alla risposta ottenuta al punto (2), allora l'input non raggiunge la dimensione minima di stampa. L'algoritmo può ora procedere in due modi:
   - Se la proprietà **input_len_control** del file di configurazione è settata a *True*, il tool scarta l'input;
   - Se la proprietà è settata a *False*, il tool prosegue con l'input corrente e assegna alla dimensione minima il valore 0.
5. Il tool determina due marcatori (stringhe) unici, che possono essere uguali tra loro, rispetto alla risposta ottenuta al punto (4). Questi verranno usati per incapsulare l'input passato al programma target, semplificandone poi l'estrazione dalla risposta. I marcatori possono avere una lunghezza minima di 2 caratteri, fino ad una massima di 50. Per ogni lunghezza si effettuano fino a 200 tentativi di creazione; nel caso in cui il tool non riesca a creare marcatori non ripetuti nella risposta l'input viene scartato;
6. Il tool determina la dimensione stampata dell'input, fino ad un numero di caratteri pari a

   $$500 + \text{lunghezza dei marcatori } + \text{dimensione minima}$$

   Per fare ciò eseguirà più volta il programma target, associando all'input in test il marcatore sinistro ripetuto un numero n di volte, incrementato ad ogni esecuzione, fino a che l'incremento successivo non supera il numero massimo di caratteri (o l'esecuzione ritorna 2 volte uno qualsiasi tra segmentation fault e EOF error). La dimensione stampata viene quindi determinata, a partire dall'ultima risposta ottenuta, moltiplicando il numero di ripetizioni del marcatore per la sua dimensione.
7. Il tool procede a determinare la posizione dell'input passato all'interno dello stack, a creare la stringa di exploit finale e a testarla. Nel caso di fallimento in uno qualsiasi dei tre passaggi, il tool, come per le fasi precedenti, scarta l'input e passa al successivo.

Nel caso in cui l'input venga scartato, prima di passare al test dell'input seguente, il tool:

- Riassocia al **value** dell'input la stringa *"PPPP"* e setta l'etichetta **Free** a *False*;
- Resetta i valori associati al padding di allineamento della stringa finale;
- Resetta i valore associato al direct parameter access del generatore di stringhe, che indica ora l'utima posizione dello stack stampata dalla stringa di ricerca;
- Ripristina i valori associati agli scrittori. Come si vedrà in seguito, per ogni input testabile, l'exploit viene provato con 2 stringhe forma differente. La seconda, usata solo se la prima fallisce, prevede però una modifica del padding di ogni scrittore e quindi, al termine del test dell'input, il valori degli scrittori vengono riportati quelli calcolato alla loro creazione.

Il tool termina la sua esecuzione nel caso in cui l'exploit, con l'input in test, abbia avuto successo o non ve ne siano più di testabili.

1.2.6 Costruzione delle stringhe per la ricerca nello stack

Per determinare la posizione dell'input, il tool inizializza la stringa con un pattern particolare, che andrà poi a ricercare all'interno dello stack.

Al pattern da ricercare accoda poi un numero variabile di *"%N$p"* (per N >= 0), che per semplicità chiameremo "stampatori", fintanto che la loro dimensione in caratteri, sommata al pattern da ricercare, non supera la dimensione massima stampata dell'input meno la dimensione dei marcatori.

Poiché, nel caso in cui l'input sia passato da linea di comando, la sua posizione nello stack è influenzata dalla dimensione dell'input passato, è necessario fare in modo che tutte le stringhe associate all'input durante le varie iterazioni della ricerca abbiano tutte la stessa dimensione. Di conseguenza, agli stampatori verranno accodati il numero di caratteri necessario a raggiungere la dimensione massima stampata dell'input meno la dimensione dei marcatori. Per semplicità, il padding viene aggiunto anche se l'input viene passato durante l'esecuzione.

Il numero di stampatori e la dimensione del padding vengono ricalcolati ad ogni iterazione, in quanto in ognuna di queste cambia la posizione *N* stampata dallo stampatore: ad ogni nuova iterazione della ricerca, il tool procederà a generare una nuova serie di stampatori, che estrarranno i valori dall'ultima posizione stampato dall'intervallo precedente (o dalla posizione 1, nel caso si tratti della prima iterazione) fino a che non si raggiunge la dimensione massima stampabile. Supponendo quindi che l'intervallo di posizioni stampato dagli stampatori nell'iterazione *n* fosse [10 - 19], in *n+1* sarà [19-28] (sempre supponendo di rispettare il vincolo sulla dimensione massima). Ogni intervallo conterrà sempre l'ultima posizione stampata dall'intervallo precedente, per permettere l'identificazione del pattern nel caso in cui sia spezzato tra due entry successive dello stack.

Infine, la stringa ottenuta verrà incapsulata all'interno dei due marcatori, per facilitarne l'estrazione dalla risposta.

Quindi la struttura della stringa di ricerca può essere schematizzata come segue:

marcatore sinistro + pattern da cercare + stampatori + eventuale padding + marcatore destro

Durante la ricerca, il tool può incontrare due casi particolari, che impongono un cambiamento sul conteggio degli stampatori da accodare nella stringa:

1. Il programma target ritorna un segmentation fault. In questo caso il numero di stampatori viene limitato a 2, e le loro posizioni ripartono dalla prima posizione stampata dagli stampatori della stringa precedente (che ha generato il segmentation fault), finché non si verifica il caso (2), il pattern viene trovato e verificato o si raggiunge nuovamente un segmentation fault. Se si verifica quest'ultimo caso, il tool procede a scartare l'input corrente;

2. Nella risposta ottenuta dal programma target non sono presenti entrambi i marcatori. Questo si può verificare, ad esempio, quando l'input è passato da linea di comando ed è copiato in un'altra variabile tramite *sprintf()*, la quale viene poi stampata tramite *printf()*. In questo caso particolare, poiché *argv[]* viene "interpretato" e poi salvato nella variabile di destinazione, lo scrittore *"%1$p"*, ad esempio, non avrà più dimensione 4, ma pari al valore esadecimale estratto incrementato di due (in quanto *%p* antepone *0x* al valore stampato). Di conseguenza il tool assume che ogni scrittore abbia dimensione 10 o 18, a seconda dell'indirizzamento di target.

   Il tool procede con la nuova forma finché il pattern non viene trovato e verificato, si verifica il caso (2) o si incontra una nuova risposta in cui non sono presenti entrambi i marcatori. Se si verifica quest'ultimo caso, il tool procede a scartare l'input corrente.

Indifferentemente dal tipo di stringa utilizzata, nel caso in cui il pattern non sia ancora stato trovato e non sia possibile stampare la posizione successiva dello stack, in quanto il nuovo stampatore porterebbe a superare la dimensione massima stampata, l'input viene scartato.

1.2.7 Ricerca nello stack

Vediamo ora come funziona l'algoritmo di ricerca all'interno dello stack:

1. l tool assegna come pattern da ricercare la stringa *"AAAA"*, nel caso di indirizzamento a 32 bit, o *"AAAAAAAA"*, nel caso sia di 64, che saranno quindi presenti nello stack, estraendoli come valori esadecimali, come una serie di 4 o 8 *41*;

2. Il tool genera la stringa di ricerca, secondo quanto specificato sopra, la assegna all'input in test, esegue il programma target e ne salva la risposta;

3. Dalla risposta ottenuta il tool estrae la sottostringa con estremi i due marcatori ed esegue una serie di conversioni.

   Supponendo che la stringa usata come input sia *"xyAAAAAAAA%1$p%2$p%3$p%4$pzz"*, dove *"AAAAAAAA"* è il pattern da ricercare e *"xy"* e *"zz"* i due marcatori, un esempio di stringa estratta dalla risposta potrebbe essere *"AAAAAAAA0x11a0xcbbbdf0x1(nil)"*. Il tool procede quindi a sostituire tutte le istanze di *"(nil)"* con *"0x0"* e, dalla nuova stringa, ad estrarre tutti i valori esadecimali (riconosciuti tramite lo "0x" che li precede), che sarebbero, in questo caso, *11a, cbbbdf, 1, 0*.

4. Il tool ricerca il pattern all'interno della serie di valori estratta, verificando se:
    1. Il valore *n* corrisponde esattamente al pattern cercato. Se trovato, ritorna la posizione stampata dallo stampatore che ha ritornato *n*;
    2. Il valore *n* o il valore *n+1* contengono delle istanze di *00*:
        1. Se non ve ne sono, il tool conta le istanze di *41* contenute nel valore *n* e nel valore *n+1* e verifica se corrisponde al numero di *41* del pattern cercato. Se il conteggio corrisponde, ritorna la posizione stampata dallo stampatore che ha ritornato *n*;
        2. Se ve ne sono:
            - Se sono presenti nel valore *n*, il tool conta solo le istanze di *41*, contenute nel valore *n*, poste a sinistra dell'istanza di *00*;
            - Se sono presenti nel valore *n+1*, il tool conta solo le istanze di *41*, contenute nel valore *n*+1, poste a sinistra dell'istanza di *00*.

            Se la somma tra i due conteggi , ritorna la posizione stampata dallo stampatore che ha ritornato *n*.

            Questa verifica è stata aggiunta per gestire il caso particolare in cui, ad esempio, il programma target accetti due input da linea di comando, il primo vincolato ad avere valore *"AAAA"*. Poiché gli input da linea di comando, insieme al nome dell'eseguibile, vengono salvati sullo stack come un'unica stringa e separati da *00*, il conteggio delle istanza di *41* potrebbe risultare sbagliato, in quanto, nel valore *n* potrebbero essere presenti dei *41* relativi all'input vincolato *"AAAA"* (questo passo è stato verificato solo con processori Little Endian).
    3. Se il tool non trova il pattern cercato, procede a reiterare i punti (2), (3) e (4) fino a quando il pattern non viene trovato, o si verifica una delle condizioni che porta a scartare l'input descritta nella sezione di generazione delle stringhe di ricerca.
5. Trovata la posizione *n* del pattern, il tool procede a verificare la sua correttezza. Infatti potrebbe essere possibile che la stringa *"AAAAAAAA"* sia presente già nello stack, a causa del programma target stesso.

    Il tool sostituisce quindi le *"A"* del pattern con delle *"B"*, rigenera la stringa che precedentemente ha trovato il pattern e verifica se, nella serie di valori estratti dalla nuova risposta, alla posizione *n* (e *n+1*, nel caso il pattern fosse spezzato nello stack), sono presenti il numero esatto di istanze di *42* cercate.

    Nel caso il conteggio non corrisponda, l'algoritmo riparte dal punto (1), e la prima posizione estratta dal primo stampatore della nuova stringa di ricerca sarà *n+1*.

    Nel caso in cui il conteggio corrisponda, l'algoritmo ritorna la posizione *n* come posizione iniziale dell'input nello stack e la quantità di caratteri da anteporre al pattern affinché questo sia allineato alla posizione *n+1*, nel caso il pattern fosse spezzato, o 0, nel caso in cui il pattern fosse perfettamente allineato.

1.2.8 Generazione della stringa finale e test dell'exploit

Ottenuta la posizione iniziale dell'input, il tool procede a creare la stringa da utilizzare per l'exploit del programma target.

La forma della stringa finale è:

1. Serie di scrittori;
2. (Padding di allineamento);
3. Serie di indirizzi a cui scrivere;
4. Padding finale per rispettare la dimensione degli input, nel caso in cui l'input in test sia passato da linea di comando.

Ad ogni scrittore viene assegnata ora la posizione dell'indirizzo all'interno dello stack, come stringa *"%N$"* (*N* rappresenta la posizione), specificato nel punto (3), a cui deve scrivere. La sua posizione è calcolata partendo dalla posizione iniziale dell'input nello stack, calcolata con l'algoritmo precedente, incrementata in base al numero

di caratteri della stringa che precedono l'indirizzo. Se necessario, viene poi aggiunto un padding di allineamento prima della serie di indirizzi.

Solo nel caso l'input sia passato da linea di comando, il tool somma la dimensione del marcatore sinistro e il numero di caratteri di allineamento ritornati dall'algoritmo precedente: se non corrisponde alla dimensione degli indirizzi in caratteri (4 o 8 a seconda dell'indirizzamento), significa che, durante la ricerca della posizione nello stack dell'input, anche se la stringa non avesse avuto anteposto il marcatore sinistro, il pattern sarebbe comunque stato disallineato e quindi il paddign di allineamento deve essere incrementato della dimensione del marcatore sinistro e del numero di caratteri di allineamento (affinché la serie di indirizzi sia effettivamente allineata alle posizioni calcolate). Questo non è necessario per gli input passati a run-time in quanto, sempre supponendo che siano salvati in variabili semplici (array di caratteri), sono allineati rispetto allo stack e il pattern da cercare, essendo contenibile esattamente in una singola entry e posizionato all'inizio della stringa ricerca, non sarebbe mai spezzato tra due entry consecutive.

Nel caso in cui il controllo sulla dimensione della stringa fosse attivo (proprietà **input_len_control** del file di configurazione), nel caso in cui cui la dimensione della stringa finale fosse superiore alla dimensione massima stampata dell'input (ovvero la dimensione di riferimento per la generazione delle stringhe di ricerca), il tool procede a scartare l'input.

A questo punto la stringa finale viene testata:

1. Nel caso in cui il processo termini correttamente il tool suppone che l'exploit sia andato a buon fine e riporta la risposta ricevuta dal processo, come scritto prima, e termina.
2. Nel caso in cui il processo termini con un segmentation fault il tool ritenta l'exploit modificanto la struttura stringa. Questa opzione è possibile solo se nessun indirizzo target, a prescindere da come sia passato l'input, non contenga null byte e il primo valore da scrivere (padding del primo scrittore) sia inferiore alla dimensione in caratteri della serie di indirizzi più l'eventuale padding di allineamento, ricalcolato in base alla nuova forma, che la precede. Verificato ciò, il tool ricostruisce la nuova stringa, riducendo il padding del primo scrittore della quantità di caratteri che ora lo precedono e ricalcola la nuova posizione degli indirizzi target.

   La nuova stringa può essere così schematizzata:

   1. (Padding di allineamento);
   2. Serie di indirizzi;
   3. Serie di scrittori;
   4. padding finale per rispettare la dimensione degli input, nel caso in cui l'input in test sia passato da linea di comando.

Anche qui vale quanto scritto appena sopra per il controllo sulla dimensione e l'incremento del padding di allineamento. Ovviamente, in questo caso, se il padding di allineamento viene aumentato, sarà necessario ridurre il padding del primo scrittore. Se questo non è possibile, il tentativo termina e l'input viene scartato.

La nuova stringa viene quindi testata. Nel caso in cui il processo termini correttamente il tool suppone che l'exploit sia andato a buon fine (riporta la risposta e termina), altrimenti procede a testare l'input successivo.

1.3 Test

Nella cartella *test* sono inclusi alcuni programmi di test/esempio sui quali è possibile impiegare il tool.