

Website gehacked?

1. Inleiding

- » De essentie: hackers zijn losers ;-)
- » Waarom? Grote misvatting: "hacken is high tech" → Fout: de meeste hackpogingen zijn niet meer dan wat copy/paste-werk van internet
- » Twee soorten hackers:
 - ★ Scriptkiddies: de overgrote meerderheid; hacken voor de fun en voor de kick (cfr. Graffiti-sputters); laten meestal duidelijke sporen na → je ziet het quasi onmiddellijk als er hackers zijn gepasseerd
 - ★ Professionals: de top; hacken om er grof geld mee te verdienen en hebben als doel een organisatie verlammen of data kopiëren; gaan heel goed voorbereid te werk en laten nauwelijks sporen na → heel moeilijk te achterhalen
- » Is hacken moeilijk? Nee, niet veel moeilijker dan Google kunnen gebruiken
- » Voorbeeld: websites met allerlei exploits van open source pakketen (<http://www.securityfocus.com>, <http://nvd.nist.gov>)
- » Voorbeeld: zoeken op Google naar "mysql_connect" +filetype:inc
- » Voorbeeld: <http://www.zone-h.org>

2. OWASP

- » OWASP staat voor "The Open Web Application Security Project"
- » Is een project (met vrijwilligers maar ook ondersteund door universiteiten, de grootste banken ter wereld, IBM, Sun Microsystems,...) dat werkt rond security op internet
- » Maakt elk jaar de "OWASP Top Ten": 10 meest voorkomende beveiligingslekken voor websites
- » Bedoeling: zorgen dat je deze ergste fouten kent vóór je aan een site begint, en regelmatig controleren of de pagina's inderdaad foutenvrij zijn (en blijven!)
- » Volledig? Nee, zeker niet, maar het is een goed begin; meer belangrijkste beveiligingslekken op de site van OWASP
- » OWASP: algemene beveiligingsproblemen; wij passen toe op/illustreren met PHP

3. A3: Malicious file execution

- » Wat?
 - ★ Het misbruiken van situaties waarbij (externe) bestanden worden ingevoegd of uitgevoerd
 - ★ Zéér goed opletten met:

- `include()`
- `include_once()`
- `require()`
- `require_once()`
- `fopen()`
- `imagecreatefromXXX()`
- `file()`
- `file_get_contents()`
- `copy()`
- `delete()`
- `unlink()`
- `upload_tmp_dir()`
- `$_FILES move_uploaded_file()`
- `system("rm -r /")`
- `eval()`
- `passthru()`

» Voorbeeld?

★ Zo niet:

```
// opvragen als A3_vb.php?pagina=home.php  
  
include ($_REQUEST["pagina"]);
```

★ Zo wel:

```
// opvragen als A3_vb.php?pagina=start  
  
if (isset($_GET["pagina"])) {  
    $pagina = $_GET["pagina"];  
} else {  
    $pagina = "";  
}  
  
switch ($pagina) {  
    case "start":  
        $echte_pagina = "home.php";  
        break;  
    case "takken":  
        $echte_pagina = "takken.php";  
        break;  
    default:  
        $echte_pagina = "home.php";  
        break;  
}  
  
include ("{$echte_pagina}");
```

★ Hoe voorkomen?

- ★ Werk indirect (bijvoorbeeld: "start" verwijst naar "home.php")
- ★ Gebruik onvoorspelbare namen/id's
- ★ Doe een expliciete controle op de opgeroepen pagina's (schrijf eigen functie)
- ★ Schakel `register_globals` en `allow_url_fopen` uit

4. A2: Injection flaws

» Wat?

- ★ Het misbruiken van gebruikersvariabelen om een commando of query te wijzigen
- ★ Hier: beperkt tot SQL injection

» Voorbeeld?

★ Zo niet:

```
$sql = "SELECT * FROM table WHERE id = '" . $_REQUEST['id'] . "'";  
  
// als een bezoeker als id de waarde 1' OR '1'='1 meegeeft,  
// dan wordt de query:  
// SELECT * FROM table WHERE id = '1' OR '1'='1'  
// of nog: àlle records worden opgevraagd (/weergegeven), in plaats  
// van enkel de record met een correcte id
```

★ Zo wel:

```
$id = (int) $_GET["id"];  
$sql = "SELECT * FROM table WHERE id = '$id'";
```

» Hoe voorkomen?

- ★ Vermijd situaties waarbij bezoekers deze queries rechtstreeks kunnen benaderen
- ★ Vermijd foutmeldingen op bezoekersniveau (zie verder)
- ★ Maak gebruik van `mysql_real_escape_string()` (beter dan `addslashes(!)`)
- ★ Gebruik PDO (PHP Data Objects): "a lightweight, consistent interface for accessing databases"

5. A6: Information leakage and improper error handling

» Wat?

- ★ Het misbruiken van een teveel aan informatie voor bezoekers
- ★ Meer bepaald: te informatieve foutmeldingen of debugberichten
- ★ Ook: functies die verschillende resultaten leveren volgens verschillende invoer (bijvoorbeeld: foutieve gebruikersnaam en juist wachtwoord of foutieve gebruikersnaam en foutief wachtwoord; in beide gevallen wordt best dezelfde foutmelding getoond)

» Hoe voorkomen?

- ★ Schakel error- en debug-berichten uit op een website die in productie is! (kan in `php.ini` maar ook via `.htaccess`)
- ★ Voorzie één exception handling systeem (enkel voor grote projecten)
- ★ Zorg voor gelijkaardige foutmeldingen bij gelijkaardige fouten (zie voorbeeld hierboven)
- ★ Zorg dat een error handler steeds een `HTTP 200 (OK)`-signaal uitstuurt (niet belangrijk voor gewone websites, wel voor die van grote firma's)

6. A8: Insecure cryptographic storage

» Wat?

- ★ Het kraken van geëncrypteerde data
- ★ Komt vaak voor bij te zwakke of eigen algoritmes (MD5, SHA-1, RC3,...)

» Voorbeeld?

```
// MD5-encryptie:
$wachtwoord = "test";
$md5 = md5($wachtwoord);
echo ("Het wachtwoord &quot;$wachtwoord&quot; omgezet naar MD5
geeft &quot;$md5&quot;");

// output:
// Het wachtwoord "test" omgezet naar MD5
// geeft "098f6bcd4621d373cade4e832627b4f6"

// MD5-encryptie met een salt:
$wachtwoord = "test";
$md5 = md5($wachtwoord . "nog iets");
echo ("Het wachtwoord &quot;$wachtwoord&quot; omgezet naar MD5
geeft &quot;$md5&quot;");

// output:
// Het wachtwoord "test" omgezet naar MD5
// geeft "daefd40e27035efca2dc234e32cedae8"
```

» Hoe voorkomen?

- ★ Maak gebruik van sterke encryptie-algoritmes (AES, SHA-256,...)
- ★ Vertrouw niet op eigen algoritmes
- ★ Bewaar enkel de persoonlijke gegevens die je echt nodig hebt of zinvol zijn

7. A1: Cross Site Scripting (XSS)

» Wat?

- ★ Eenvoudig gezegd: het door hackers invoegen van code of scripts op een site waardoor die gebruikersgegevens kan verzamelen of scripts op de achtergrond kan uitvoeren
- ★ Ook bekend als "HTML injection" of "user agent injection"

» Voorbeeld?

- ★ Reflected: klikken op een valse link, die gegevens verzamelt en je dan onmiddellijk doorstuurt naar de echte pagina (cfr. phishing: bank scams,...)
- ★ DOM: Javascript misbruiken om bezoekers te redirecten en ondertussen cookie-gegevens opslaan (XmlHttpRequest)

» Hoe voorkomen?

- ★ Zorg dat `register_globals` af staat
- ★ Maak gebruik van `$_GET`, `$_POST`, `$_SESSION` (en vermijd `$_REQUEST`)
- ★ Valideer steeds externe variabelen!

```
// zo niet:
$getal = $_GET["getal"];

// zo wel:
$getal = (int) $_GET["getal"];
```

- ★ Gebruik `htmlentities()` en `htmlspecialchars()`
 - ★ Geef geen urls mee als externe variabele, en als dat wel moet, gebruik dan `urlencode()`
 - ★ Gebruik geen "blacklist validatie" (= enkel "<" en ">" vervangen in een string om HTML-tags uit te schakelen is niet voldoende!); liever "whitelist": "dit en alleen dit is toegelaten"
 - ★ Valideer je variabelen slechts één keer ("canonicalization errors")
 - ★ Let op met open source-pakketten: upgrade indien nodig!
- » Voorbeeld?
- ★ Zo niet:

```
echo "<a href=\" http://www.mijnsite.be/?gebruikersnaam=" .
$_GET["gebruikersnaam"] . "\">link</a>";
```
 - ★ Zo wel:

```
if (isset($_GET["gebruikersnaam"])) {
    $gebruikersnaam = urlencode($_GET["gebruikersnaam"]);
} else {
    $gebruikersnaam = "standaard";
}
$link = "http://www.mijnsite.be/?gebruikersnaam={$gebruikersnaam}";
$link = htmlentities($link, ENT_QUOTES, 'UTF-8');
echo ("<a href=\"\$link\">link</a>");
```
 - ★ Demofilmpje: <http://ferruh.mavituna.com/blogs/xsstunnelling-video.zip>

8. A7: Broken authentication and session management

- » Wat?
- ★ Het opvragen en misbruiken van authenticatie- en sessiegegevens
 - ★ Vaak een gevolg van functies als logout, remember me, secret question,...
- » Hoe voorkomen?
- ★ Gebruik enkel de ingebouwde "session management"-mechanismen
 - ★ Verbied het aanpassen of gebruiken van ongeldige sessiegegevens
 - ★ Zorg voor code die niet zomaar een "remember me"-functie mogelijk maakt
 - ★ Maak het authenticatiemechanisme ingewikkeld genoeg
 - ★ Voorzie op elke pagina een duidelijke "logout"-knop en moedig bezoekers aan om hem te gebruiken
 - ★ Controleer het oude wachtwoord vooraleer een gebruiker een nieuw wachtwoord kan invoeren
 - ★ Let op met het versturen van wachtwoorden/secret questions naar e-mailadressen en voorzie zeker een beperkte bruikbaarheidsduur

9. A9: Insecure communications

- » Wat?
- ★ De "man in the middle attack" (MITM) op niet-beveiligde verbindingen
- » Hoe voorkomen?
- ★ Gebruik SSL-verbindingen voor belangrijke/persoonlijke en écht geauthenticeerde communicatie met de server

- ★ Niet echt haalbaar voor gewone websites, wel natuurlijk voor banken,...

10. A10: Failure to restrict url access

» Wat?

- ★ Het vinden van "geheime" links die onterecht toegang tot data verschaffen

» Voorbeeld?

- ★ Als <http://www.mijnsite.be/?user=quest> werkt, dan werkt <http://www.mijnsite.be/?user=admin> misschien ook?

» Hoe voorkomen?

- ★ Zorg dat elke pagina met beperkte toegang eerst de toegangsrechten controleert
- ★ Let goed op met include/library-bestanden: gebruik nooit blabla.inc, wel blabla.inc.php!
- ★ Blokkeer alle bestandstypes die je website toch nooit zullen aanbieden; werk liever met een whitelist (.htm, .html, .php, .jpg, .png, .gif, .pdf,...)

11. A4: Insecure Direct Object Reference

» Wat?

- ★ Het misbruiken van een verwijzing naar een intern geïmplementeerd object of bestand
- ★ Vrij technisch en misschien niet voor elke PHP'er van toepassing, maar kan wel zeer gevaarlijk zijn!

» Voorbeeld?

- ★ Zo niet:

```
// opvragen als "A4_vb.php?language=../../../../etc/passwd%00"
require_once($_REQUEST['language']."lang.php");
```

- ★ Zo wel:

```
// opvragen als "A4_vb.php?language=../../../../etc/passwd%00"

if (isset($_GET["language"])) {
    $language = addslashes($_GET["language"]);
} else {
    $language = "";
}

switch ($language) {
    case "nl":
        $echte_pagina = "nl.inc.php";
        break;
    case "fr":
        $echte_pagina = "fr.inc.php";
        break;
    default:
        $echte_pagina = "en.inc.php";
        break;
}

require_once("$echte_pagina");
```

- » Hoe voorkomen?
 - ★ Vermijd het blootstellen van objecten aan bezoekers
 - ★ Valideer vooraleer te verwerken en gebruik hierbij het whitelist-principe

12. A5: Cross site request forgery (CSRF)

- » Wat?
 - ★ Het opvangen en misbruiken van een openstaande sessie omdat bij acties van gebruikers enkel het sessie-id gebruikt wordt om te controleren of een gebruiker het recht heeft deze actie uit te voeren
 - ★ Ook bekend onder de naam XSRF
 - » Voorbeeld?
 - ★ Een beetje te technisch voor de scope van deze workshop ;-)
 - » Hoe voorkomen?
 - ★ Let op voor XSS vulnerabilities
 - ★ Vertrouw niet enkel op het sessie-id, maar voeg ook een random token toe aan elk formulier of elke url en controleer deze tijdelijk bruikbare token vòòr de verdere verwerking
- Voorbeeld:
- ```
<form action="/actie.php" method="post">
 <input type="hidden" name="8438927730" value="43847384383">
 ...
</form>
```
- ★ Zorg voor re-authenticatie bij het verwerken van belangrijke formulieren (bijvoorbeeld formulieren met grote admin-rechten)
  - ★ Maak gebruik van `$_POST` in plaats van `$_GET`

## 13. Tot slot

---

- » Dit document is er om te gebruiken, niet om het te misbruiken
- » Denk vooraf goed na over hoe je een webproject gaat aanpakken
- » Controleer regelmatig of alles (nog) veilig is
- » Bekijk je logs!
- » Volg beveiligingstrends op
- » Informeer je langs verschillende kanalen
- » Hulp nodig? Surf naar <http://forum.scoutnet.be!>

## 14. Referenties

---

- » <http://www.owasp.org>
- » [http://www.owasp.org/index.php/Top\\_10](http://www.owasp.org/index.php/Top_10)
- » [http://www.owasp.org/index.php/PHP\\_Top\\_5](http://www.owasp.org/index.php/PHP_Top_5)
- » <http://www.sklar.com/page/article/owasp-top-ten>
- » <http://www.php.net>