Web Application Security: Bug Hunting e Code Review

Pisa 11 Dicembre 2008

Antonio Parata
antonio.parata@emaze.net

Francesco Ongaro
ascii@ush.it

Chi siamo?
Antonio Parata

Antonio Parata

- Lavoro nel SOC di Telecomitalia come security auditor per conto di Emaze Networks (siamo alla ricerca di personale!)

- Membro del gruppo di ricerca indipendente USH

- Faccio parte del gruppo Owasp Italy

- IT Security researcher

Chi siamo?
Francesco Ongaro

Francesco Ongaro

- IT Security consultant
- Fondatore del gruppo di ricerca indipendente USH
- IT Security researcher

Introduzione

A chi è rivolto ?

Professionisti del settore

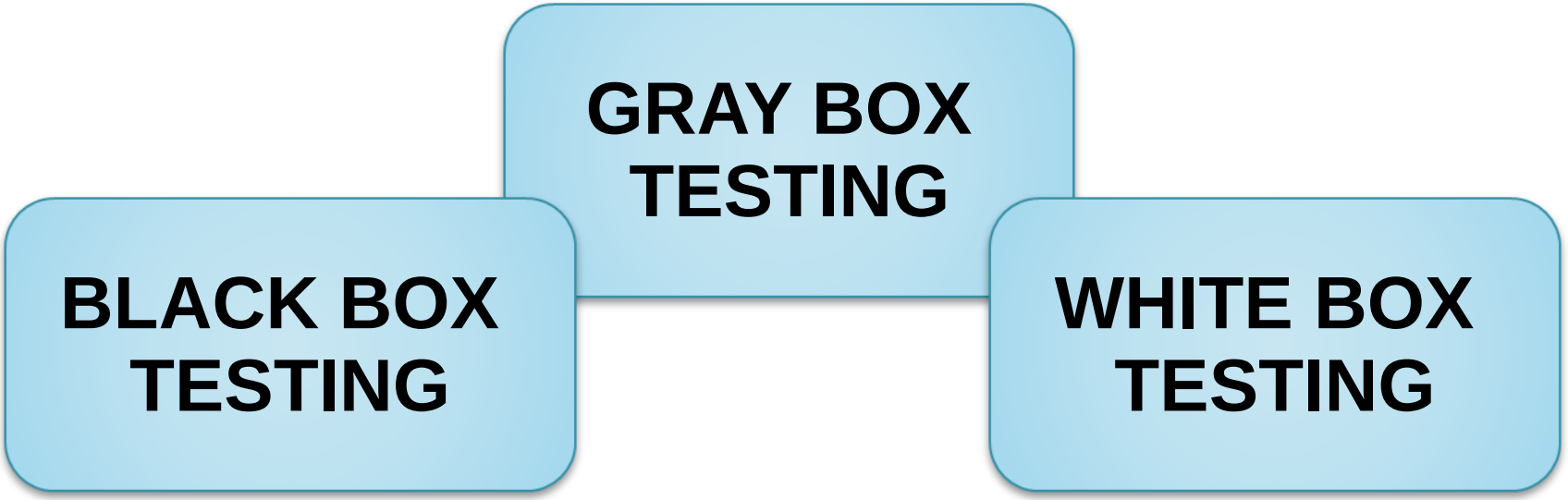
- Ethical hackers
- Source code auditors

Programmatori Web

Computer security enthusiasts

Vulnerability Assessment

È il processo che identifica, quantifica e valuta le vulnerabilità presenti in un sistema (nel nostro caso un'applicazione web).



Vulnerability Assessment

Tipologie di testing

- **Black box testing** → *Nessuna conoscenza* sul sistema

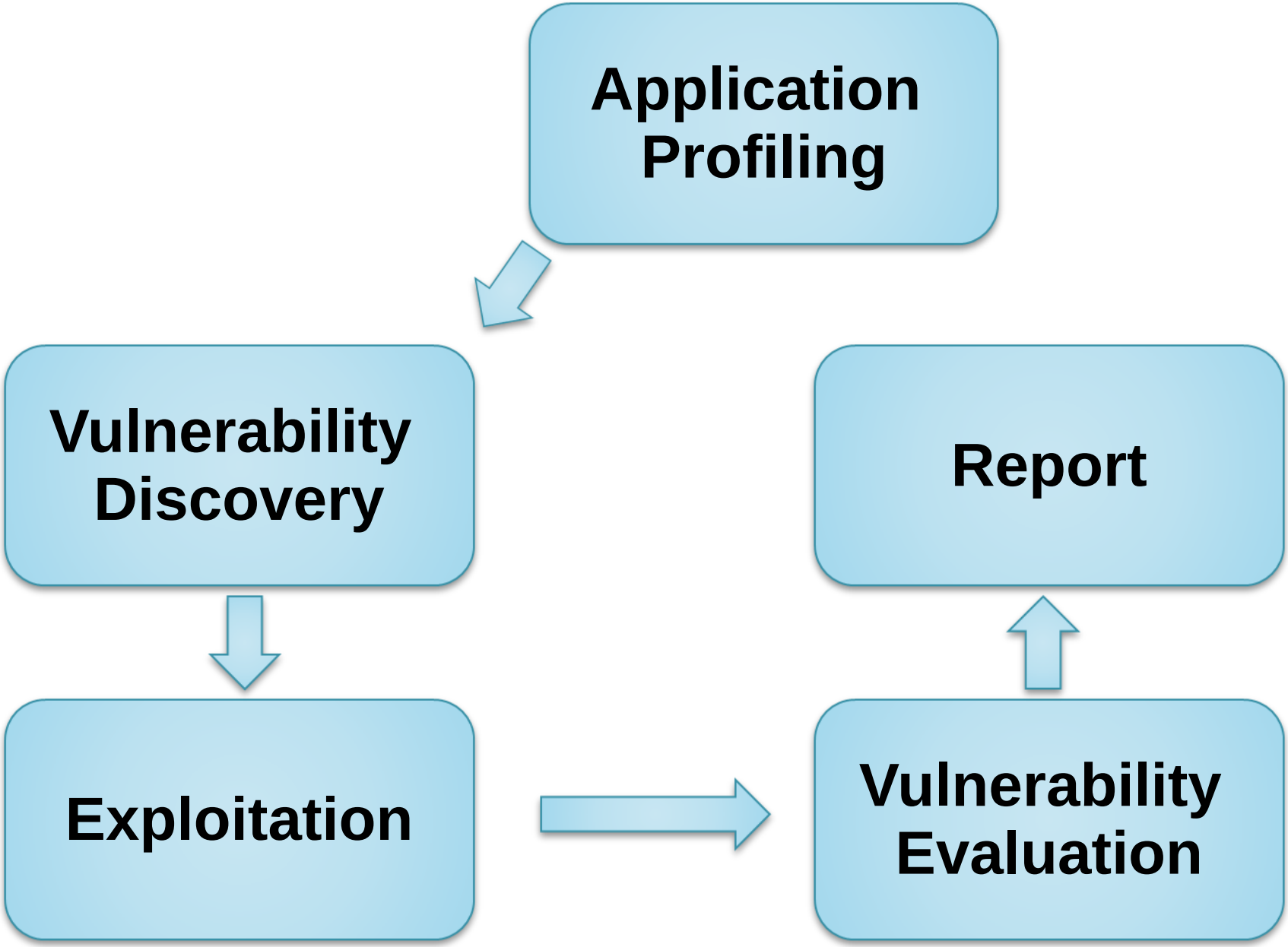
- **Gray Box Testing** → Vengono *fornite delle informazioni* sul sistema, come ad esempio tipo di applicazione, policy implementate, account di prova

- **White box testing** → Ho *pieno accesso* alle informazioni dell'applicazione, compreso il codice sorgente

BLACK BOX TESTING

Vulnerability Assessment

Penetration Testing Process



Application Profiling

Cosa identificare

Si cerca di capire con che tipo di applicazione si ha a che fare e che tecnologie utilizza:

- Tipo di applicazione (CMS, Forum, Webmail, Custom …)
- Linguaggio di programmazione (PHP, Java, ASP .NET …)
- Web Server e Application Server (Apache, IIS, Jboss, …)
- Database Server (MySQL, Acces, MS SQL Server, Oracle,…)
- Sistema operativo (Windows, Linux, …)
- Architettura (numero di tier, etc…)

Application Profiling

A che scopo fare profiling

Per identificare eventuali debolezze note:

- Forum → Stored XSS
- Webmail → Header Injection (per mail spamming)
- CMS → Arbitrary file upload (code execution)

Capire quali attacchi possono essere portati a termine a seconda del linguaggio utilizzato:

- Java, ASP .NET → difficile Code Execution, Web Server Misconfiguration
- PHP → SQLi, RCE, LFI,…

Capire che tipo di operazioni si possono fare sul DB. (eseguire codice, query annidate, stacked query, creare tabelle,…)

Vulnerability Discovery

In questa fase si identifica il maggior numero di vulnerabilità, facendo tesoro di quanto appreso nella fase di profiling per ottimizzare i tempi.

Un aiuto non indifferente viene fornito dai tool automatici:

- Dirbuster
- W3af
- Appscan (commerciale, costo circa 30K)
- Webinspect (commerciale, costo circa 30K)
- Acunetix (commerciale, costo circa 6K)

Web application vulnerabilities

1. Cross Site Scripting (XSS)
2. Injection Flaw (sqli, OS Injection, xpath injection,…)
3. Malicious File Execution (LFI,RFI)
4. Insecure Direct Object Reference
5. Cross Site Request Forgery (CSRF/XSRF)
6. Information Leakage and Improper Error Handling
7. Broken Authentication and Session Management
8. Insecure Cryptographic Storage
9. Insecure Communications
10. Failure to Restrict URL Access

OWASP TOP 10

SQL Injection vulnerability

Si presenta in quei casi in cui l'applicazione fa utilizzo di un Database come strumento di storage.

Si cerca di alterare la struttura delle query eseguite sul DB per scopi malevoli.

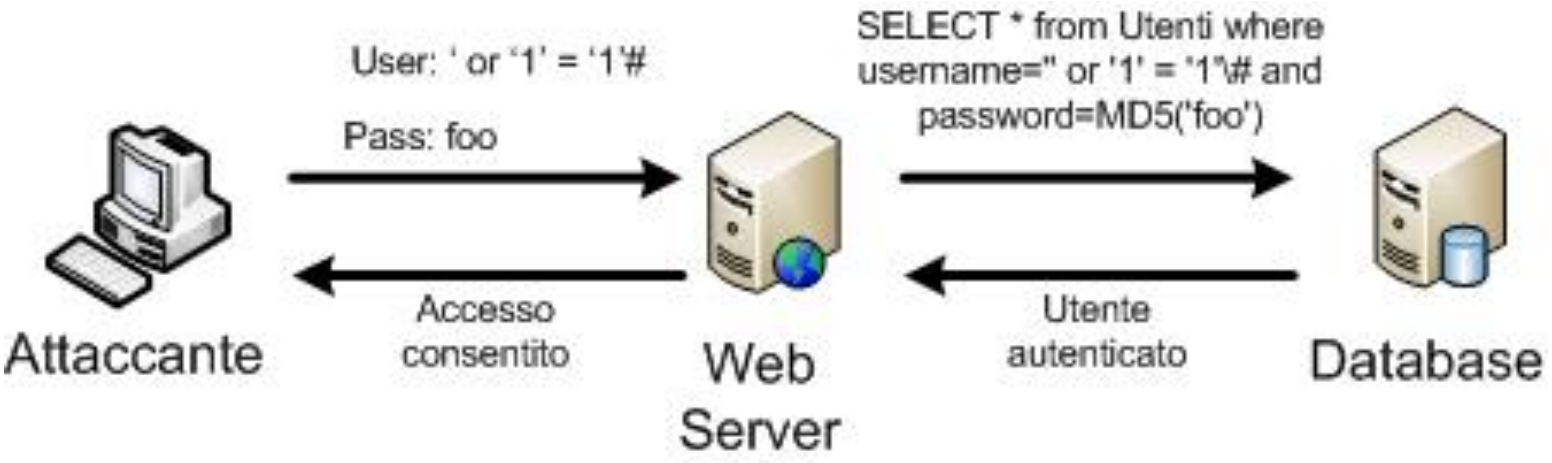
Modificando la query si può:

- Bypassare routine di autenticazione
- Ottenere username e password degli utenti dell'applicazione
- Eseguire codice sul macchina su cui è installato il DB

SQL Injection vulnerability

Query di autenticazione con DB Mysql:

*SELECT * FROM Utenti WHERE username='**<input User>**' and password=MD5('**<input Pass>**')*



SQL Injection vulnerability

Gli sviluppatori web sono ormai coscienti di questo tipo di attacco →*non più tanto comune, o per lo meno nella sua forma "standard"*

Blind SQL Injection
Second Order SQL Injection

SQL Injection vulnerability

Blind SQL Injection

In base al “tipo” di risposta ottenuta si deduce se la query è andata a buon fine o meno.

Necessita di una prima fase di scoperta della vulnerabilità:

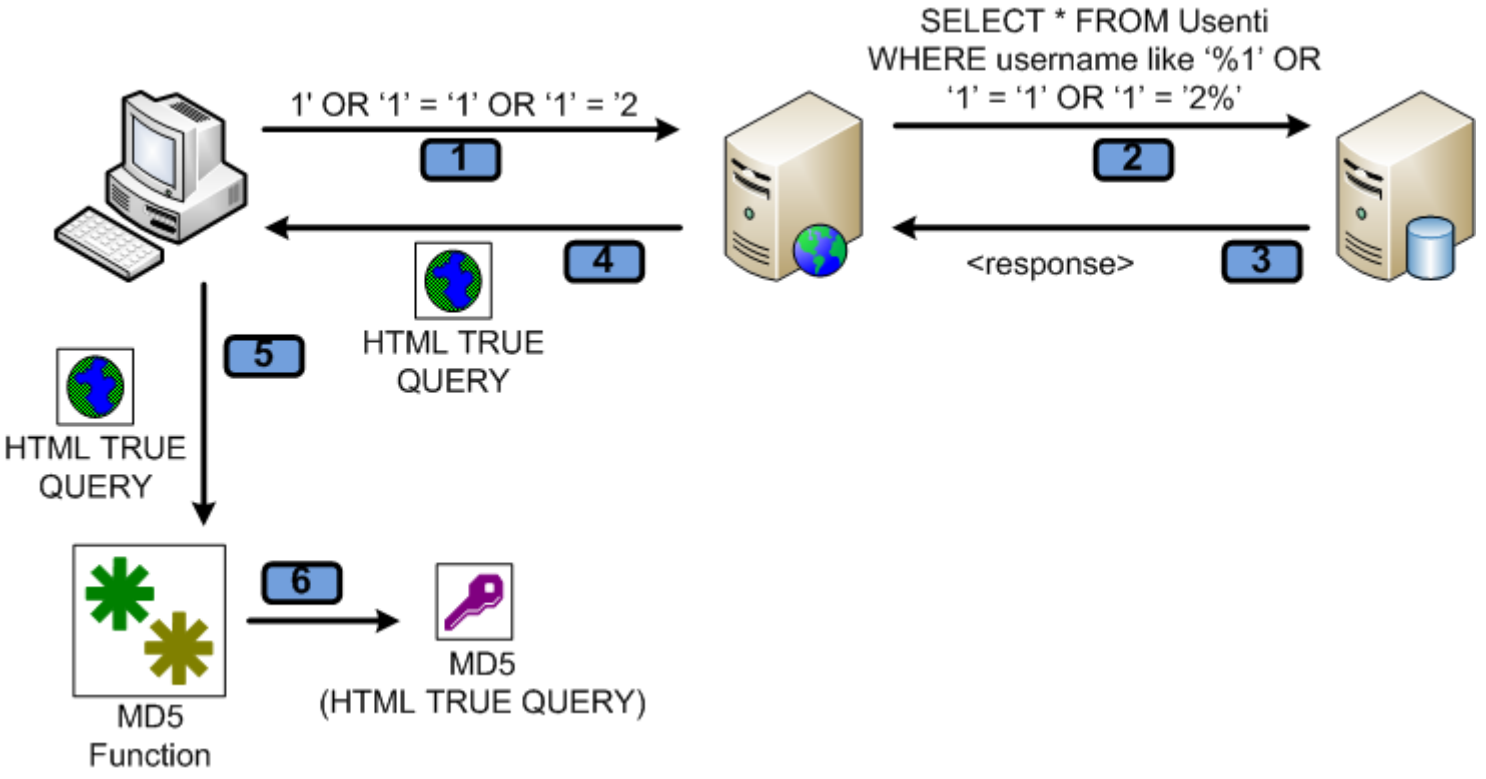
1’ OR ‘1’ = ‘1’ OR ‘1’ = ‘2 (query sempre vera)
1’ OR ‘1’ = ‘2’ OR ‘1’ = ‘2 (query sempre falsa)

Le risposte ricevute verranno utilizzate per discriminare sul risultato della query.

SQL Injection vulnerability

Blind SQL Injection

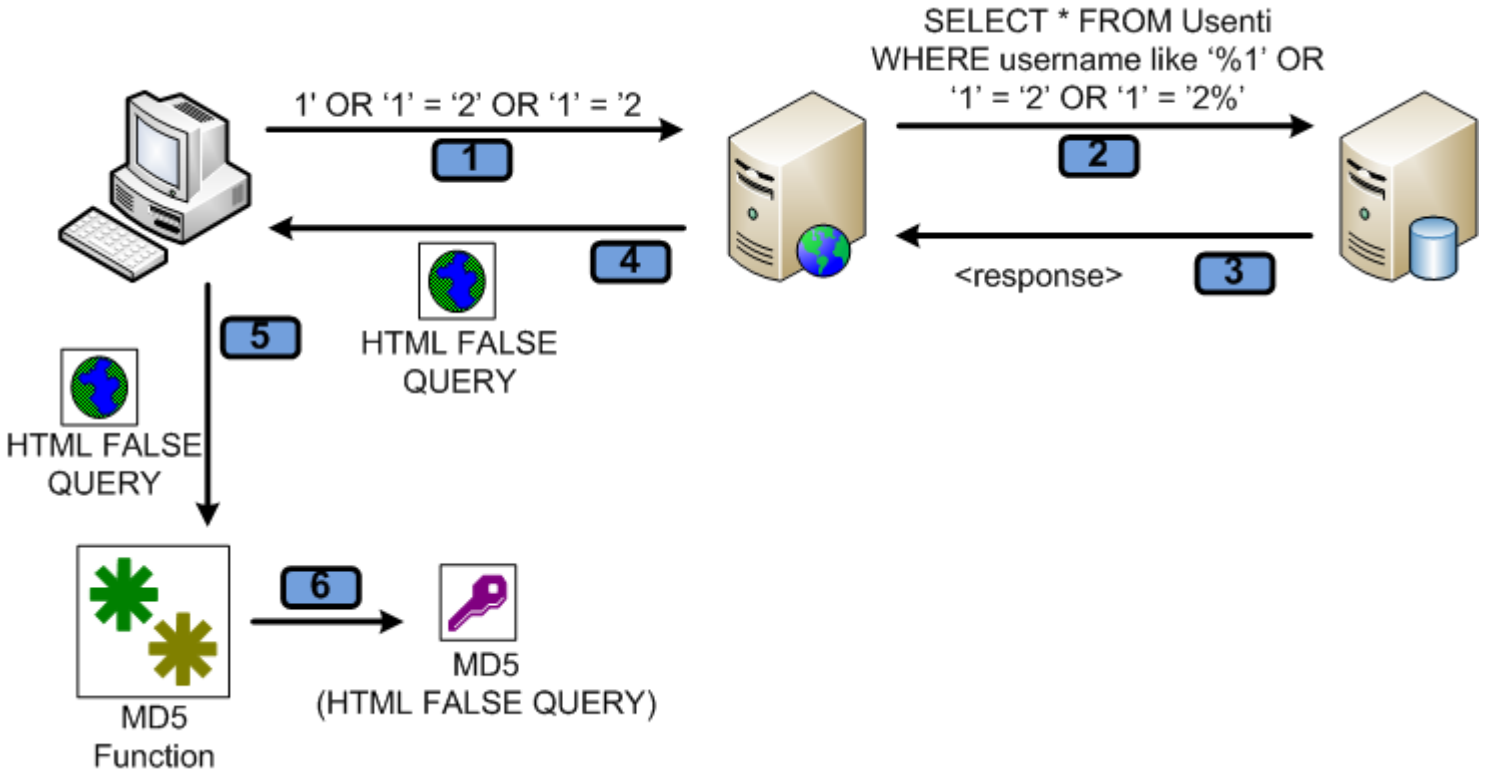
Calcoliamo il valore della risposta sempre vera:



SQL Injection vulnerability

Blind SQL Injection

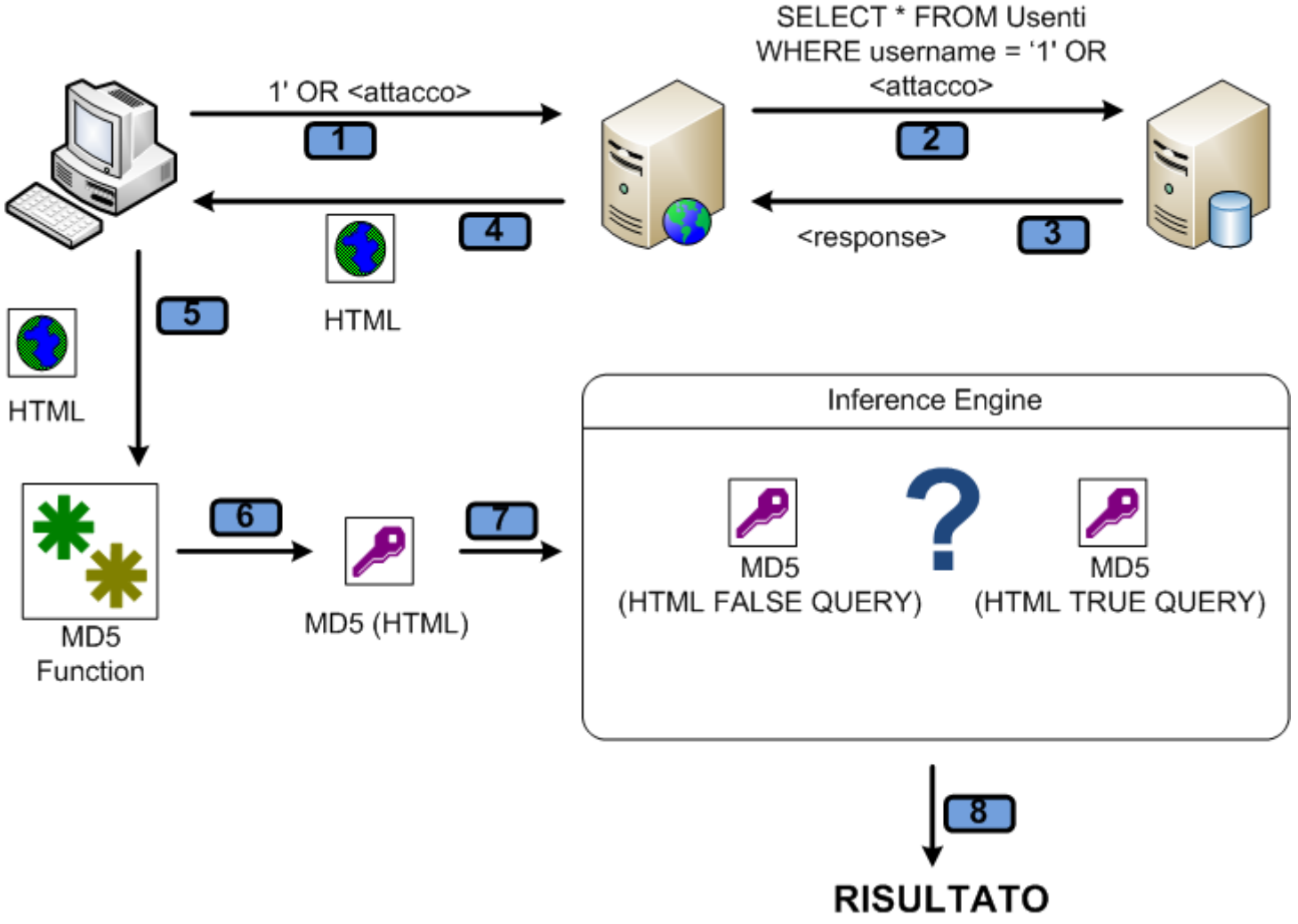
Calcoliamo il valore della risposta sempre falsa:



SQL Injection vulnerability

Blind SQL Injection

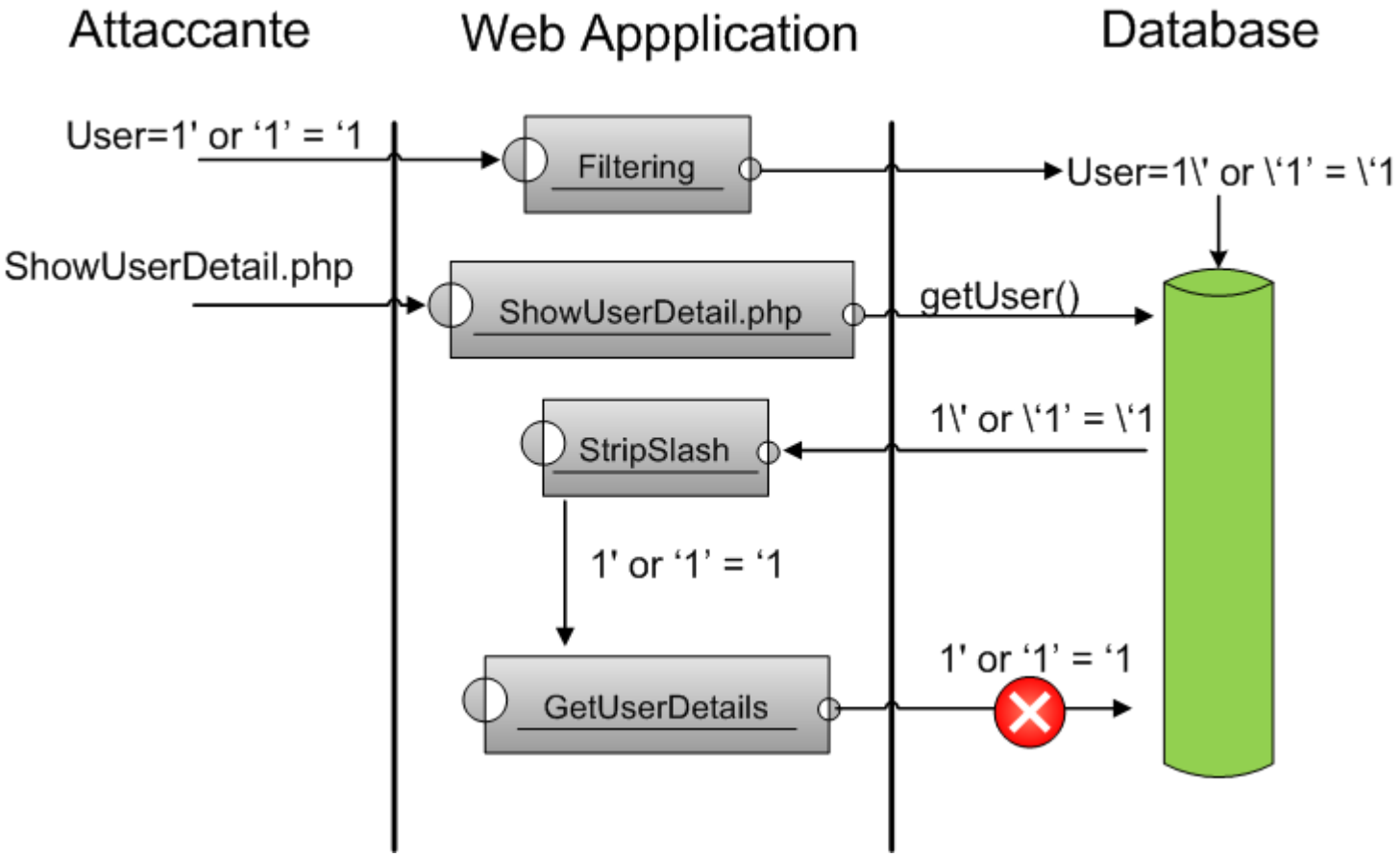
Attacco:



SQL Injection vulnerability

Second Order SQL Injection

A volte accade che l'input utente venga filtrato correttamente salvo poi essere riutilizzato in un secondo momento in modo errato:



SQL Injection vulnerability

Attack Optimization

Le vulnerabilità di tipo SQLi (in particolare di tipo Blind) richiedono un alto numero di richieste.

Esistono varie tecniche per poter ottimizzare il numero di richieste:

- Ricerca Binaria
- Mappable Blind SQL Injection
  http://www.wisec.it/sectou.php?id=4706611fe9210
  http://www.ush.it/team/ascii/hack-charmap/charmap_0.1.tar.gz

SQL Injection vulnerability

Problemi di automazione

Purtroppo esistono alcuni impedimenti ad una completa automazione degli attacchi di tipo SQLi:

- La non stabilità della pagina, attualmente rappresenta la sfida più impegnativa da parte degli sviluppatori di tool automatici (esistono comunque degli studi a riguardo non ancora pubblicati)

- Complessità delle query, in particolare la presenza di parentesi e l'utilizzo di keyword particolari (come ad esempio LIKE) complica la fase di automatizzazione dell'attacco

SQL Injection vulnerability

Tools

Tool che vale la pena di provare:

- Sqlmap http://sqlmap.sourceforge.net/
- Sqlninja http://sqlninja.sourceforge.net/
- Absinthe http://www.0x90.org/releases/absinthe/

Altri tool:

- BSQL http://labs.portcullis.co.uk/application/bsql-hacker/
- Scrawl https://download.spidynamics.com/products/scrawlr/

OS Command Injection

Utilizza funzioni che permettono di eseguire comandi di sistema o valutazione dinamica di codice di scripting:

- System
- Exec
- Eval
- Open (per script perl)
- …

Inutile dire che se dell'input utente arriva inalterato in una di queste funzioni, cose molto brutte possono accadere ☺

OS Command Injection

Considerazioni

È buona regola non far passare input utente a tali funzioni: anche in caso di escaping dell'input si possono avere brutte sorprese.

Esempio:
system('ls '.escapeshellarg($dir));

Ma se siamo su sistemi linux cosa succede se mettiamo:

$dir= `*ls*` → "`*ls*`" il codice viene eseguito ☺

Malicious File Execution

Permette di eseguire codice sul sistema.

Rappresenta una delle vulnerabilità a maggior impatto.

Tipologie:

- *Remote File Inclusion*
- *Local File Inclusion*
- *OS Command Injection*

Malicious File Execution

Remote File Inclusion

Sfrutta le funzioni di inclusione dinamica di file per eseguire codice:

- *include()*
- *include_once()*
- *require()*
- *require_once()*

Esempio (Joomla Component Simple RSS Reader 1.0):

include("$mosConfig_live_site/components/com_rssreader/about.html");

Malicious File Execution

Remote File Inclusion – Considerazioni

In declino:

- I più diffusi linguaggi di programmazione web non supportano l'inclusione dinamica. (J2EE, ASP .NET)

- PHP non permette di default di includere file remoti. (falso per le vecchie versioni di PHP)

Malicious File Execution

Local File Inclusion

Permette di visualizzare o eseguire il contenuto di file presenti sul filesystem. (permessi di lettura permettendo)

Permette di eseguire codice remoto.

Sfrutta le stesse funzioni del Remote File Inclusion.

Esempio (FTP Admin v0.1.0):

include("$page.php"); → http://localhost/ft/index.php?page=../../../../../../../../etc/passwd%00&loggedin=true

Malicious File Execution

Local File Inclusion

Non tutti sanno che su sistemi Linux è possibile eseguire del codice tramite una LFI

Esempio:
index.php?page=../../../proc/self/environ&cmd=ls
HTTP_USER_AGENT: <?php system($_GET[\'cmd\'])?>

More info:
http://www.ush.it/2008/08/18/lfi2rce-local-file-inclusion-to-remote-code-execution-advanced-exploitation-proc-shortcuts/
http://www.g-brain.net/tutorials/local-file-inclusions.txt

Failure to Restrict URL Access

Anche detta *Force Browsing***.**

Consiste nell'essere fiduciosi che chi accede a certe sezioni dell'applicazioni deve per forza essere autorizzato.

In genere ciò si ottiene:

- "Nascondendo" i link relativi a sezioni privilegiate
- Errata implementazione delle politiche di accesso

Failure to Restrict URL Access

Esempi

La pagina admin non appare nel menu principale a meno che non si è Administrator → basta far puntare il browser su /admin.php

Viene inserito un “redirect header” per redirezionare l’utente su una pagina di errore ma ci si dimentica di uscire dall’applicazione → analizzando il codice ritornato dalla pagina di redirect si ottiene il codice HTML della pagina di admin

CODE REVIEW

Code Review

Code Review è il processo di analisi del codice sorgente con il fine di identificare le eventuali vulnerabilità. (o più in generale software bugs)

Diverse tipologie di **navigazione del codice**

- **Control-flow sensitive** → si analizza il codice senza tener conto del flusso dei dati ma solo del percorso del programma.

- **Data-flow sensitive** → si analizza il codice sorgente seguendo il flusso dei dati. Scelta una variabile si considerano solo le chiamate di funzione che interessano tale variabile.

Code Review

Navigazione del codice

```
1.  int bob(int c) {
2.      if (c == 4)
3.              fred(c);
4.      if (c == 72)
5.              jim();
6.      for (; c; c)
7.              updateglobalstate();
8.  }
```

Control-flow sensitive → 123|45|67|8

Data-flow sensitive (var *c*) → 123|46|8

Code Review

Strategie di Code-auditing

Trace Malicious Input
Si identificano i punti in cui l'input utente "entra" nell'applicazione (entrypoint). Da qui si traccia l'input alla ricerca di vunerabilità

Candidate Point Strategies
Si identificano i punti in cui è più probabile che si riscontrino vulnerabilità (esecuzione di query o di comandi di sistema) e si procede a ritroso fino ad una correlazione con l'input utente

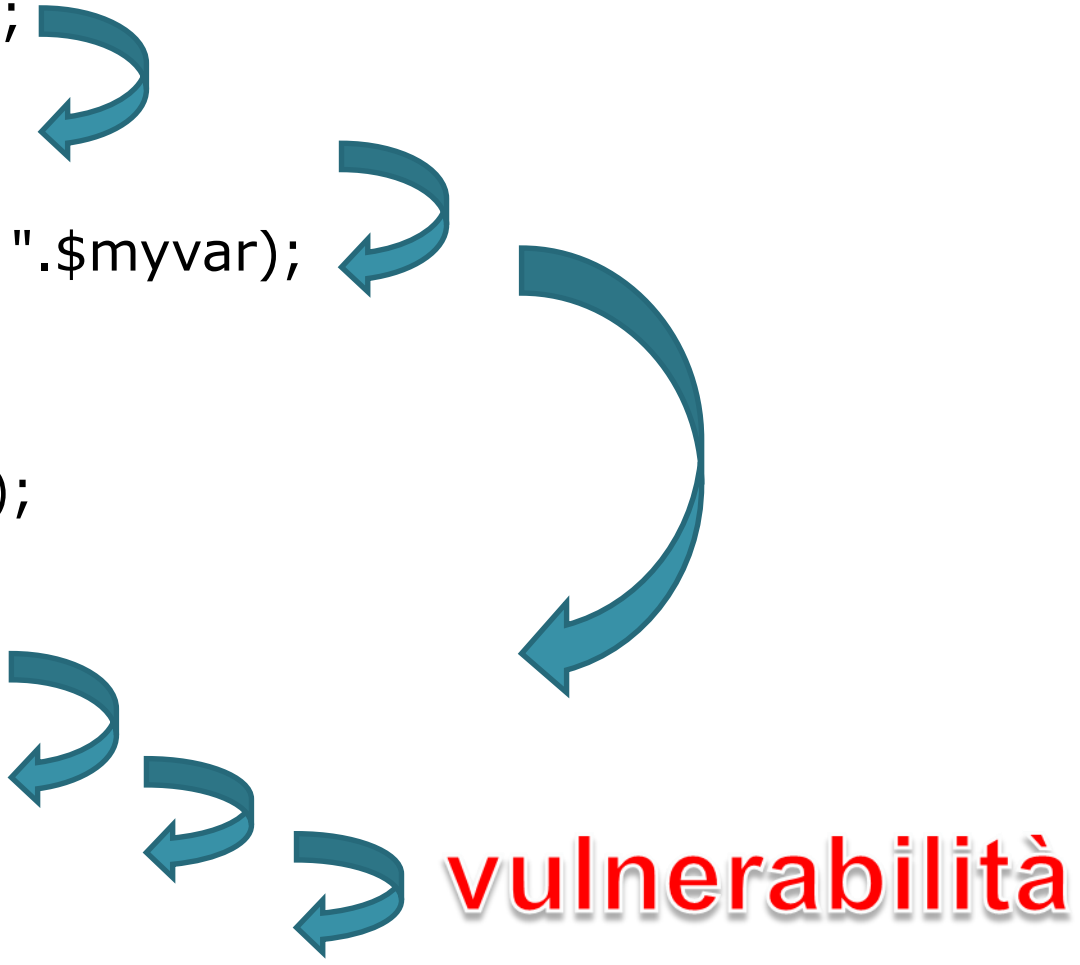
Code Review

Strategie di Code-auditing → Trace Malicious Input
Navigazione del codice → Control-Flow Sensitive

```
...
1.  $myvar = $_GET['name'];

2.  if ($counter > 0)
3.  {
4.      sayhello("Ben tornato ".$myvar);
5.  }
6.  else
7.  {
8.      sayhello("Benvenuto");
9.  }

10. function sayhello($msg)
11. {
12.     print_header();
13.     print_menu();
14.     print $msg;
15.     print_footer();
16. }
...
```



Code Review

Strategie di Code-auditing → Candidate Point Strategies
Navigazione del codice → Data-Flow Sensitive

```
...
1.    $U = $_POST['username'];
2.    $P = $_POST['password'];

3.    if (!isset($U) || !isset($P)) {
4.        echo "Password non valida";
5.    }
6.    Else {
7.        $resOk = check_login($U, $P);
8.        if ($resOk) {
9.            print "Accesso consentito";
10.           doAdminStuff();
11.       }
12.       else {
13.           print "Password e/o Username errati";
14.       }
15.   }

16.   function check_login($username, $password) {
17.       $sql = "SELECT count(*) FROM Utenti WHERE
18.           Username='$username' AND Password=MD5('$password')";
19.       $res = execute_query($sql);
20.       return $res > 0;
21.   }
...
```

vulnerabilità

Code Review

Strategie di Code-auditing – quale utilizzare?

Trace Malicious Input

Utile quando si vogliono identificare vulnerabilità derivanti da funzioni utilizzate con un'alta frequenza (print,echo → XSS)

Candidate Point Strategies

Utile quando si vogliono identificare vulnerabilità derivanti da funzioni utilizzate con bassa frequenza (system, eval → RCE). Generalmente è il primo da provare a causa del limitato uso di funzioni altamente pericoloso.

Code Review

Stato attuale della ricerca per l'automazione della Code Review

È un campo di ricerca ancora molto giovane

Si è passati da tool *"grep style"* (tool di prima generazione) a tool che comprendono un completo parser del linguaggio con complessi algoritmi di analisi:

- Data flow analysis
- Intraprocedural analysis (analisi del Call Graph)
- Interprocedural analysis (analisi dell'AST)
- Dinamic/Static taint analysis
- Symbolic execution
- <aggiungere nome altisonante qui> :P

Code Review

Tool automatici – evoluzione

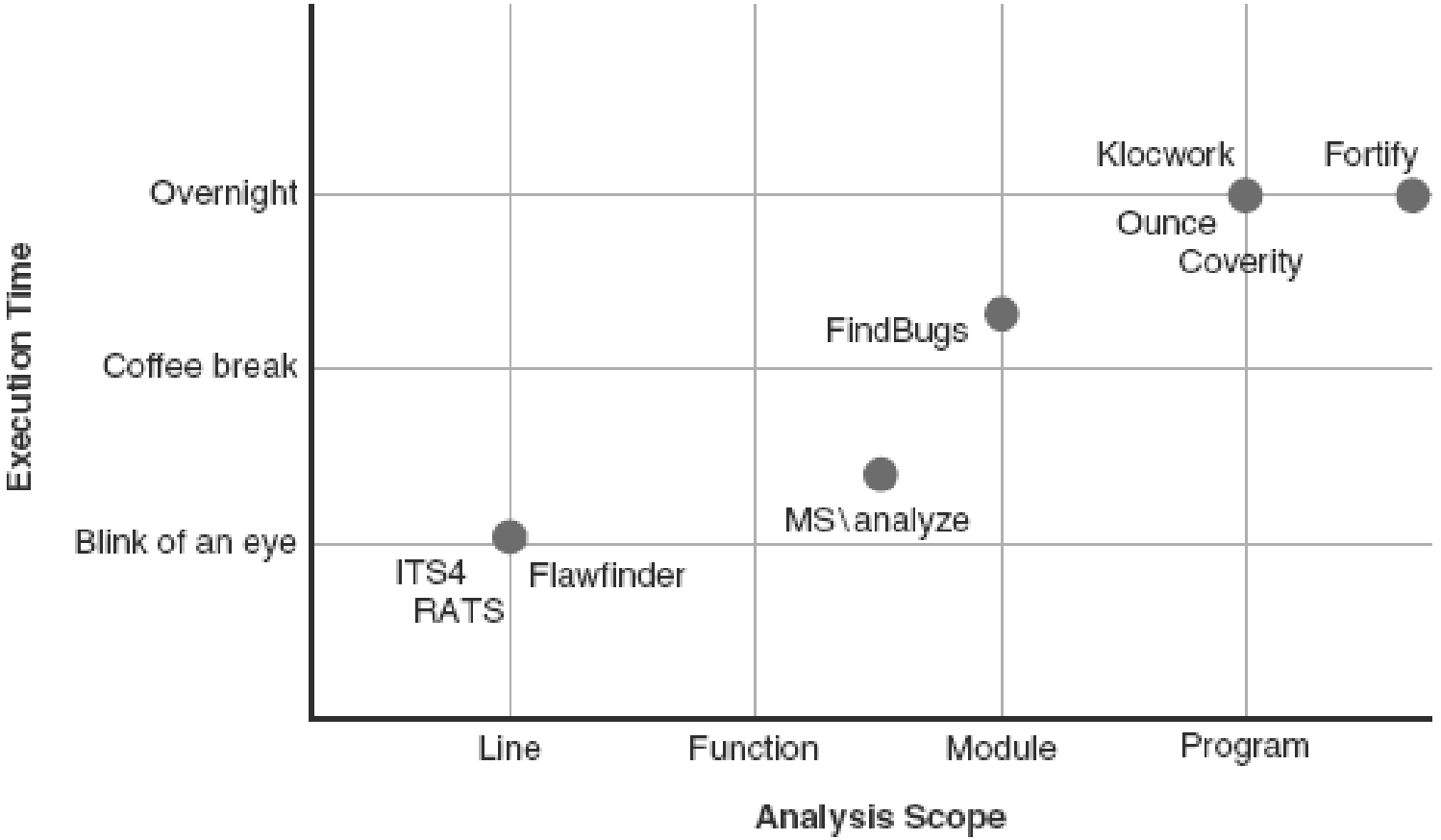


Figure 2.2 Analysis scope vs. execution time for the bug finding and security tools discussed in Section 2.1.

Fonte *"Secure Programming with Static Analysis"*

Code Review

Tool automatici – web based related

OWASP Orizon

"*This project born in 2006 in order to provide a framework to all Owasp projects developing code review services. The project is in a quite stable stage and it is usable for Java static code review and some dynamic tests against XSS. Owasp Orizon includes also APIs for code crawling, usable for code crawling tools.*"

Project Leader: Paolo Perego

Pixy

"Pixy is a Java program that performs automatic scans of PHP 4 source code, aimed at the detection of XSS and SQL injection vulnerabilities. Pixy takes a PHP program as input, and creates a report that lists possible vulnerable points in the program, together with additional information for understanding the vulnerability."

Spot the Bug

Moodle 1.9.3

La vulnerabilità è di tipo Remote Code Execution

Strategie di Code-auditing: Candidate Point Strategies

Spot the Bug

Moodle 1.9.3 - tex.php

```
...
1.  error_reporting(E_ALL);
2.  $texexp = urldecode($_SERVER['QUERY_STRING']);
3.  $texexp = str_replace('formdata=','',$texexp);
4.  if ($texexp) {
5.     $image  = md5($texexp) . ".gif";
6.     $filetype = 'image/gif';
7.     if (!file_exists("$CFG->dataroot/filter/tex")) {
8.           make_upload_directory("filter/tex");
9.     }
10.    $texexp = str_replace('&lt;','<',$texexp);
11.    $texexp = str_replace('&gt;','>',$texexp);
12.    $texexp = preg_replace('!\r\n?!',' ',$texexp);
13.    $cmd = tex_filter_get_cmd($pathname, $texexp);
14.    system($cmd, $status);
15.    if (file_exists($pathname)) {
16.          send_file($pathname, $image);
17.    } else {
18.          echo "Image not found!";
19.    }
...
```

Spot the Bug

Moodle 1.9.3 - lib.php

```
...
1.  function tex_filter_get_cmd($pathname, $texexp) {
2.     $texexp = escapeshellarg($texexp);
3.     $executable = tex_filter_get_executable(false);
4.     if ((PHP_OS == "WINNT") || (PHP_OS == "WIN32")
5.     || (PHP_OS == "Windows")) {
6.             $executable = str_replace(' ', '^ ', $executable);
7.             return "$executable ++ -e  \"$pathname\" -- $texexp";
8.     } else {
9.              return "\"$executable\" -e \"$pathname\" -- $texexp";
10.    }
11. }
...
```

Spot the Bug

Moodle 1.9.3

tex_filter_get_cmd non esegue alcuna validazione sul parametro *$pathname.*

Le globals devono essere abilitate.

Trace:

- Input formdata → $formdata
- Evil Input pathname → $pathname
- tex_filter_get_cmd($pathname,…) → $cmd
- $cmd → system($cmd,…)

Exploit:

http://www.example.com/moodle/filter/tex/texed.php?formdata=foo&pathname=foo";ls+-l;echo+"

Spot the Bug

Collabtive 0.4.6 - admin.php

La vulnerabilità è di tipo Force Browser con Authentication Bypass

Strategie di Code-auditing: Trace Malicious Input

Spot the Bug

Collabtive 0.4.6 - admin.php

```
...
1.  $repeatpass = getArrayVal($_POST, "repeatpass");
2.  $template->assign("mode", $mode);$user = new user();
3.  $project = new project();
4.  $theset = new settings();
5.  $mainclasses = array("desktop" => "desktop",
                         "profil" => "profil",
                         "admin" => "admin_active");
1.  $template->assign("mainclasses", $mainclasses);
2.  if ($adminstate < 5) {// TRUE se non siamo admin
3.     $errtxt = $langfile["nopermission"];
4.     $noperm = $langfile["accessdenied"];
5.     $template->assign("errortext", "$errtxt<br>$noperm");
6.     $template->display("error.tpl"); // è una print di codice HTML
7.  }
8.  // OK siamo admin
9.  if ($action == "index") {
10.    $classes = array("overview" => "overview_active",
                        "system" => "system",
                        "users" =>"users");
...
```

Spot the Bug

Collabtive 0.4.6 - admin.php

Se *$adminstate* è minore di 5 viene stampato un messaggio di errore, ma l'esecuzione del programma continua

Fix:
Inserire un *exit* nel blocco *if* →se le globals sono abilitate il bug rimane exploitabile

Exploit:
http://www.example.com/collabtive/admin.php

Spot the Bug

XXXX – 0day ☺

La vulnerabilità è di tipo Arbitrary file creation con Remote Code Execution

Strategie di Code-auditing: Trace Malicious Input

Prerequisiti: Nessuno!!

Spot the Bug

XXXX – 0day ☺

```
...
1.  function _holiday_cmp($a,$b){
2.    if (($year_diff = ($a['occurence'] <= 0 ?
3.          0 : $a['occurence']) - ($b['occurence'] <= 0 ?
4.          0 : $b['occurence'])))
5.          {
6.                return $year_diff;
7.          }
8.    return $a['month'] - $b['month'] ?
9.          $a['month'] - $b['month'] : $a['day'] - $b['day'];
10. }
11. $send_back_to =
    str_replace('&locale='.$_POST['locale'],'',$send_back_to);
12. $file = './holidays.'.$_POST['locale'].'.csv';
13. if(!file_exists($file) || filesize($file) < 300)       {
14.   if (count($_POST['name']))        {
15.         $fp = fopen($file,'w');
16.         if ($_POST['charset'])
17.               fwrite($fp,"charset\t".$_POST['charset']."\n");
18.         $holidays = array();
...
```

Spot the Bug

XXXX – 0day ☺

Trace:

- Input File locale → $file
- fopen($file,…) → $fp
- Evil input charset → fwrite($fp,…)

Exploit:

- Sorry no exploit evailable :P

Pubblicare una vulnerabilità

Policy

- **Full disclosure** → Trovata la vulnerabilità, *si scrive l'advisory e lo si pubblica*.

- **Responsible disclosure** → Invece di pubblicare l'advisory subito, *si avverte il vendor e si coordina con esso la pubblicazione dell'advisory dopo che è stato rilasciata una patch* (se prevista).

Responsible disclosure

Processo

- Notifica della vulnerabilità al vendor

- Aspettare ed eventualmente collaborare con il vendor per la risoluzione e il fix del bug

- Pubblicare la vulnerabilità sulle varie ml di sicurezza senza omissioni di dettagli ed eventualmente con un *poc* (proof of concept)

Esiste una policy non ufficiale

(ma riconosciuta come tale)

http://www.wiretrip.net/rfp/policy.html

Responsible disclosure

Relazionarsi con il vendor

Nella prima mail di contatto dovreste specificare:

- Versione del software vulnerabile
- Dettagliata descrizione della vulnerabilità (eventualmente allegate il draft dell'advisory)
- Suggerire una possibile soluzione
- Specificate che lo fate **solo ed esclusivamente a scopo di ricerca**
- Specificate bene qual'è la vostra politica per quanto riguarda i tempi di rilascio

Responsible disclosure

Eventuale problematiche

Può capitare che il vendor non risponda alla vostra mail, in questo caso:

- Aspettate una settimana dopodiché inviate un'altra mail di notifica, specificando che non avete ricevuto risposta (controllate se esistono altri indirizzi e-mail a cui inviare l'adv)

- Se ancora non risponde, aspettate un'altra settimana e inviate un'altra mail dicendo che il giorno seguente pubblicherete l'advisory

- Se non risponde in tempo, pubblicate l'advisory ☺

Conclusioni

- Web 2.0 is everywhere!
- Ogni azienda che si possa definire tale ha un sito web → ci si sta rendendo conto che la sicurezza delle applicazioni web è una cosa di cui tener conto.
- Scovare bug ad alto impatto (sqli, rce) non è più tanto semplice

In futuro vedremo (imho):

- Tool di analisi statica sempre più intelligenti
- Identificazione di nuove vulnerabilità lato client
- Tool di testing sempre più *Context-aware*

Domande?


Software Failure. Press left mouse button to continue.
Guru Meditation #00000004.0000AAC0