La **sécurité** en cours de **développement**

Martin Renaud
Egyde Cybersécurité

Introduction (suite)

- Qui suis-je?
  - Martin Renaud (**50%** du Saguenay et **50%** du Lac-St-Jean)
  - Consultant en Cybersécurité chez **Egyde Cybersécurité** à Québec
  - Plus de 15 ans dans le domaine des T.I.
  - Ancien développeur, maintenant Pentesteur
  - Membre d'**OWASP Québec**
  - Maîtrise en informatique : Machine Learning + Mobile Malware

Introduction (suite)

- Background professionnel comme développeur
  - Fonction : Full Stack Developer
  - Langages : Java, C#/VB/ASP.net et Python (… j'ai aussi fait du Cobol)
  - Base de données : Oracle, MySQL et MSSQL
  - Système : Linux, Windows et Mac
  - Projets : Principalement des clients lourds

- Maintenant :
  - Pentesteur
  - Évaluation des menaces et des risques
  - Révision de code
  - Chercheur en herbe au LIF (*Laboratoire d'informatique formelle à l'UQAC*)


Private zone
your name
your password
password forgotten
Login

+


SSL

=

AVANT

+

=

MAINTENANT

(C’est clair que c’est pu suffisant!)

Introduction (suite)

Mon constat

> Un manque significatif de connaissances en développement sécuritaire chez les développeurs

Principes d’une programmation sécuritaire

> La **détection des vulnérabilités**
> en cours de développement
> commence par
> les **bonnes pratiques**
> de développement **sécuritaire**

Introduction (suite)

Mon **objectif** aujourd'hui

Vous donner des **pistes**

Introduction (suite)

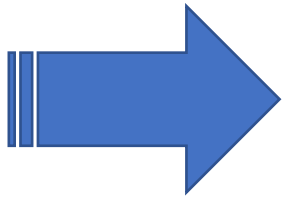
Dans le contexte de la présentation, je **devrais** parler de :

- SQLi - SQL Injection
- XSS - Cross-site Scripting / CSRF - Cross-site Request Forgery
- Unrestricted File Upload (valider frontend & backend)
- Des exemples de Input validation
- Brute force (politique de tentative)
- Créer une liste de vulnérabilités potentielles
- De Bypass de l'authentification
- Chiffrement et robustesse des « maux de passe »
- etc…

50 minutes… alors faut faire court!

Introduction (suite)

- Alors de quoi **vais** – je parler aujourd'hui?
  - Principes d'une programmation sécuritaire
  - « Connaître son langage »
  - Les frameworks
  - Quelques cas/exemples
  - Bonis?



Si on a le temps!

- Ce dont je **NE** parlerai **PAS** :
  - De Pentests applicatifs
  - De bonne configuration de serveurs Web
  - Des paradigmes de développement (DevOps, Agile, etc.)
  - Des SDLC (Systems Development Life Cycle)
  - De chiffrement des données
  - De Hack
  - … et pas de démo (mes démos plantent tout le temps!)

Principes d’une programmation sécuritaire

L'utilisation de **bonnes pratiques** de programmation sécuritaire ne dispense en aucun cas des

tests d'intrusion

Principes d’une programmation sécuritaire (suite)

- Qu’est-ce que la programmation sécuritaire?
  - C’est comprendre les **risques**
  - C’est comprendre ce qu’est une **faille** ainsi que son **impact**
  - C’est comprendre quels sont les **menaces**
  - C’est de savoir ce qu’on veut **protéger**

- C’est aussi :
  - Connaître son langage
  - Avoir une bonne « hygiène » de programmation
  - Utiliser des « *Guidelines* » ou « *Checklists* » (ex.: SCP, SWAT, etc)

Principes d’une programmation sécuritaire (suite)

OWASP - SCP (Secure Coding Practices)

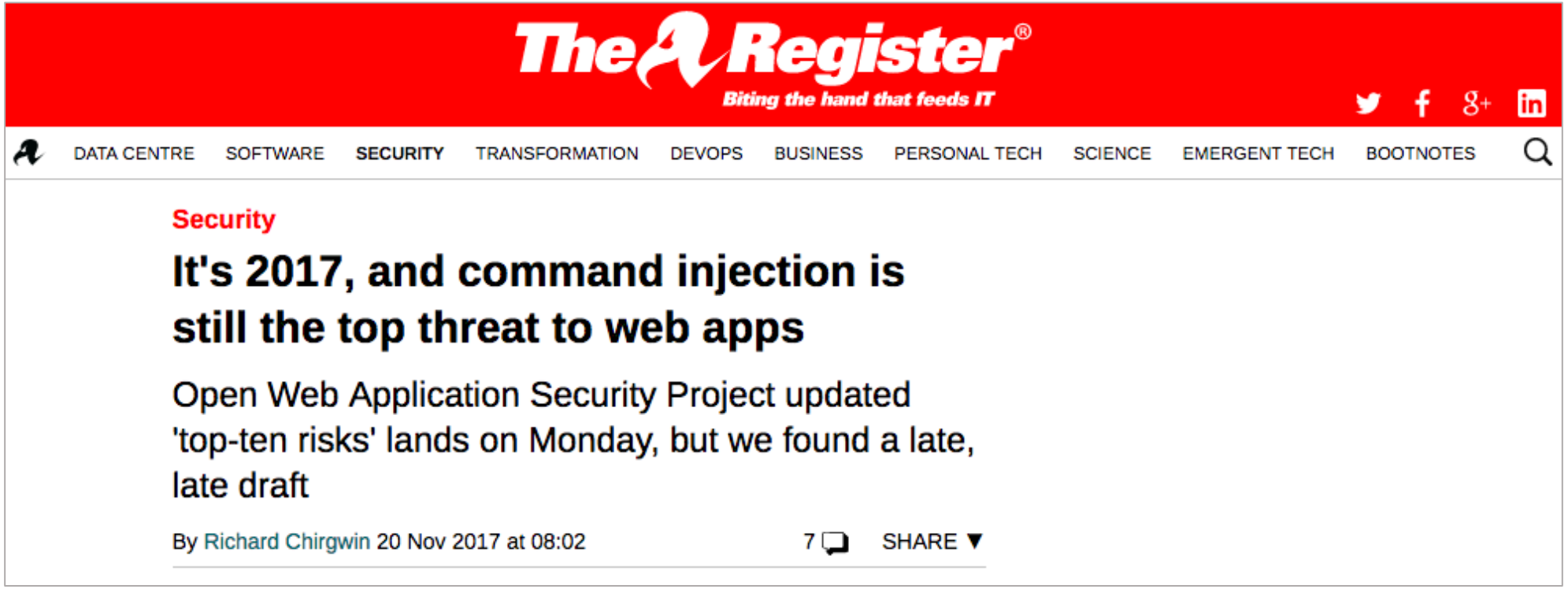
1. Validation des intrants
2. Encodage des sorties
3. Authentification et gestion des mots de passe
4. Gestion des sessions utilisateurs
5. Contrôle des accès
6. Pratiques cryptographiques
7. Gestion des erreurs/exceptions
8. Sécurité des communications
9. Configuration des systèmes
10. Sécurité de la base de données
11. Gestion des fichiers
12. Gestion de la mémoire
13. Et les bonnes pratiques de programmation

Principes d’une programmation sécuritaire (suite)

SANS - SWAT (Securing Web Application Technologies)

1. Gestion des erreurs/exceptions et des logs
2. Protection des données
3. Configuration et opération
4. Authentification
5. Gestion des sessions
6. Gestion des intrants et sortants
7. Contrôle des accès

Principes d’une programmation sécuritaire (suite)



« Nous sommes en 2017 et l'injection reste la principale menace pour les applications Web »

Principes d'une programmation sécuritaire (suite)

TOP10 OWASP

OWASP Top 10 - 2013		OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Connaître son langage

Connaître son langage de programmation,

c’est comme connaître sa langue (ex.: français, anglais, espagnol).

Plus tu la **maîtrises**,

plus tu **évites les erreurs** (aka les **bugs** et les **failles**)

(et tu deviens plus **performant**).

Il est toujours possible de faire rouler un outil d’analyse et de correction, **mais rien ne vaut la maîtrise de sa langue**.

Connaître son langage (suite)

Exemple d'une injection SQL (SQLi) sous PHP avec MySQL

```
<?php

$sql = "SELECT * FROM user_table WHERE username = " .  $_GET["user"] . "
AND password = " . $_GET["pwd"];

?>
```

On affecte au paramètre **user** : `' OR 0=0 #`

La requête SQL devient donc

```
SELECT * FROM user_table WHERE username = '' OR 0=0 # ' AND passwd = 'blah'
```

Connaître son langage (suite)

Exemple d'une requête MySQL paramétrée sous PHP

```
<?php

$stmt = $mysqli->prepare("SELECT * FROM user_table  WHERE username= ? AND password = ?");
$stmt->bind_param("ss", $_GET["user"], $_GET["pwd"]);  // s = string param
$stmt->execute();
…
$stmt->close();

?>
```

Connaître son langage (suite)


```
String query = "SELECT account_balance FROM user_data WHERE user_name = "
  + request.getParameter("customerName");

try {
       Statement statement = connection.createStatement( … );
       ResultSet results = statement.executeQuery( query );
}
```

```
String custname = request.getParameter("customerName"); // This should REALLY be validated too
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

Réf.: https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

Connaître son langage (suite)

Évasion (ou Échapement) des caractères HTML

& --> &	**<** --> <
/ --> /	**>** --> >
" --> "	**‘** --> '

Pour que ceci :

```
<SCRIPT>alert(‘Hello World’);</SCRIPT>
```

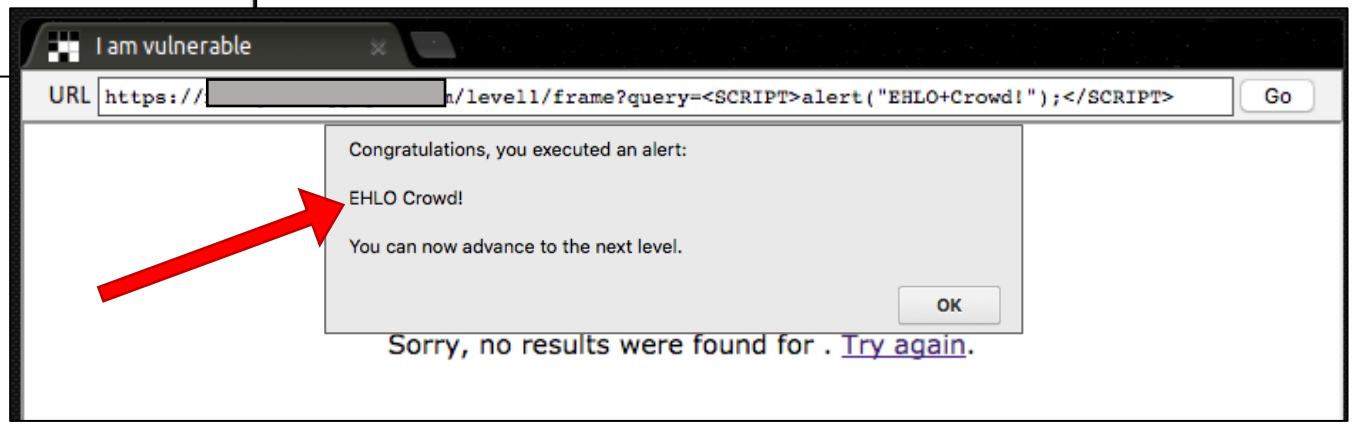
Devienne cela:

```
&lt;SCRIPT&gt;alert(&#x27;Hello World&#x27;)&lt;&#x2F;SCRIPT&gt;
```

Connaître son langage (suite)

… Afin d'éviter que cela :

Donne ceci :



Réf.: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Connaître son langage (suite)

Exemple d'évasion sous :

Python 3
`html.escape("")`

```
➜ ~ python3
Python 3.6.2 (default, Jul 17 2017, 16:44:45)
[GCC 4.2.1 Compatible Apple LLVM 8.1.0 (clang-802.0.42)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import html
>>> html.escape("<script>alert('EHLO World');</script>")
'&lt;script&gt;alert(&#x27;EHLO World&#x27;);&lt;/script&gt;'
>>>
```

PHP
`htmlspecialchars("");`

```
➜ ~ php -a
Interactive shell

php > $escp = htmlspecialchars("<script>alert('EHLO World');</script>");
php > echo $escp;
&lt;script&gt;alert('EHLO World');&lt;/script&gt;
php >
```

Réf.: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Connaître son langage (suite)

Preventing SQL injections in Python (and other vulnerabilities)

Insecure packages

When you `import` a module into your Python application, the interpreter will run the code. This means you need to be cautious when importing modules, PyPi is a wonderful resource, but the submitted code is not checked, and malicious packages have found their way into PyPi named with common misspellings. How many times have you added a package to your system without checking its content or origins.

If you're unsure as to the authenticity and content of an external package, do some research and leave it well alone if you're still uncertain.

Réf.: https://blog.sqreen.io/preventing-sql-injections-in-python/

Framework : 

> **Security** is a number of **first-class features** in Django
>
> - Un gars sur StackExchange

Possède des mécanismes de protection clairement documentés

- Cross site scripting (XSS) protection
- Cross site request forgery (CSRF) protection
- SQL injection protection
- Clickjacking protection
- SSL/HTTPS
- Host header validation
- Session security
- User-uploaded content
- …

Réf.: https://docs.djangoproject.com/en/1.11/topics/security/

Framework : 

> **Security** is a number of **first-class features** in Django
>
> - Un gars sur StackExchange

Possède des mécanismes de protection clairement documentés

- Cross site scripting (XSS) protection
- Cross site request forgery (CSRF) protection
- **SQL injection protection**
- Clickjacking protection
- SSL/HTTPS
- **Host header validation**
- Session security
- User-uploaded content
- …

Réf.: https://docs.djangoproject.com/en/1.11/topics/security/

Framework : 

SQL injection protection

SQL injection is a type of attack where a malicious user is able to execute arbitrary SQL code on a database. This can result in records being deleted or data leakage.

By using Django's querysets, the resulting SQL will be properly escaped by the underlying database driver. However, Django also gives developers power to write raw queries or execute custom sql. These capabilities should be used sparingly and you should always be careful to properly escape any parameters that the user can control. In addition, you should exercise caution when using `extra()` and `RawSQL`.

Réf.: https://docs.djangoproject.com/en/1.11/topics/security/

Framework : 

Host header validation

Django uses the **`Host`** header provided by the client to construct URLs in certain cases. While these values are sanitized to prevent Cross Site Scripting attacks, a fake **`Host`** value can be used for Cross-Site Request Forgery, cache poisoning attacks, and poisoning links in emails.

Because even seemingly-secure web server configurations are susceptible to fake **`Host`** headers, Django validates **`Host`** headers against the **`ALLOWED_HOSTS`** setting in the **`django.http.HttpRequest.get_host()`** method.

This validation only applies via **`get_host()`**; if your code accesses the **`Host`** header directly from **`request.META`** you are bypassing this security protection.

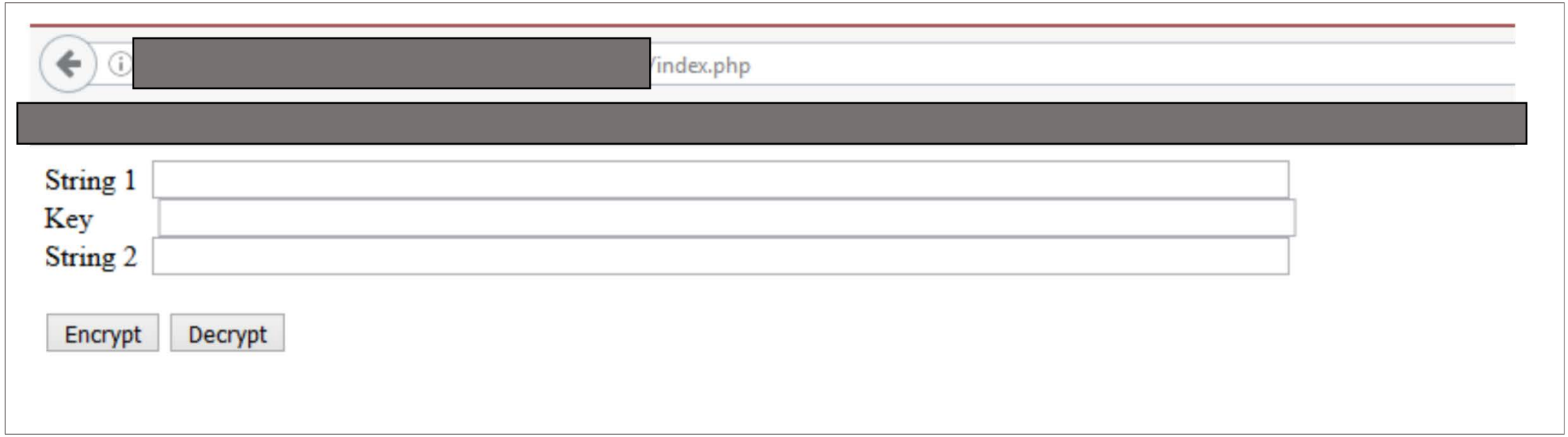
For more details see the full **`ALLOWED_HOSTS`** documentation.

Réf.: https://docs.djangoproject.com/en/1.11/topics/security/

Framework++

Quelques cas

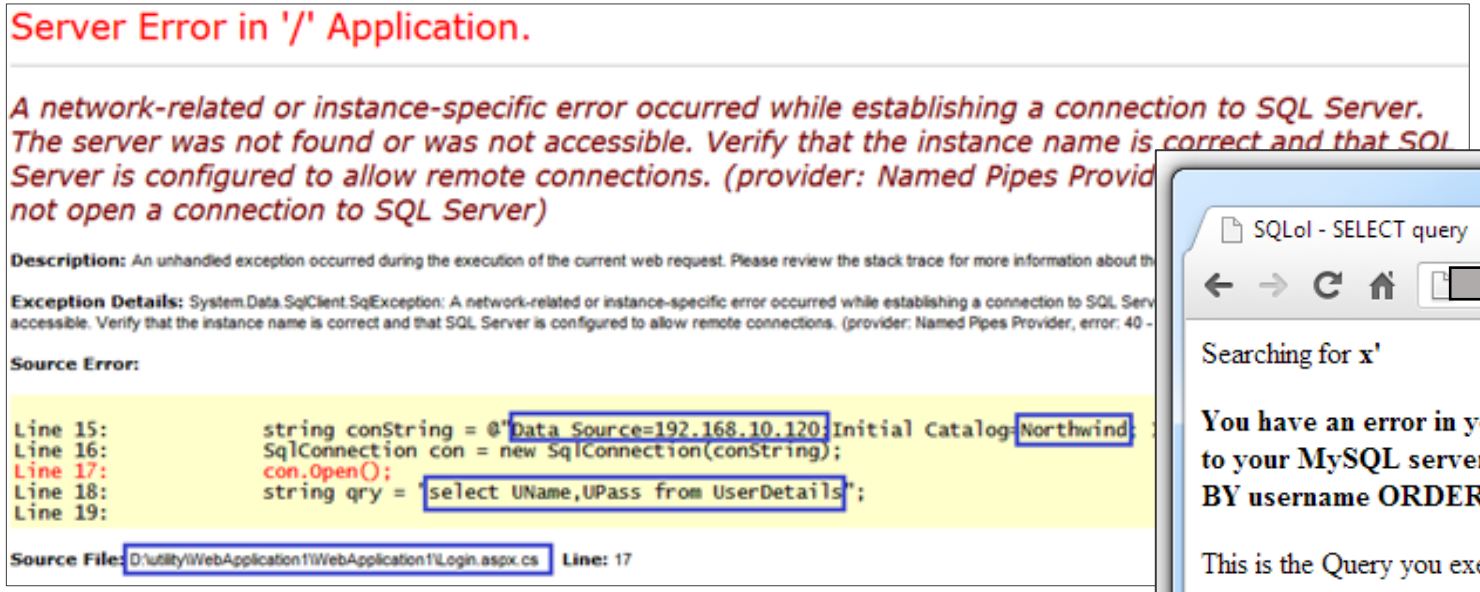
Un cas de « Chiffrement maison »…



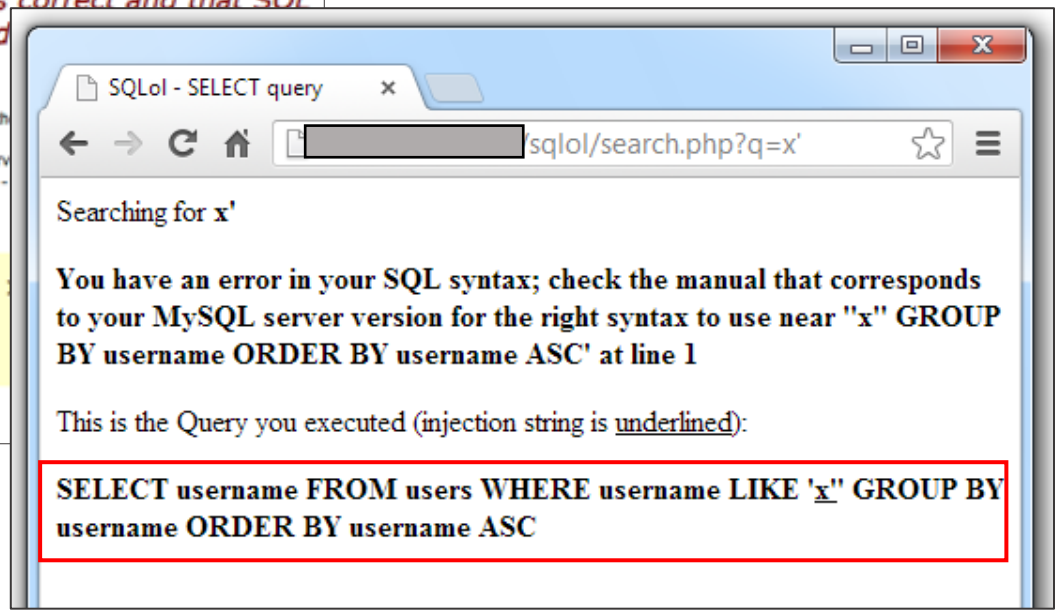
Formulaire de chiffrement/déchiffrement accessible… **publiquement**

Quelques cas (suite)

Un cas de mauvaise gestion des erreurs/exceptions



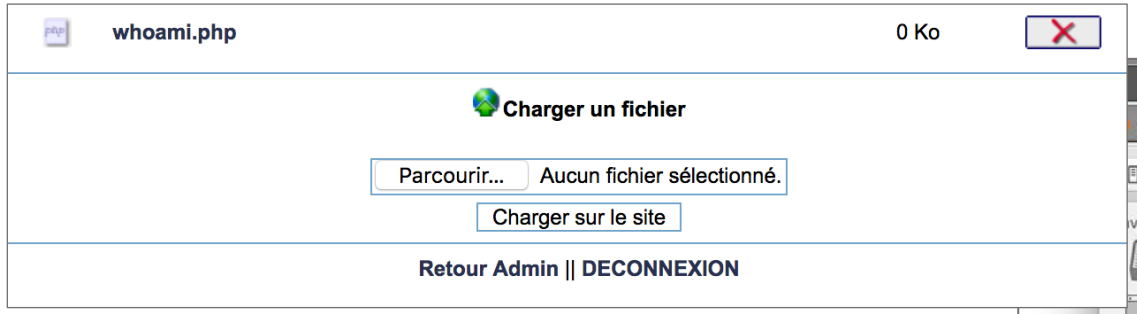
IIS + .Net + MSSQL



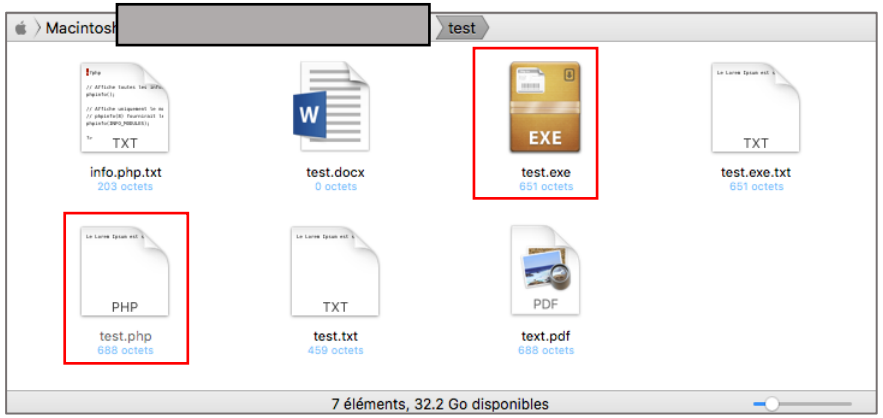
Apache + PHP + MySQL

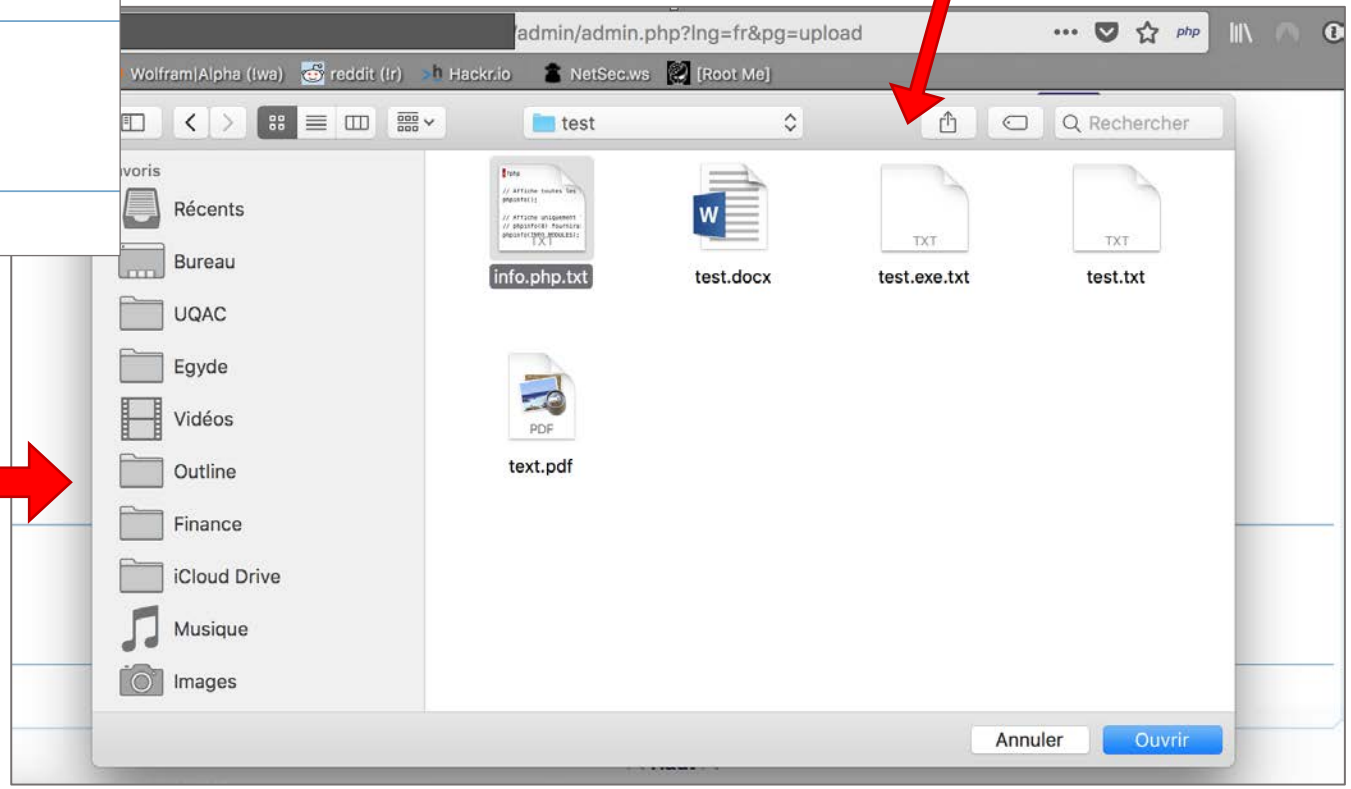
Quelques cas (suite)

Un cas d'Upload de fichier sans restriction (ou presque!)



Filtre



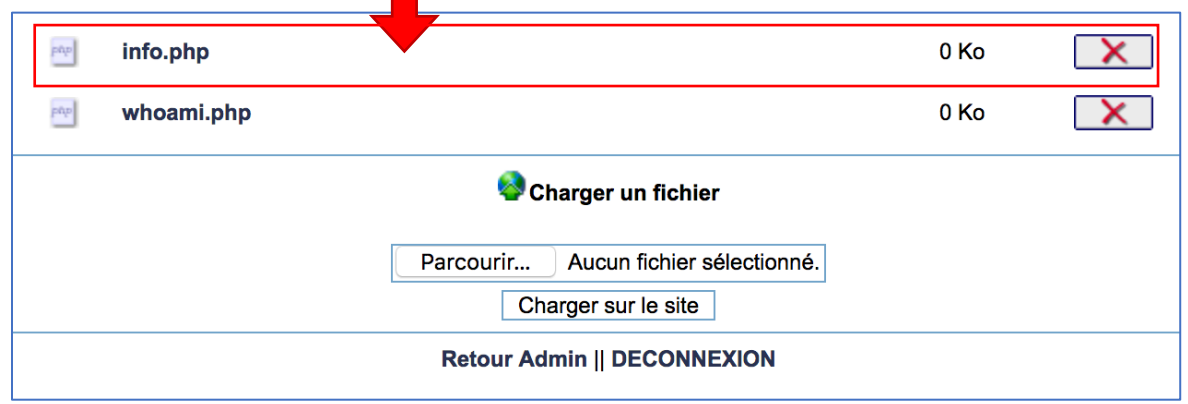


(Contenu du répertoire /test)

Quelques cas (suite)

```
POST /admin/admin.php?lng=fr HTTP/1.1
Host: www.exemple.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:57.0) Gecko/201
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://www.exemple.com/admin/admin.php?pg=upload
Content-Type: multipart/form-data; boundary=---------------------------92790
Content-Length: 772
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1

-----------------------------927908089325007309134692804 2
Content-Disposition: form-data; name="pg"

upload
-----------------------------927908089325007309134692804 2
Content-Disposition: form-data; name="rep"

file
-----------------------------927908089325007309134692804 2
Content-Disposition: form-data; name="ficup"; filename="info.php.txt"
Content-Type: text/plain

<?php

// Affiche toutes les informations, comme le ferait INFO_ALL
phpinfo();

// Affiche uniquement le module d'information.
// phpinfo(8) fournirait les mêmes informations.
phpinfo(INFO_MODULES);

?>
-----------------------------927908089325007309134692804 2
Content-Disposition: form-data; name="OK"
```

```
730913469280 42
"; filename="info.php"
```



Quelques cas++

Bonis?

OUPS! Peut-être la prochaine fois!

Quelques bonnes adresses

Article intéressant

https://motherboard.vice.com/en_us/topic/know-your-language

SANS SWAT (Checklist)

https://software-security.sans.org/resources/swat

OWASP SCP (Checklist)

https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_Checklist

https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

SEI CERT (Software Engineering Institute, Carnegie Mellon University)

https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards

Quelques bonnes adresses (suite)

PHP

http://www.phptherightway.com (excellent site de @codeguy sur Github)

http://php.net/manual/en/security.php

Python

http://www.pythonsecurity.org (un autre projet OWASP)

https://github.com/openstack/bandit (un tool)

Java

https://www.java.com/en/security/developer-info.jsp

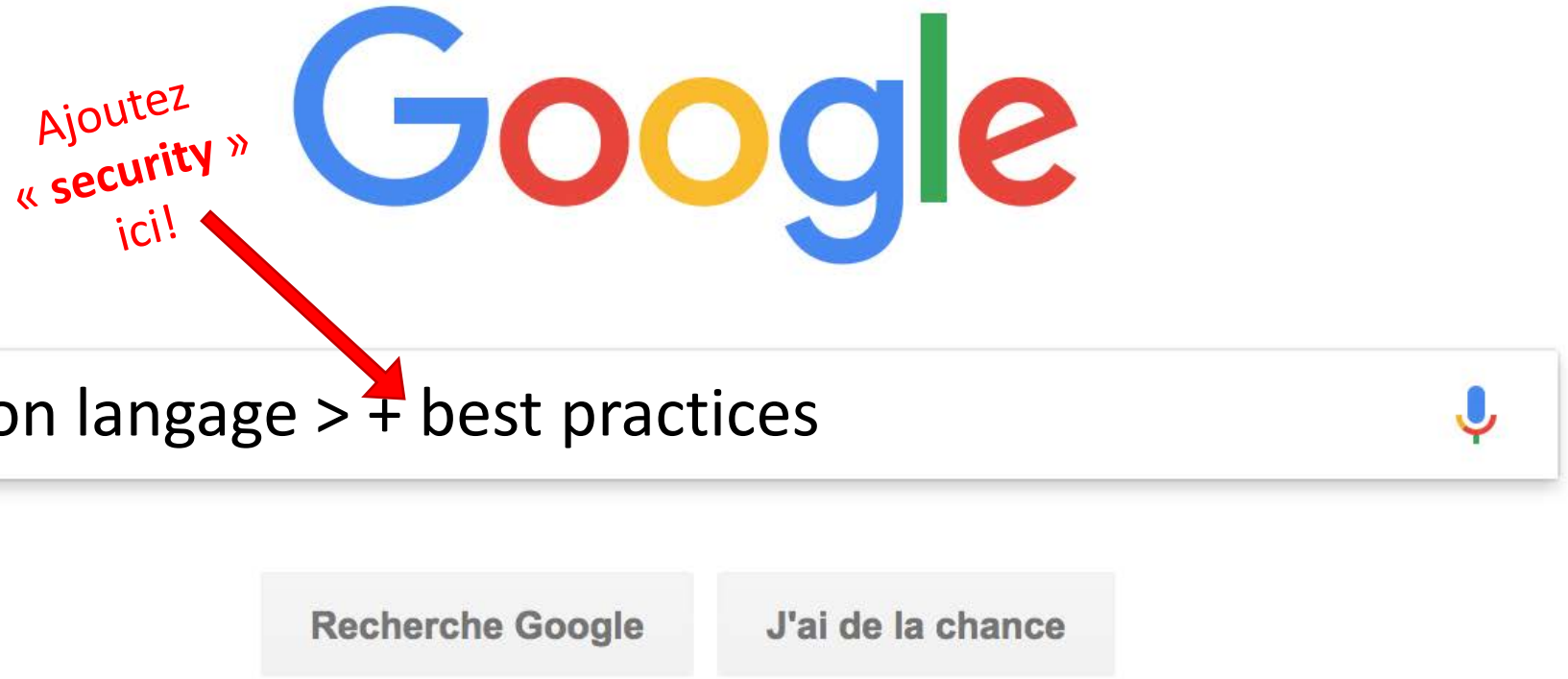
Frameworks

https://docs.djangoproject.com/fr/1.11/topics/security

https://pythonhosted.org/Flask-Security

https://symfony.com/doc/current/best_practices/security.html

Finalement… **Google est ton ami**



Conclusion

L’équation est simple

Plus la communauté de **développeurs** sera sensibilisée au développement **sécuritaire** et plus il y aura des …

Systèmes / Logiciels / Sites Web

Sécuritaires

0 + 0 = ?

Merci!

Des questions?

Egyde : www.egyde.ca
OWASP Québec : www.owasp.org/index.php/Quebec_City

Courriel : mrenaud@egyde.ca | martin.renaud1@uqac.ca
Twitter : @martinrenaud
Researchgate : @Martin_Renaud

EGYDE
CYBERSÉCURITÉ