Sicurezza nelle Applicazioni

Definizioni introduttive

Security flaw errore di progettazione, implementazione o configurazione del software che, sotto specifiche condizioni di esecuzione, può abilitare una vulnerabilità. **Weakness** è un sinonimo, ma può riferirsi al *tipo* di errore di progettazione.

Security policy insieme di regole che indica come un sistema o un'organizzazione debba fornire servizi sicuri per proteggere risorse critiche e/o sensibili.

Vulnerabilità falla sfruttabile per violare le policy di sicurezza.

Threat circostanza o evento che ha il potenziale di colpire un sistema attraverso accesso non autorizzato, distruzione/modifica/rivalazione di informazioni, o disabilitazione di un servizio (DoS),

Exploit software o tecnica che si avvantaggia di una vulnerabilità per violare una policy di sicurezza.

Countermeasure azione, componente, procedura, tecnica che affronta o si oppone ad una minaccia/vulnerabilità eliminandola o prevendendola attraverso la minimizzazione dei disagi che può causare o riportandone gli effetti in modo che siano prese azioni correttive.

Anatomia di un attacco

Un atto intenzionale tramite il quale un entità attaccante cerca di eludere i servizi di sicurezza e violare le policy di sicurezza di un sistema. Si distinguono per:

- Tiplogia
  - **Attivo**, altera le risorse del sistema e le sue operazioni
  - **Passivo**, acquisisce informazioni dal sistema
- Point of initiation
  - **Interno**, avendo già accesso alle risorse interne
  - **Esterno**, partendo da fuori il il perimetro di sicurezza del sistema
- Method of delivery
  - **Diretto**, attacca il sistema target direttamente
  - **Indiretto**, utilizza un terzo sistema per accedere al target o amplificare l'attacco

Cataloghi di vulnerabilità note: MITRE, OWASP.

Falle di Sicurezza

- Input Validation
  - *Buffer Overflow*
  - *XSS*
  - *SQL injection*
  - *Integer overflow*
- Time and State (difetti legati a parallelismo)
  - *Type confusion*
- Code Quality
  - *Null Pointer Dereference*

Input validation

XSS

Cross site scripting, code injection, eseguire codice malevolo nel browser di un utente.

Conseguenze:

- Furto di cookies
- Keylogging
- Phishing

Tipologie:

- **Persistent XSS** Codice malevolo risiede nel DB.
  1. *Attacker* POST request con `<script>`
  2. *Victim* GET request -> *Server* 200 OK
  3. *Victim* GET request all'*Attacker* con i dati rubati
- **Reflected XSS** Codice malevolo risiede nella HTTP request.
  1. *Attacker* link con `<script>` nella HTTP request
  2. *Victim* GET request -> *Server* 200 OK
  3. *Victim* GET request all'*Attacker* con i dati rubati
- **DOM-based XSS** Vulnerabilità nel codice eseguito lato client.
  1. *Attacker* link con `<script>` nella HTTP request
  2. *Victim* GET request -> *Server* 200 OK
  3. *Victim* modifica del DOM (ad esempio aggiunta di `<script>`)
  4. *Victim* GET request all'*Attacker* con i dati rubati

Metodi di prevenzione (configurati in base al contesto, al momento del controllo e se client o server side):

- Encoding
- Validation (blacklisting, whitelisting -> rejection, sanitization)

SQL Injection

Code injection, inserimento di una query SQL attraverso input.

Conseguenze:

- Lettura/modifica dati DB
- Esecuzione operazioni admin DB
- Recuperare file DB

Metodi di prevenzione:

- Encoding (per esempio tramite librerie)

Heartbleed

OpenSSL, richiedere più bit di quanto necessario.

Conseguenze:

- Ricezione di ulteriori dati oltre a quelli richiesti

Buffer overflow

Conseguenze:

- Modifica di variabili locali / argomenti (puntatori a funzione, puntatori a strutture dati con puntatori a funzioni, puntatori a variabili)
- Modifica di return address
- Modifica di puntatore al frame precedente

Shellcode piccolo programma che utilizza syscall execve per sostituire il processo corrente con la shell. Solitamente si usa per minare servizi con privilegi di amministratore, e circondandosi di istruzioni che servono ad ammortizzare il salto. In particolare viene utilizzato per attacchi interni. Generalizza in realtà il concetto di codice malevolo (*payload*) da utilizzare dopo l'exploit/attacco.

JIT spraying compilatori JIT interpretano uno script di input generando al volo l'eseguibile, che deve per forza trovarsi su pagine eseguibili. Anche questo può essere usato per generare malware.

Prevenzione: **Canary Value** tipo tdi protezione inserito dal compilatore a compile time. Ad ogni invocazione di funzione, viene inserito questo canarino di controllo dopo il base pointer (tipicamente un valore random), cosicché se all'uscita della funzione è variato si può generare un errore con cui terminare il programma.

Address Space Layout Randomisation (ASLR) contromisura degli OS moderni, per cui alcuni indirizzi di memoria del virtual address space cambiano di esecuzione in esecuzione, perciò è più difficile che gli exploit indovinino dove andare a beccare il return address.

Executable Space Protecion l'OS può distinguere le aree di memoria col codice dalle aree di memoria coi dati (serve anche supporto hardware congiunto), realizzato tramite un bit di controllo nell'indirizzo delle page table.

Gli array java sono protetti dal buffer overflow, ma tutto il codice nativo intorno ad un programma java potrebbe esservi vulnerabile. La JVM è scritta in C++, si possono usare librerie impremente in linguaggio nativo, e il compilatore JIT può comunque rimanerne vulnerabile.

Code quality

NULL pointer dereference

Funzione MMAP: mappare indirizzi fisici nello spazio di memoria del processo.
Con MMAP si può eseguire funzioni puntate da puntatori NULL (modificando cosa significa NULL con MMAP).

Normalmente exception/crash, se l'attaccate riesce ad eseguire mmap prima e poi causare volutamente il dereference può eseguire codice arbitrario. Operazioni svolte tipicamente:

- MMAP (rimap del null a 0/indirizzo shellcode)
- syscall -> NULL pointer dereference nel kernel -> esegue ciò che sta in 0
- assegnamento di permessi di root al processo, o robe simili

Java security model

The Java security model is based on controlling the operations that a class can perform when it is loaded into a running environment. For this reason, this model is called code-centric or code-based.

Permissions

A permission is a set of permissible operations on some set of resources. Every Java class loaded into a running environment is assigned a set of permissions according to some criteria, each permission granting a specific access to a particular resource. For example, a permission can constrain the access to a database or disallow the editing of a file.

In code-based security, permissions are granted based on code characteristics, such as where the code is coming from and whether it is digitally signed (and by whom). A codebase is a URL indicating code location.

Protection Domains and Security Policies

A protection domain associates permissions with codesources. The policy currently in effect is what determines protection domains. A protection domain contains one or more codesources. It may also contain a Principal array describing who is executing the code, a classloader reference, and a permission set (java.security.PermissionCollection instance) representing a collection of Permission objects.

A security policy defines the protection domains of an environment, that is, it identifies the permissions assigned to classes from specified sources. The permissions assigned to a class by a protection domain are bound statically, when the class is loaded, or dynamically, when the executing code attempts a security-sensitive operation. Protections domains are specified in one or several policy files.

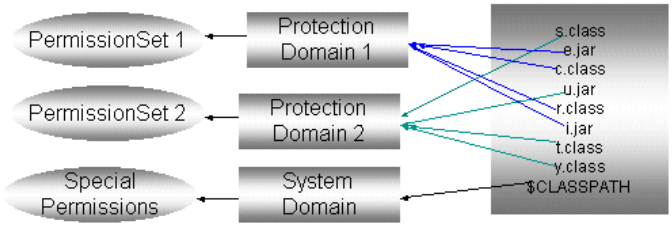

Figure 1: Associating classes with permissions through protection domains

Security Managers and Access Controllers

A security manager is the component of the Java security model that enforces the permissions granted to applications by security policies. For any security-sensitive operation that an application attempts, the security manager checks the application permissions and determines whether the operation should be allowed. The Java class java.lang.SecurityManager represents a security manager and includes several check methods to determine whether an operation should be allowed or a given permission is in effect.

An access controller is the object used by the security manager (or directly by an application, if the security manager is not enabled) to control operations and decisions. More specifically, an access controller:

- Decides whether access to a system resource should be allowed or denied, based on the current security policy in effect.

- Marks code as being privileged, thus affecting subsequent access determinations.
- Allows saving the current calling context so access-control decisions that consider the saved context can be made from other, different contexts.

The Java class java.security.AccessController represents an access controller and includes the method checkPermission(aPerm) that determines whether the access requested in the passed permission should be granted. The following code snippet illustrates the use of this method to allow reading the file /temp/testFile:

```
FilePermission perm = new FilePermission("/temp/testFile", "read");
AccessController.checkPermission(perm);
```

`checkPermission` evaluates according to the particular implementation of the access controller. The default implementation examines the entire call stack, the classes in it, and the permissions granted to those classes to determine whether to grant a request. The method returns silently if the request is granted or throws an exception if the request is denied (or the passed permission is invalid).

Gestisce gli accessi alle risorse di sistema usando Policy configurabili.
Codice caricato -> permessi assegnati in base alle Policy.

Policy il codice può accedere alla risorsa X? read, write, o entrambi?

Criteri per determinarlo:

- Codebase (URL o Path dei .class)
- Signer (sviluppatore che ha firmato il .jar)
- Principal (ruolo assegnato a runtime al codice)

Security policy file policy specificate dall'utente in uno o più file.

Il SecurityManager, tramite l'AccessController, controlla a runtime se la richiesta di accesso ad una risorsa protetta è ammessa o meno.

Default Policy

- Accedere alla CPU
- Accedere alla memoria
- Accedere al webserver da cui è stata scaricata l'applet

CodeSource identifica la provenienza di una o più classi (Codebase, Signer).

ProtectionDomain rappresenta l'insieme di tutti i permessi che una classe può avere (CodeSource, permessi assegnati dalla CodeSource).

Privilege Blocks abilita del codice trusted ad accedere temporaneamente a più risorse di quelle garantite al CodeSource.

Malware

Malware, short for malicious software, is an umbrella term used to refer to a variety of forms of hostile or intrusive software, including computer viruses, worms, trojan horses, ransomware, spyware, adware, scareware, and other

malicious programs. It can take the form of executable code, scripts, active content, and other software. Malware is defined by its malicious intent, acting against the requirements of the computer user - and so does not include software that causes unintentional harm due to some deficiency.

Programma usato per effettuare azioni dannose su un sistema informatico sfruttando debolezze di sistemi e/o utenti.

- **Trojan Horse** malicious software;
- **Virus** TrojanHorse with replication/propagation techniques;
- **Worm** Analogue to virus but *standalone*, often uses network and security flaws to spread;

Purposes and uses

- Destructive
  - Crashing the computer or device.
  - Modification or deletion of files.
  - Data corruption.
  - Block any anti-virus program.
  - Block any installation process.
  - Formatting disks, destroying all contents.
  - Spreading malware across the network.
  - Spying on user activities and access sensitive information.
  - Setting up a vulnerability for further attacks (Backdoor)
- Use of resources or identity
  - Use of the machine as part of a botnet (e.g. to perform automated spamming or to distribute Denial-of-service attacks)
  - Using computer resources for mining cryptocurrencies
  - Using the infected computer as proxy for illegal activities and/or attacks on other computers.
  - Infecting other connected devices on the network.
- Money theft, ransom (Ramsonware)
  - Electronic money theft
  - Installing ransomware such as CryptoLocker
- Data theft
  - Data theft, including for industrial espionage
  - User passwords or payment card information
  - User personally identifiable information
  - Trade secrets
- Spying, surveillance or stalking (Spyware)
  - Keystroke logging
  - Watching the user's screen
  - Viewing the user's webcam
  - Controlling the computer system remotely

Virus Signature

Una sequenza di istruzioni o parti di file utilizzato dagli antivirus per individuare software malevolo. Ovviamente questi ultimi si sono evoluti e sono state sviluppati numerose tecniche per evitare l'individuazione:

Polimorfi

Si modificano in fase di infezione, nel caso piu semplice inserendo istruzioni `nop`.

Cryptographic

Contengono chiave di crittazione, routine di decrypt e codice cryptato. Ovviamente la routine di decrypt può essere utilizzata per discriminare il software.

Metamorphic

Modifica tutto il corpo del software, incluse le routine di decrypt/encrypt.

Trojan Horse

Trojan horse, or Trojan, is any malicious computer program which is used to hack into a computer by misleading users of its true intent.

Trojans are generally spread by some form of social engineering, for example where a user is duped into executing an e-mail attachment disguised to be unsuspicious, (e.g., a routine form to be filled in), or by drive-by download or from spam links and fake pop up & Advertisement. Although their payload can be anything, many modern forms act as a backdoor, contacting a controller which can then have unauthorized access to the affected computer. Trojans may allow an attacker to access users' personal information such as banking information, passwords, or personal identity (IP address). Also, Ransomware attacks—which blocks access to data or threatens to publish it until a ransom is paid—are usually carried out using a Trojan.

Unlike computer viruses and worms, Trojans generally *do not* attempt to inject themselves into other files or otherwise propagate themselves.

Virus

Composto da un *payload*, software esecutivo, analogo alla descrizione del TrojanHorse e da un *meccanismo* di replicazione/propagazione. Può appendere se stesso ad altri possibili programmi ospiti o risiedere in macro di programmi, settore di avvio, memoria e librerie.

Worm

A computer worm is a standalone malware computer program that replicates itself in order to spread to other computers. Often, it uses a computer network to spread itself, relying on security failures on the target computer to access it. Worms almost always cause at least some harm to the network, even if only by consuming bandwidth, whereas viruses almost always corrupt or modify files on a targeted computer.

Many worms that have been created are designed only to spread, and do not attempt to change the systems they pass through. However, as the Morris worm and Mydoom showed, even these "payload-free" worms can cause major disruption by increasing network traffic and other unintended effects.

Harm

Any code designed to do more than spread the worm is typically referred to as the "payload". Typical malicious payloads might delete files on a host system (e.g., the ExploreZip worm), encrypt files in a ransomware attack, or exfiltrate data such as confidential documents or passwords.

Probably the most common payload for worms is to install a backdoor. This allows the computer to be remotely controlled by the worm author as a "zombie". Networks of such machines are often referred to as botnets and are very commonly used for a range of malicious purposes, including sending spam or performing DoS attacks.

Mitigation techniques include:

- ACLs in routers and switches
- Packet-filters
- TCP Wrapper/ACL enabled network service daemons
- Nullroute