

Vulnerable Web Application Enumeration

Stefan Friedli

April 27, 2009

Contents

1	Abstract	2
2	Introduction	3
2.1	All websites are created equal	3
2.2	Private vs. Corporate	3
2.3	Drive-by Infection	3
3	Research	5
3.1	Scenario	5
3.2	Frequently Asked Questions	5
3.3	Finding suitable targets	5
3.4	Verifying Results	6
3.5	Fingerprinting Wordpress	6
4	Results	10
4.1	About these results	10
4.2	Result Summary	10
4.3	Detailed Results	10
4.4	Conclusion	13
	References	14

1 Abstract

This paper discusses the automatic enumeration and fingerprinting of web applications. In this case, the popular WordPress blogging software was used as an example to gain insight about the patch levels in "casual" environments. Deprecated versions of WordPress are known to include several critical vulnerabilities, which make them an easy target to compromise systems in order to perform browser-based exploitation on their visitors or use the underlying infrastructure for several malicious purposes, for example sending spam or hosting malware.

In the first part of the paper, the technical solution to identify installations of the target application using openly available technology is being discussed. Further, the basic method of fingerprinting different versions of WordPress (1.2 up to 2.7.1) are being illustrated.

In the second part of this document, some analysis of a enumeration scenario can be found. The scenario includes the enumeration, fingerprinting and analysis of thousand blogs powered by WordPress in Switzerland and Liechtenstein of which 60 per cent were found to be deprecated and partly prone to certain well-known security vulnerabilities.

2 Introduction

2.1 All websites are created equal

"With great power comes great responsibility." - Spider-Man

The Internet has brought us some fabulous innovations. One of them is personal publishing, which allows anyone to publish almost anything at any given time online with minimal (financial) effort. Anyone is invited to contribute to a huge interlinked network of content, which can nowadays contain nearly any sort of media, including audio, video and offers full interactivity.

While certain politicians in certain european countries still don't really understand this, the Internet is a very plain construct and gives almost anyone the same options, when it comes to content publishing. There is no essential technical difference between <http://www.microsoft.com> and <http://www.fadetoblack.ch> from a client perspective. One of this URIs points to a huge software vendor, making millions a year. The other points to my personal blog. Both use official TLDs, both are accessible from any point of this planet with an Internet connection, except those countries oppressing their population by censoring the Internet probably. Both are serving their HTML content using standard HTTP [RFC2616, Fielding u. a. (1999)]. The same thing applies for any other website, no matter what it contains. There are actually exceptions to this concept, which don't matter for this paper though.

When it comes to security, this might be a problem. Because there is no implicit reason to treat www.nastymalwarepage.com any other than www.nytimes.com, we need to ensure that no malicious resources are being accessed. Usually, this problem is solved using blacklist approaches, that try to enumerate and block malicious websites before employees can actually load them and get their clients compromised. Since August 2006, even Google tries to detect so-called "harmful pages" and warn the user from accessing it by displaying an interstitial page informing about the possible threat.

2.2 Private vs. Corporate

Most people responsible for corporate IT security spend a lot of time and money on secure development and accompanying measures like web application penetration tests and source code analysis to eliminate common threats like persistent script injection vulnerabilities. Individuals running their own non-commercial websites or blogs usually don't, because they don't calculate any risk resulting from potential vulnerabilities usually. This makes private websites the perfect target for hosting malware and/or infecting people while visiting a blog or whatever these pages offer.

2.3 Drive-by Infection

I will keep this really brief here, as there are a lot of very good resources available. Basically, Drive-by Infection is just a fancy word for "You visit a legitimate website that has been compromised and now serves malicious content among the original - non-malicious

stuff". Basically, it's a simple and very understandable concept: Instead of luring people onto a malicious website, you repurpose a legitimate (and vulnerable) website to serve your purpose. Obviously, this can be achieved on different paths, which include any possible way to modify the web content served by the webserver. Some non-conclusive examples commonly used are upload by ftp or WebDAV, persistent XSS- and SQL injection vulnerabilities. Once an attacker found a way to inject his payload onto the legitimate site, there is a huge variety of ways to compromise clients, ranging from very basic and cheeky "Hey, download xyz.exe for [insert fancy feature here]" to browser exploits (see H.D. Moores browser_ autopwn module [Moore (2008)] for a neat demonstration) or browser-based attack frameworks like BeEF [Alcorn (2006)] or SWARM [Friedli (2007)].

3 Research

3.1 Scenario

So we know, we have a lot of possibilities to compromise clients, when we get to embed malicious content into legitimate sites. So what we need to find are legitimate sites that contain serious flaws that have not been patched or are now known to the public yet.

I was very curious to find out, how hard it would be to find a few dozens of vulnerable pages, so I constructed the following simple scenario to gather some data:

- Enumerate blogs located in Switzerland and Liechtenstein that are powered by the popular open-source software "Wordpress"
- Verify that the blog is active and contains more than 5 posts. Make sure the oldest post has a maximum age of 14 days to prevent inactive blogs to be included.
- Fingerprint the installation to find out the exact version of the Wordpress installation being used. Map this to existent vulnerabilities.
- Feed all data into a database to allow the analysis of the results.
- Find out that a lot of people didn't do their homework...

3.2 Frequently Asked Questions

Before we start: The first question I frequently get, when I talk about the results of this little experiment is: "Why do you keep picking on WordPress? There is other software being used, that sucks more!". So I am gonna answer this one first here: I don't. While my personal opinion on Wordpress is irrelevant and not representative, Wordpress is currently the most popular blog engine out there, followed by Movable Type. On Technoratis Top 100 list, about 25% of all blogs are using Wordpress. Of course, the most comprehensive approach would be to fingerprint all products available, but fortunately, people looking to spread malware don't care too much about scientific perfection as well and compromise whatever is vulnerable.

I also chose to limit myself to blogs being hosted on a .ch (Switzerland) or .li (Liechtenstein) domain. Both domains are managed by the Swiss NIC Switch¹. Switch requires any domain owner to register with a valid technical and administrative contact and supplies a standard whois service, that allows me to get in touch with blog owners when I run into something unexpected.

3.3 Finding suitable targets

Thanks to Google, finding a certain type of application is easy nowadays. So finding targets is quite easy - the following google dorks will yield more than enough resources than we need.

¹<http://www.switch.ch/>

```

1  inurl:.ch/ inurl:wp-login.php?action
2  inurl:.li/ inurl:wp-login.php?action

```

While the google dork is basically a no-brainer, enumerating search results can be quite tricky, as the old SOAP API does not seem to be active any more. You can still automate Google Searches using ruby with the fabulous scRUBYt gem, but as this (in my understanding) would violate Googles Acceptable Use Policy, I will not recommend this. Also, Google happens to think, you're infected with malware when you squeeze out too many results in a short period of time and reacts with a captcha, which scRUBYt cannot handle. I re-used an old Greasemonkey Script I used years ago in Web Application Pentests to dump URIs and browsed the results manually. Collecting 1000 unique URIs took about 3 minutes. It's not pretty, but it works quite well:

```

1  // ==UserScript==
2  // @name           Google Result Dump
3  // @include        http://www.google.*/search*
4  // ==/UserScript==
5
6  var allDivs , thisDiv ;
7
8  allDivs = document.evaluate(
9    "//a[@class='l ']",
10   document , null ,
11   XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
12   null);
13
14  for (var i = 0; i < allDivs.snapshotLength; i++) {
15    thisDiv = allDivs.snapshotItem(i);
16    GM_xmlhttpRequest({
17      method: 'GET',
18      url: 'http://localhost/grd/d.php?d='+thisDiv ,
19      headers: { 'User-agent': 'Mozilla/4.0 (compatible) Greasemonkey',
20                'Accept': 'application/atom+xml,application/xml,text/xml', }
21    });
22  }

```

Listing 1: Fetching Google result links to a local dump script

3.4 Verifying Results

Now we got a lot of (possible) Wordpress installation. As I stated above, I wanted to avoid old and inactive blogs to be included in the results, so I used RubyRSS [Tikhonov (2008)] to check if my minimum requirements (5 posts, last post made in the last 14 days) are being fulfilled, by parsing the RSS feed generated by WP installation. As a nice side effect, requests resulting in a 404 or 500 HTTP error message could be flagged for review and/or deleted right away.

3.5 Fingerprinting Wordpress

Now this is where things get interesting. I compiled a list of the top 1000 blogs I collected using the script above and looked for ways to determine the exact version. And thanks

to the WordPress Development Team, it's very easy to do in most cases.

While WordPress offers a huge lot of possibilities to customize the frontend theme, which is used to display posts, categories and so on, no one really cares about the backend. As a result, wp-login.php (which is also used for registered, but non-administrative users to login and therefore public) is usually left untouched. Now all we have to do is to find out, how those pages looked in the various versions. Well - we would probably have to do that, but lucky for us, the developers leave us the exact version in the delivered contents source code:

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
   w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml" dir="ltr">
3 <head>
4   <title>WP-Test251 &rsaquo; Login</title>
5   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
6   <link rel='stylesheet' href='http://localhost/wp/251/wp-admin/css/login.
   css?version=2.5.1' type='text/css' />
7 <link rel='stylesheet' href='http://localhost/wp/251/wp-admin/css/colors-
   fresh.css?version=2.5.1' type='text/css' />
```

Listing 2: WordPress Version Information in /wp-login.php

So the easiest way to find out the version in about 90% of all installations is to extract the "version" parameter, which is included in the external stylesheet reference. I do not know exactly why this information is supplied there, it's much likely to be used for statistical reasons. However, it is useful if you search vulnerable software as well. It seems, that this has been introduced somewhere along the 2.x branch of Wordpress as really old versions do not have this parameter attached to the stylesheet link.

To extract the information, I would recommend using another fabulous ruby gem: HPricot

```
1 def getVersion(wpuri)
2   begin
3     doc = Hpricot(open(wpuri, 'User-Agent' => 'Mozilla/5.0 (Macintosh; U;
   Intel Mac OS X 10.5; de; rv:1.9.0.7) Gecko/2009021906 Firefox/3.0.7
   '))
4     result = doc.search("//link[@rel='stylesheet']:first")
5
6     result.each { |res|
7       str = res.attributes['href']
8       version = /.css?v\w+=(.*)$/.match(str)[1]
9       return version
10    }
11  rescue
12    return "ERROR"
13  end
14 end
```

Listing 3: Function getVersion - returns a string containing version information

The code above uses XPath to search the DOM tree for the first <link> element with the "rel" attribute set to "stylesheet". It extracts the href attribute, extracts the version

string (either defined via `?version=` or `?ver=`, depending on the exact version being used) using a simple regular expression and returns it as a string. If something goes wrong, the (lousy...) error handling just uses `ERROR` as the version string, to allow manual investigations when necessary.

Among the thousand blogs I checked using this script, the percentage of failed fingerprinting attempts was very low. In a dozen cases, the owner of the blog obviously tried to customize the backend looks by altering the corresponding templates, wiping away the original stylesheet reference. Actually, one of those thousand blogs has intentionally replaced the version string for security reasons.

Luckily, the `wp-login.php` file is not the only source for version information. If the information cannot be retrieved there, there are some other ways. As I mentioned before, older versions of Wordpress do not supply the version parameter in the stylesheet reference. So, let's take a look in the RSS feed generated by Wordpress 1.5

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <!-- generator="wordpress/1.5" -->
3 <rss version="2.0"
4   xmlns:content="http://purl.org/rss/1.0/modules/content/"
5   xmlns:wfw="http://wellformedweb.org/CommentAPI/"
6   xmlns:dc="http://purl.org/dc/elements/1.1/"
7 >
8 <channel>
9   <title>WordPress Test Blog</title>
10  <link>http://localhost/wp/151/</link>
11  <description>Another installation...</description>
12  <pubDate>Thu, 23 Apr 2009 14:21:38 +0000</pubDate>
13
14  <generator>http://wordpress.org/?v=1.5</generator>
15  <language>en</language>
```

Listing 4: WordPress Version Information in `/wp-rss2.php`

This way worked for almost any blog, except the single one that has removed its version information completely, as mentioned above. Also, some authors crippled their installations to remove the RSS feeds, which renders this method useless. Anyway, we are talking about 3 out of 1000 here. Exactly those three blogs were actually running Wordpress 2.0 and guess what you will find, if you just look into the source of the initial page you get, when visiting those blogs:

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
   w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml">
3
4 <head profile="http://gmpg.org/xfn/11">
5   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
   />
6
7   <title>[remove because of explicit content]</title>
8
9   <meta name="generator" content="WordPress 2.0" /> <!-- leave this
```

```
for stats —>
```

Listing 5: WordPress Version Information in /index.php

All these steps can be automated as shown in the first listing using HPricot within minutes, so realizing a scanner implementing all these sources is no problem. These are actually so easy, that you don't need to resort to the more esoteric stuff, for example:

- People tend to forget things. For example the file `readme.html` in root directory of the WordPress tarballs, that are usually left behind. It does include the version of the package. It is not 100% accurate in some cases though. And guess who forgot to remove this file? The single person who removed any other version reference of the installation. Nobody's perfect after all.
- Nobody likes comment spam. This is why people use Akismet, an anti-spam plugin that was introduced when WordPress 2.0 came out. If the version is older, trying to access `/wp-content/plugins/akismet/akismet.gif` will yield a HTTP 404 error instead of a graphics file.
- Managing versions can be a pain. The WordPress Team illustrated this problem quite well. The file `/wp-content/themes/default/style.css` contains the style information for the famous WordPress default theme Kubrick, which no one seems to remove as well.

For example, there is a version change of this file between WP 2.0 and 2.2, which takes it from 1.5 to 1.6 and removes a part of the initial comment. In current versions (2.7.x) the version is still 1.6 - but obviously since version 2.6.1, the following line is included:

```
1      Tags: blue, custom header, fixed width, two columns, widgets
```

Listing 6: WordPress Version Information in /index.php

- Cosmetic issues can be used for fingerprinting as well. The graphic used to illustrate the `wp-login.php` file has changed a few times over the last years. I collected some data, which is almost certainly not complete but gives an impression about those specific changes.

Version	Filename	SHA1 Hash
1.5 - 2.1	<code>wp-small.png</code>	<code>0f8b955fa18b0dc450d8b20f50160e438e0a0778</code>
2.2 - 2.3 (en)	<code>login-bkg-tile.gif</code>	<code>7c7a149f4d3ec560c9578424708fc01caf317a91</code>
2.2 up to 2.3 (de)	<code>login-bkg-tile.gif</code>	<code>60ca64dd716d039723b65bc6e2468ae4c46520a1</code>
2.5 - 2.7+ (en)	<code>wordpress-logo.png</code>	<code>6563a0043e5bc4632a65aef8be79e15dbd3b8999</code>
2.5 - 2.7+ (de)	<code>wordpress-logo.png</code>	<code>7875476c54bd39f45568ee71994f135706ad1185</code>

Checking the absence of new features by directly accessing them via HTTP is a good way to find additional points to identify deprecated versions of an application. This works on almost every other application for obvious reasons.

4 Results

4.1 About these results

Automated fingerprinting is a lot of fun - illustrating how you can do it is as well. However, you might be more interested in the results of the enumeration and fingerprinting steps before.

As I described before, thousand active blogs on .ch and .li domains were tested. Every single validated blog could be fingerprinted, so no results were purged. Of course, these thousand blogs are only a small fragment of all installations available on the Internet, so this results should be considered a random sample with a limited significance.

4.2 Result Summary

As you probably expected, a lot of the installations that were tested are not up to date. Actually, 60% of the installations do not represent the newest version of Wordpress currently available (2.7.1). 59% do not even use the same newest branch (2.7.x) which is available since nearly half a year.

2% of all blogs tested were still using the ancient 1.x branch of WordPress, which is nearly four years old and has some really ugly vulnerabilities.

Another interesting fact: You probably read about the compromise of the WordPress Source of the version 2.1.1 in March 2007. Unknown individuals gained access to the repository and added a `passthrough()` function to `theme.php` and `feed.php`, that would execute any command via the `ix` and `iz` GET parameters. This was slashdotted on 3rd of March [Zonk (2007)] and hit several other IT news sites. Compromised versions (2.1.1 with the `passthru()` function enabled) can still be found nowadays. In this test, 9 vulnerable installations were found. The owners have been notified.

4.3 Detailed Results

As you are still reading, you are probably interested in the more juicy details. I compiled several result graphics to illustrate the current situation, that I gathered using the mechanisms explained earlier in this document. I did mention this in the introduction to this document, but I will do it again: All this data was gathered on a limited amount of Wordpress installations in Switzerland and Liechtenstein only. Though I did not tamper with any data, (e.g. by removing up-to-date blogs to make things look even more worse...), this results do not represent a global perspective - though I honestly do not expect any significant change to the results.

Figure 1 shows a simple pie chart with two sectors. It illustrates that 60 per cent of all the installations that were fingerprinted are not running the latest version of Wordpress at the date of the test. The latest version verification was performed approximately two months after the release of WordPress 2.7.1, which is still the most recent version at the time of writing.

So before we get more into details, let's stick to this very basic interpretation of the gathered data. Most enterprise customers I worked with over the last few years have

Deprecated Software Quota

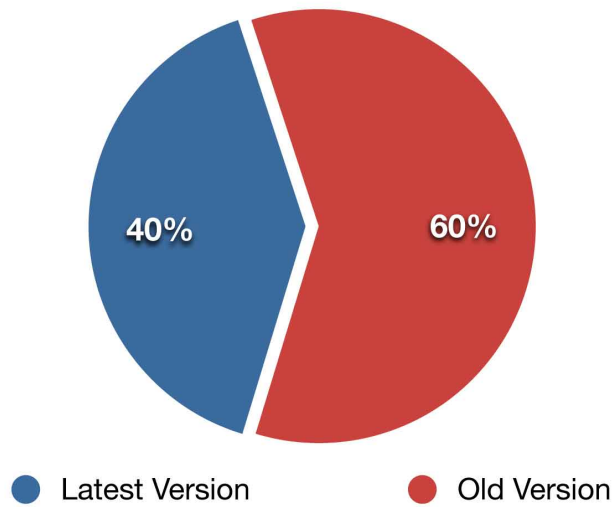


Figure 1: Deprecated Software Quota

sophisticated patch management process workflows in place. Most of these require, that critical security flaws are patched within five days, while minor flaws need to be addressed within 30 days. If you work in such an environment and try to get all your Oracle installations up to date in time, you will probably find hell a quiet comfy place to be - but let's stick to WordPress. Unlike complex environments, a single WordPress Installation can be patched up to date in a timeframe of 5 (for a minor update) to 120 minutes (to patch from 1.2 to 2.7.1). So time should not be a problem - but the lack of awareness obviously is.

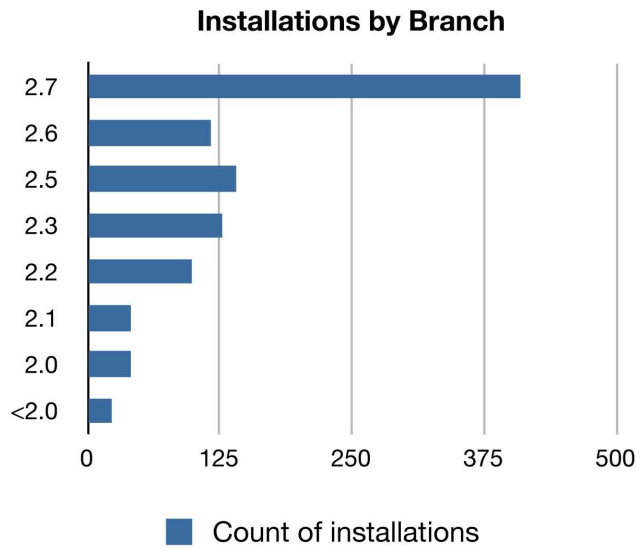


Figure 2: Installations by Branch

If we take a look at figure 2, we see a bar chart illustrating the distribution of the inspected installations over all minor branches in version 1 and 2 of WordPress up to a current date. We eventually discover that the current branch 2.7 is the branch most commonly used. All those installations are either quite new or are maintained actively. Though not all of these installations use the very latest version of WordPress, we still have to consider these as acceptable.

However, as we go down the other versions, we discover a huge number of deprecated installations, that should be considered a risk for a) the owner, b) the hosting ISP as well as c) any visitor. A lot of versions still being used frequently have security issues that were reported long time ago and are (obviously) not maintained in a responsible and acceptable manner by their respective owners.

Age of versions being used

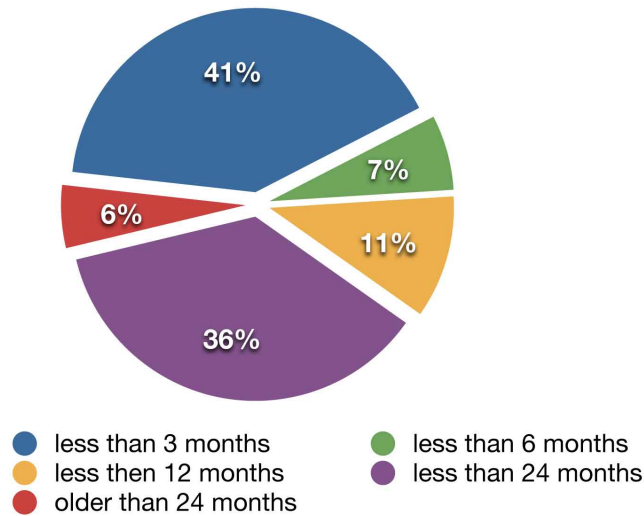


Figure 3: Age of Versions

If we look up, when the single versions of Wordpress were released, we can take this a bit further to gather information on how old these versions being used really are. This is being illustrated in figure 3, which shows the distribution of the installed versions being used over the last five years. As we can see, a lot of installations were made quite a while ago. Even if we consider all versions released in late 2008 and 2009 to be acceptable, we still find an alarming number of older installations. And yes, there are still versions being used that were released in 2004 (WordPress Version 1.2.2).

4.4 Conclusion

Sometimes it is very interesting to investigate something, even though you can imagine the outcome from the very beginning. I have been doing a good amount of large-scale security assessments over the last few years and I did not expect the casual blogging community to be more successful in patch management than some of the ambitious corporate customers I worked with.

The interesting things about all the deprecated installations is, that they are all potential targets for future attacks. Anyone who is running a website should be responsible for taking appropriate measures to avoid abuse of the services he provides, that could harm himself or others. Someone running WordPress 1.2.2 with a whole bucket of critical vulnerabilities obviously lacks the basic awareness to acknowledge this.

During the whole research, no hostile actions were taken against vulnerable sites at any time. After the public release of this paper, the affected blogs were informed about deprecated versions being used. I will probably do a differential analysis at some point in the future, to gain further information about the development of these issues. Until then, feel free to to contact me by eMail, if you want to share you thoughts on the issue.

List of Figures

1	Deprecated Software Quota	11
2	Installations by Branch	12
3	Age of Versions	13

Listings

1	Fetching Google result links to a local dump script	6
2	WordPress Version Information in /wp-login.php	7
3	Function getVersion - returns a string containing version information	7
4	WordPress Version Information in /wp-rss2.php	8
5	WordPress Version Information in /index.php	8
6	WordPress Version Information in /index.php	9

References

- [Alcorn 2006] ALCORN, Wade: BeEF - Browser Exploiting Framework. 2006. – URL <http://www.bindshell.net/tools/beef/>. – [Online; accessed 22-April-2009]
- [Fielding u. a. 1999] FIELDING, R. ; GETTYS, J. ; MOGUL, J. ; FRYSTYK, H. ; MASINTER, L. ; LEACH, P. ; BERNERS-LEE, T.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard). Juni 1999 (Request for Comments). – URL <http://www.ietf.org/rfc/rfc2616.txt>. – Updated by RFC 2817
- [Friedli 2007] FRIEDLI, Stefan: SWARM - Technical Concept. 2007
- [Moore 2008] MOORE, H.D.: Release Notes for Metasploit 3.2. 2008. – URL <http://www.metasploit.com/documents/RELEASE-3.2.txt>. – [Online; accessed 23-April-2009]
- [Tikhonov 2008] TIKHONOV, Sergey: RubyRSS - Eating feeds tastefully. 2008. – URL <http://www.rubyrss.com/>. – [Online; accessed 22-April-2009]
- [Zonk 2007] ZONK: Wordpress 2.1.1 Release Compromised by Cracker. 2007. – URL <http://it.slashdot.org/article.pl?sid=07/03/03/0427211&from=rss>. – [Online; accessed 22-April-2009]