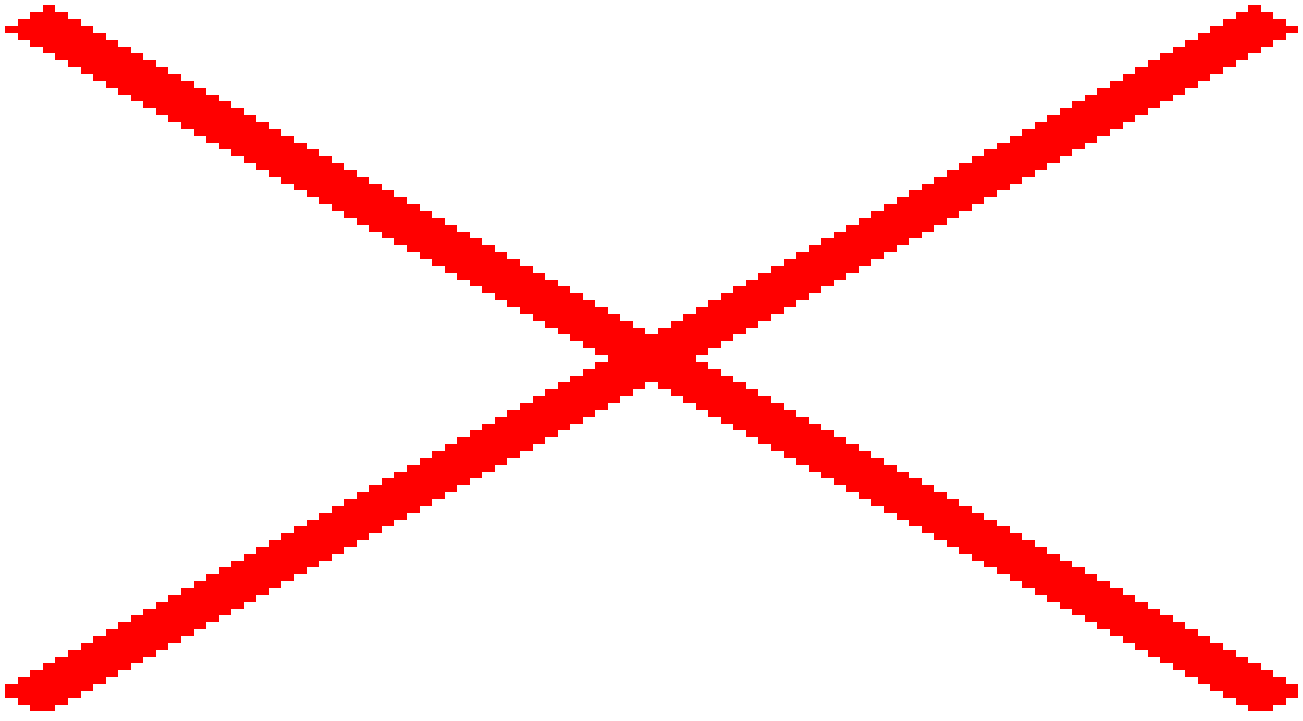
Web Application (LAMP*) Security

Attack and Defense for System Administrators



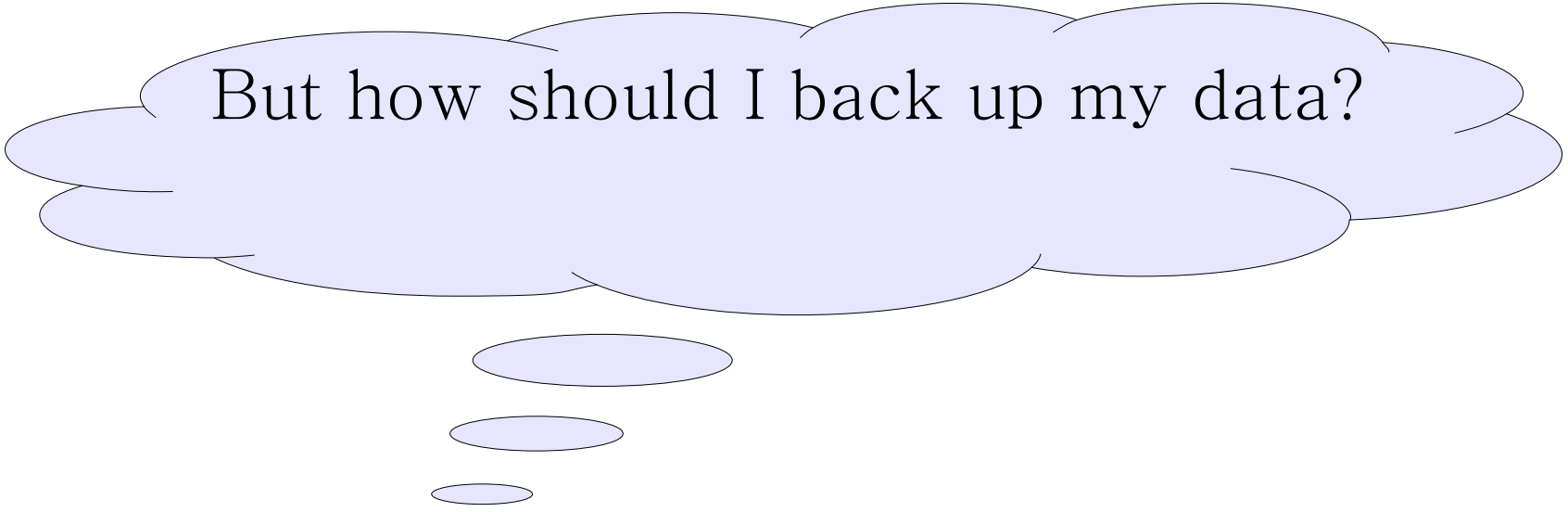
*LAMP (linux, apache, mysql, php/perl/python) application security.

A little intro

Robert Rowley
Security/Abuse

Backups!

- They are the first priority.
- They cover your ass.
- Asset for quick code comparison.
- Have multiple backups (off site preferably)
- Store on read-only media (WORM; write once read many)
- Don't assume anyone else is already handled it.



Beyond the basics.
Try a version control option

rsync/tar work fine, but CVS, SVN, GIT, etc... can do more!

Use them, master them, love them.

They not only allow you to identify and quickly correct bugs, flaws, etc... caused by bad code (and help you identify bad coders). They can be used to mitigate attacks, allowing quick distribution of clean code.

Not only for code! Use SVN for configuration files, etc..

Having a code repository does not mean you do not need to keep backups! Backup your repository too (now use rsync/tar etc..)!

Scenario:

Assessing and securing a (LAMP) web application as soon as possible when changing code is not an option.

Attack (Hired Pen test)

- Cross Site Scripting (XSS)
- SQL injection
- Insecure Code

Defense (Sys Admin)

- Application level (mod_sec)
- Network level (snort)
- File tampering detection

Attack: Cross Site Scripting (XSS)

Type 1: Reflected attacks

The payload exists in the URL and the server side code re-prints the malicious content. *(echo $_GET['varname'];)*

Type 2: Stored attacks

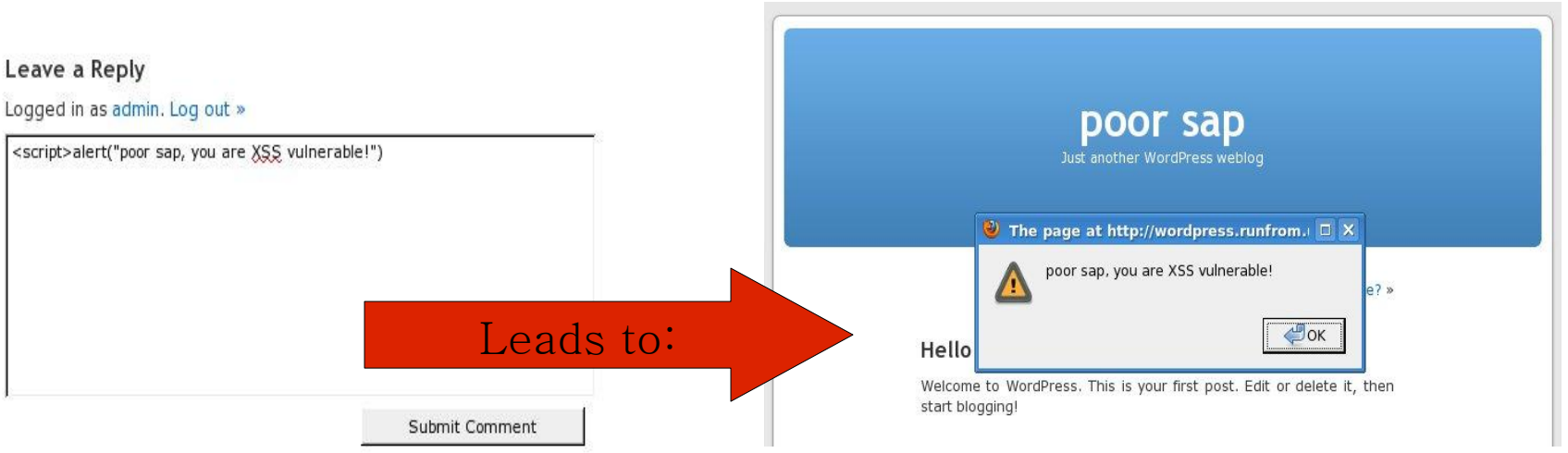
The XSS payload is stored on the server (in MySQL, files etc..) and every subsequent request to the same page displays the injected payload.

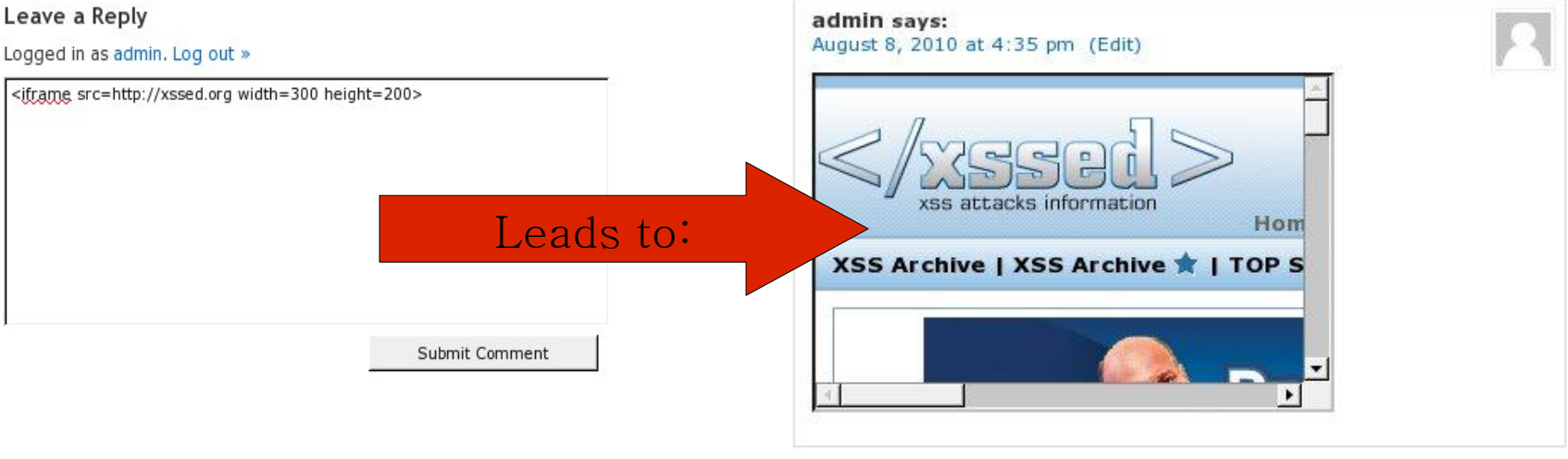
Type 3: DOM attacks

Adjusting DOM attributes on the client's browser directly. *(When designers go bad! javascript adjust data directly, nothing is actually handled in the server side code itself)*

Affect:

- Anyone who visits the page/link is initiating an attack on third party web servers
- Mis-information, changing/reposting information on a credible website
- Leaked information (javascript can access cookie information, HTTP variables etc..)





This really is just fun stuff, know enough javascript and you can cause major havoc or do a 0 height iframe for “click jacking”

“FUN”


Leave a Reply
Logged in as admin. Log out »
<script>document.getElementById('post-1').innerHTML = "whatever i please"
Submit Comment
poor sap
Just another WordPress weblog
Why is everbody always picking on me? »
whatever i please
4 Responses to “Hello world!”

XSS recap

- While limited in it's abilities, that is it's strength. Most developers do not consider XSS a security risk leaving a plethora of vulnerable sites.

- Familiarize yourself with the slight differences between persistent, reflected and DOM XSS attacks

- Study javascript and be multiple browser compliant.

Defense: mod_security

Apache module to quickly mitigate and prevent identified attacks.

You will need to first install mod_security and then include it in your apache configuration. Instructions vary, but modsecurity.org has what you will need.

(example httpd.conf changes after the .so has been compiled)

```
LoadModule security2_module modules/mod_security2.so
 SecRuleEngine On
 Include conf/modsecurity.conf
...
```

Now all you will need are rules!

There are extensive rule sets available at **gotroot.com** and **owasp.org**
but ... it is always best to roll your own.

Here is a list of variables you will have available to scan/review:

ARGS
ARGS_COMBINED_SIZE
ARGS_NAMES
REQBODY_PROCESSOR
REQBODY_ERROR
REQBODY_ERROR_MSG
XML
WEBSERVER_ERROR_LOG
FILES
FILES_TMPNAMES
FILES_NAMES
FILES_SIZES
FILES_COMBINED_SIZE
ENV
REMOTE_HOST
REMOTE_ADDR
REMOTE_PORT
REMOTE_USER
PATH_INFO
QUERY_STRING
AUTH_TYPE
SERVER_NAME
SERVER_ADDR
SERVER_PORT
TIME_YEAR
TIME_EPOCH
TIME_MON
TIME_DAY
TIME_HOUR
TIME_MIN
TIME_SEC
TIME_WDAY
TIME
REQUEST_URI
REQUEST_URI_RAW
REQUEST_LINE
REQUEST_METHOD
REQUEST_PROTOCOL
REQUEST_FILENAME
REQUEST_BASENAME
SCRIPT_FILENAME
SCRIPT_BASENAME
SCRIPT_UID
SCRIPT_GID
SCRIPT_USERNAME
SCRIPT_GROUPNAME
SCRIPT_MODE
REQUEST_HEADERS
REQUEST_HEADERS_NAMES
REQUEST_COOKIES
REQUEST_COOKIES_NAMES
REQUEST_BODY
RESPONSE_LINE
RESPONSE_STATUS
RESPONSE_PROTOCOL
RESPONSE_HEADERS
RESPONSE_HEADERS_NAMES
RESPONSE_BODY
RULE
SESSION
WEBAPPID
SESSIONID
USERID

Here is a quick rule to stop those simple script/iframe injections via the WP comment form:

(contents of conf/modsecurity.conf)

```
SecRule REQUEST_FILENAME wp-comments-post.php chain,deny
  SecRule ARGS:comment "(<script|<iframe)"
```

- Check if the requested filename is wp-comments-post.php and chain (continue checking) with the next rule ...
- Check if the argument (POST or GET) named “comment” has the string <script or <iframe.
- If both are true then take the specified action (deny)

Over-zealous version:

```
SecRule ARGS "(<script|<iframe)" deny
```

- Any argument (POST/GET variable) that matches the pattern “<script” or “<iframe” will be denied.

Remember from earlier?

mod_security recap

Pro

- Prevents attacks from succeeding
- Application level access (no worries about SSL needing to be intercepted/decrypted)
- Very flexible rule sets
- Extendibility with scripting

–(something for another time)

Con

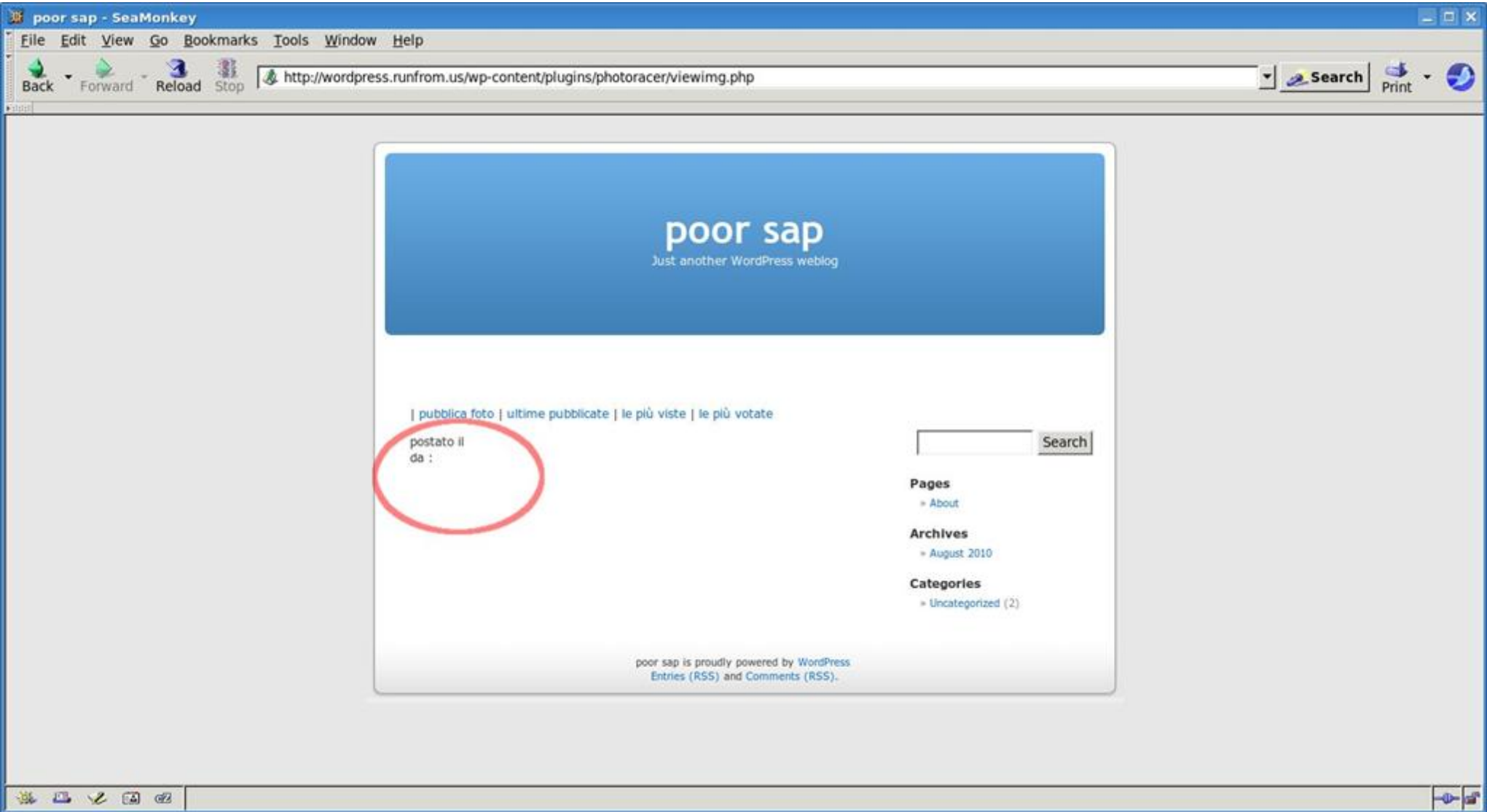
- Only as powerful as the rules used
- Bad rules will create false positives.
- Detectable by attackers, who can adjust their attacks

–(something for another time)

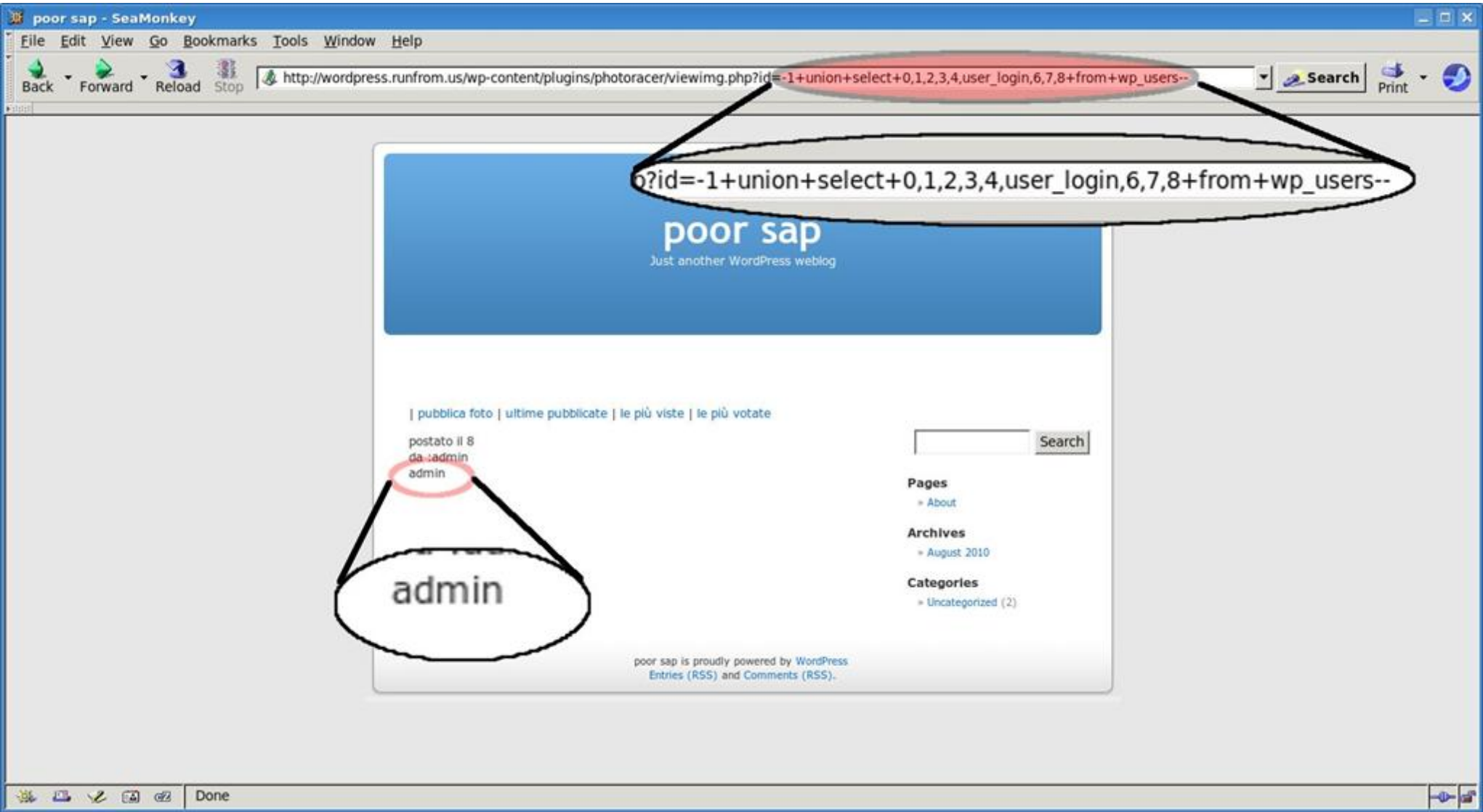
Attack: SQL injection

Do (almost) anything you want to the database!

SQL injection with “photoracer” plugin.



Simplest form, just add a union statement added to one of the GET/POST variables that gets appended to the select query



Here is what went wrong in the code:

```
    16          $imgid = $_REQUEST['id'];
    17  $q1 = "SELECT raceid, wpuid, imgid, imgpath, imgname, imgcomment, sumvotes, imgcountview, tinsert FROM ".
    18          $wpdb->prefix."photoracer WHERE imgid=$imgid";
    19
    20  $out = $wpdb->get_row($q1);
...
   157                                  "da :".get_author_name($out->wpuid)."<br/>".
   158                                  $out->imgcomment."<br/>".
```

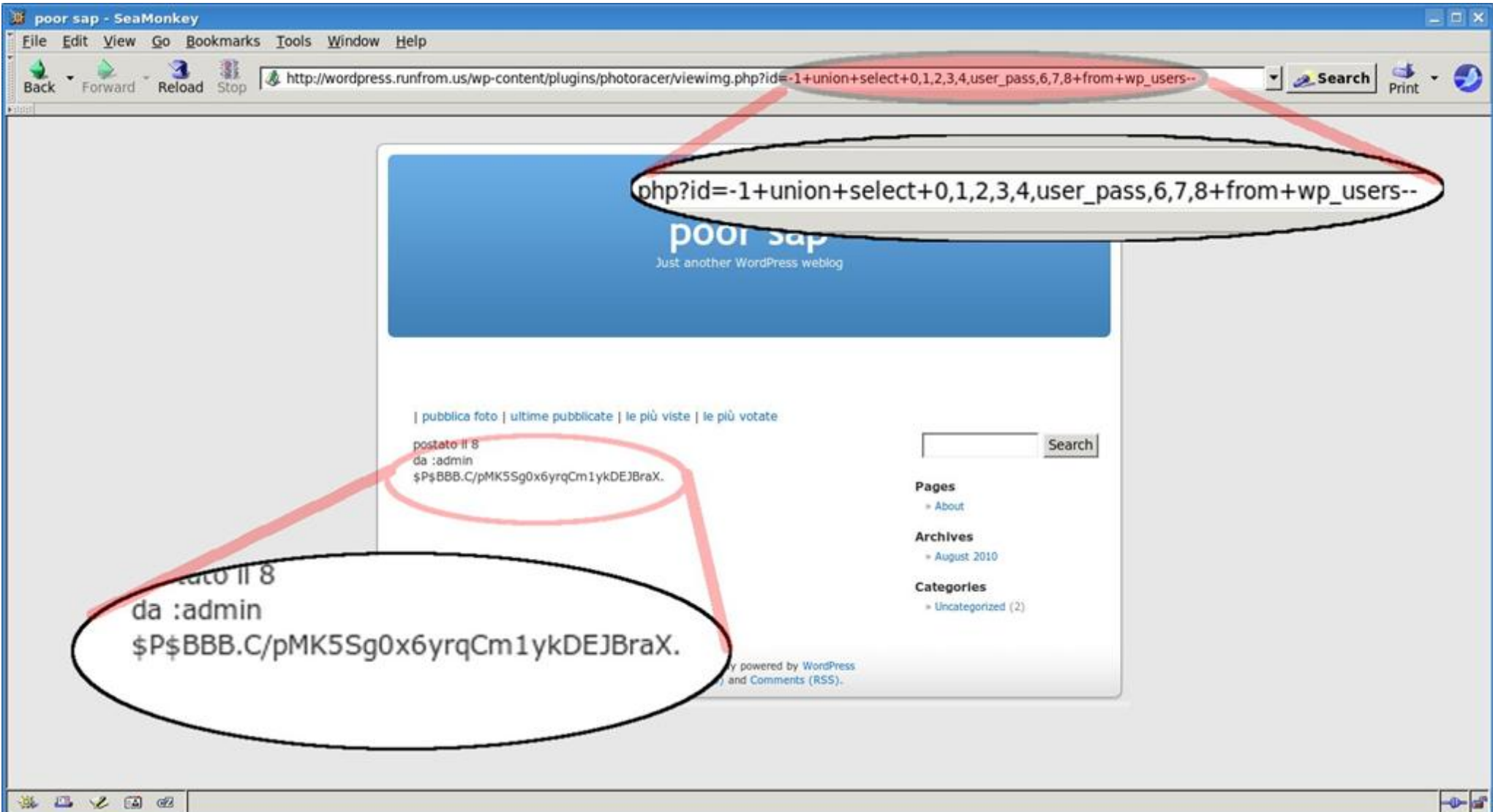
Expected SQL statement:

SELECT raceid, wpuid, imgid, imgpath, imgname, imgcomment, sumvotes, imgcountview, tinsert FROM wp_photoracer WHERE imgid=10

Injected SQL statement:

SELECT raceid, wpuid, imgid, imgpath, imgname, imgcomment, sumvotes, imgcountview, tinsert FROM wp_photoracer WHERE imgid=-1 union select 0,1,2,3,4,user_login,6,7,8 from wp_users--

But wait ... there is more. Let's get some password hashes!



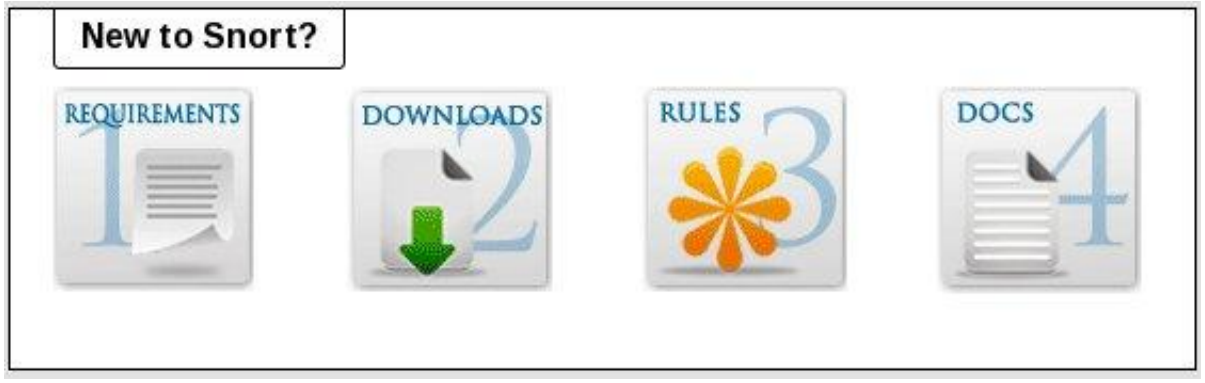
SQL injection recap

- Requires an understanding of SQL statements
- Very “noisy” if the database structure is not known
- Time consuming (if not automated)
- Can do more than SELECT based on user's privileges!
  (DROP, ALTER, CREATE, GRANT statements are all possible.)

DEFENSE: IDS/IPS (snort and more)

Excellent method to monitor network traffic and retrieve information on attacks.

Don't forget your whitelist!



- Snort is a widely used IDS (intrusion detection system)
- Lightweight
- Large user base for example rulesets and help
- Snort can detect more than just web application attacks (unlike mod_security.)
- Setup is easy: download, compile, configure, monitor and update!

- By itself snort will only log the attacks for review, plugins like SnortSam/Guardian.pl will turn snort into a powerful IPS (intrusion prevention system)

Example code to detect the SQL injection attack:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (
msg:"SQL-INJECTION photoracer Sql Injection attempt";
flow:to_server,established;
uricontent:"viewimg.php"; nocase;
uricontent:"id=";
uricontent:"union"; nocase;
uricontent:"select"; nocase;
classtype:web-application-attack;
sid:100000691; rev:2;)
```

(detects http:// ... /viewimg.php ... id=...union...select...

What the attack looks like in the logs:

```
[**] [1:100000691:2] SQL-INJECTION photoracer Sql Injection attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/08-23:26:50.252829 67.159.5.99:39314 -> 66.249.129.23:80
TCP TTL:55 TOS:0x0 ID:63052 IpLen:20 DgmLen:444 DF
***AP*** Seq: 0x4A59450C  Ack: 0x9CB41D1A  Win: 0x16D0  TcpLen: 20
```

These scripts watch the snort log files, upon evidence of an attack they will use iptables/ipfwadm etc... to firewall the attacker's IP.

SNORTSAM/Guardian.pl

The most important thing to remember is: whitelist your system's IP if you plan to test new rules. (this prevents you from locking yourself out of the server)

Example:

Using a script like snortsam/guardian.pl the attacker's IP is blocked.

```
# iptables -n -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
```

Snort recap

Pros

- Lightweight
- Runs independently
- Powerful as an IDS or IPS
- Network level can see more than application level(mod_security)
- Large user base, lots of help available
- As an IPS will stop the attacker in their tracks (at least their IP)

Cons

- Only as good as your rules
- False positives
- Can become very daunting to customize
- Requires third party scripts to be an IPS
- False positives when running a IPS may leave your IP blocked, don't forget to whitelist yourself!

Attack: Code Vulnerability

Code can be attacked!

The #1 fault is, trusting user input.

RISK: Attackers get to do whatever they want.

Attack:
Badly Coded Upload form!

Allowing people to upload files “willy nilly” will get you compromised quickly.

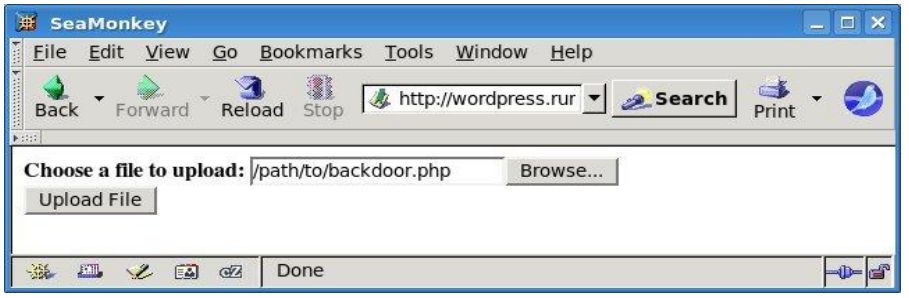
RISK: Allowing users to upload executable files (.php .pl .cgi etc..) or even .html files is giving away the keys to the castle.

```
$target_path = "images/";


/* Add the original filename to our target path.
Result is "uploads/filename.extension" */
$target_path = $target_path . basename( $_FILES['uploadedfile']['name']);


if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target_path)) {
    echo "The file ".  basename( $_FILES['uploadedfile']['name']).
    " has been uploaded";
} else{
    echo "There was an error uploading the file, please try again!";
}
```

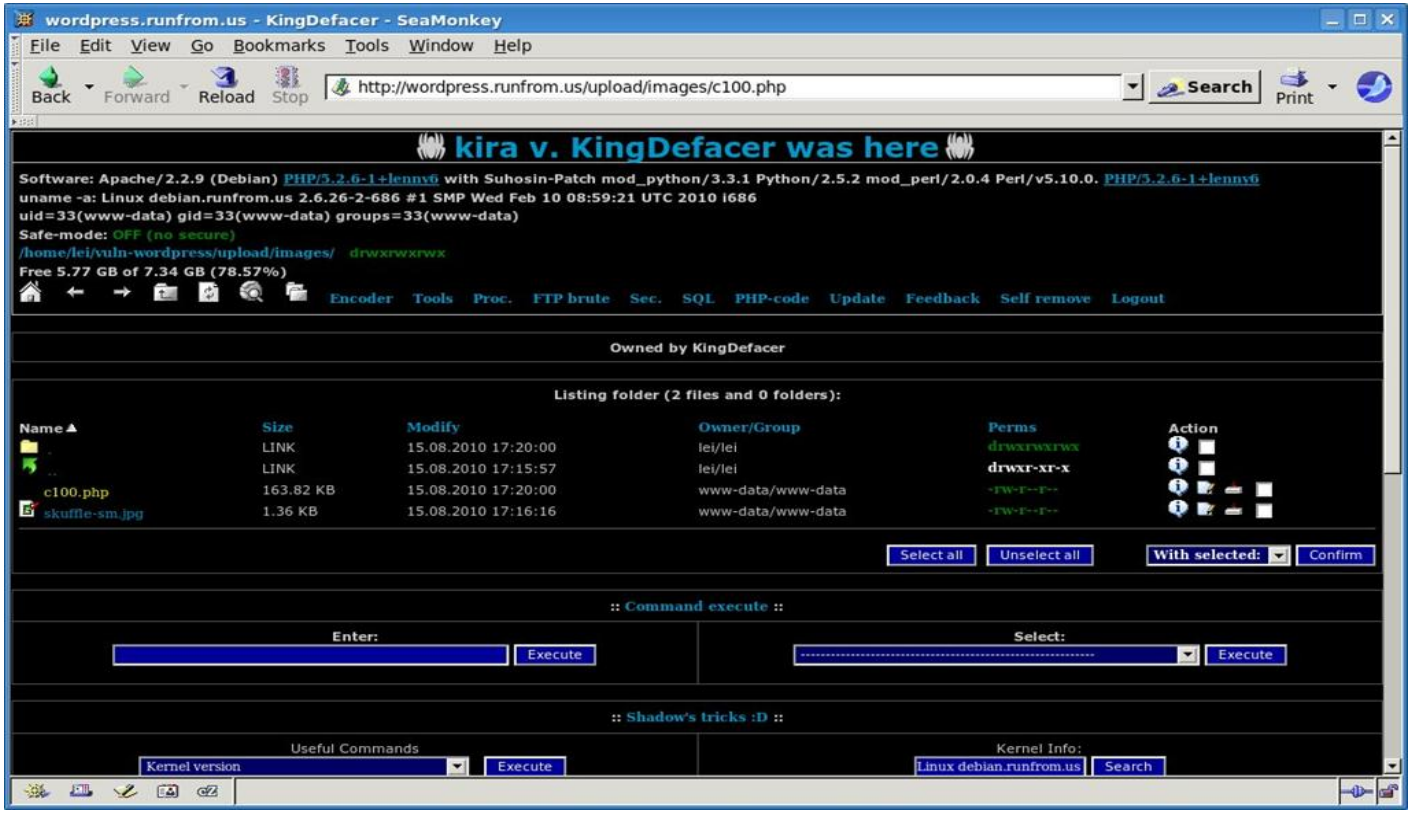
The flaw? No checking the file's extension or checking if it is really an image image, movie, etc...

Find any php shell (http://sh3llz.org/) and upload for “FUN”!

The upload page



Example backdoor



DEFENSE: File monitoring inotify

also:

(tripwire, fschange, kqueue utilities etc..)

Inotify an you...

Kernel module (released in 2.6.13)

Monitors file system changes

Improvement from “dnotify”

Inotify-tools, incron, iwatch, pynotify

inotify-tools

- libinotifytools
- inotifywait
- inotifywatch

inotify/incron quick and easy

1.Verify your kernel has inotify enabled, install incrond

2.Setup incrontab to monitor fies/directores

   i. (directory) (mask) (command)

3. Start/Restart incrond

Example incrontab:

/home/user/website IN_CREATE mail -s 'File created!' admin@website.com

More about masks

IN_ACCESS	File was accessed (read)
IN_ATTRIB	Metadata changed (permissions, timestamps, extended attributes, etc.)
IN_CLOSE_WRITE	File opened for writing was closed
IN_CLOSE_NOWRITE	File not opened for writing was closed
IN_CREATE	File/directory created in watched directory
IN_DELETE	File/directory deleted from watched directory
IN_DELETE_SELF	Watched file/directory was itself deleted
IN_MODIFY	File was modified
IN_MOVE_SELF	Watched file/directory was itself moved
IN_MOVED_FROM	File moved out of watched directory
IN_MOVED_TO	File moved into watched directory
IN_OPEN	File was opened

Passing Variables

The command may contain these wildcards:

$$ - a dollar sign

$@ - the watched filesystem path (see above)

$# - the event-related file name

$% - the event flags (textually)

$& - the event flags (numerically)

Example incron entry: /directory/to/watch IN_MODIFY /path/to/script $@/$#
This will execute the script and pass it the file name which triggered the rule

A “diff”erent solution...

Compare your live data to a backup!

A utility you are probably familiar with “diff” can do this for you.
Just run “diff -r (live directory) (backup directory)”
You will receive a report of file differences.

DEFENSE: Forensics

It's Logs, it's Logs

It's better than bad, it's good.

Logs, logs, logs!

Useful logs:

- Apache (website)
- Auth.log, FTP.log
- Syslog / messages
- Specific IDS logs

Tools to know:

- Grep
- Awk
- Perl/shell scripting

Modified object name: /home/user/website/upload/images/c100.php

Property:	Expected	Observed
* Inode Number	245157	245158
* Modify Time	Sun 15 Aug 2010 06:24:38 PM PDT	Sun 15 Aug 2010 06:26:49 PM PDT
* Change Time	Sun 15 Aug 2010 06:24:38 PM PDT	Sun 15 Aug 2010 06:26:49 PM PDT

Data found in the logs:

debian:~/apache_logs$ **grep 18:24: wp-access.log**

10.0.0.5 - - [15/Aug/2010:18:24:37 -0700] "POST /upload/uploader.php HTTP/1.1" 200

Wrap up

Writing secure code will prevent the need for all of this. Until that day comes, enjoy what you have learned and apply it.

Attacks are mostly automated; happening daily, hourly, right now...

From here on out...

You're on your own to continue researching the subject matters covered briefly in this talk. I have a list of URLs you are free to copy, as well as ideas for more talks if anyone wants to throw their hat in the ring.

- Writing secure code!
- Penetration testing!
- Why your firewall is racist.
- SELECT “MBA” As “DBA”

Further reading!

Snort

- http://www.snort.org (IDS)
- http://www.snortsam.net (Turns snort into an IPS)

mod_security

- http://www.modsecurity.org (Web application level firewall)
- http://www.gotroot.com (mod_security rule list)

iNotify (file change detection)

- http://inotify.aiken.cz (incron etc..)
- http://inotify-tools.sourceforge.net/ (iNotify toolset)

Cross Site scripting:

- http://www.xssed.org (Cross site scripting attacks archive)