Vulnerability Discovery with Time Constraints

Kaveh Razavi, Hamid Ebadi, Mehdi Shajari, Babak Sadeghian

Amirkabir University of Technology CERT

Contents

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  - PHP (CVE-2008-5498)
  - xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

CERT and Vulnerability Discovery

- Vulnerabilities are one of the main reasons for security incidents
- Prevention is better than cure, even when it comes to security
- CERTs normally have a VA team (Vulnerability Analysts)
- One of the tasks of VA is VD service in time of need
- Time of need
  - An incident has happened on the **most recently updated** software and we need to know how (for future prevention)
  - Deployment of a software in an **insecure environment**
  - A **3rd party company** may require some security assurance of the API they are using
  - …

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  - PHP (CVE-2008-5498)
  - xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

A Brief Overview of Software Security

- Security must be insured in all the steps of software development
- Software development cycle
  - Requirements
  - Analysis
  - Design
  - Implementation
  - Test
    - Software testing in software quality assurance
    - Software security assurance
    - Vulnerability discovery as a part of SQA

A Brief Overview of Vulnerabilities

- A software security vulnerability might cause:
- – Unauthorized Access
- – Information Disclosure
- – Data Invalidation
- – Denial of Service
- Causes of software vulnerabilities
- – Memory Access Violations
    - Buffer Overflows (Stack, Heap, BSS)
- – Improper I/O Handling
    - Format Strings
    - Injections (SQL Injection, Code Injection)
    - Directory Traversal
- – Race Conditions
    - TOCTOU
- – Logical Errors

Vulnerability Discovery Methodology

- A lot of contributions from academia
  - White-Box
  - Black-Box
  - Gray-Box
- Most of these methods are stayed out of technology
  - They need a lot of time and time is costly in software development
  - Security analysts are expensive

The White, The Black and The Gray

	White	Black	Gray
Implementation Time	Low ~ Medium	Low	Software
Source code	Required	Not Required	Both!
Complexity	Low ~ Medium	Low ~ Medium	Medium ~ High
Automation	Difficult	Easy	Software
Patch	Easy	Difficult	Software

- Choosing an appropriate method
  - Time
  - Automation
  - Flexibility
  - Effectiveness

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  - PHP (CVE-2008-5498)
  - xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

Our Proposed VD Procedure I

1. Gathering Information
2. Designing/Implementing Test Case Generator
3. Vulnerability Analyzing
4. Patching and Reporting

Our Proposed VD Procedure II

- Backgrounds of this proposed VD procedure
- Discussed with two real world example showing the effectiveness of this procedure
  - You all know PHP! It has shown very risk induced when used in shared hosting environments
  - xrdp is a remote desktop server with a lot of features
- Software source code independency

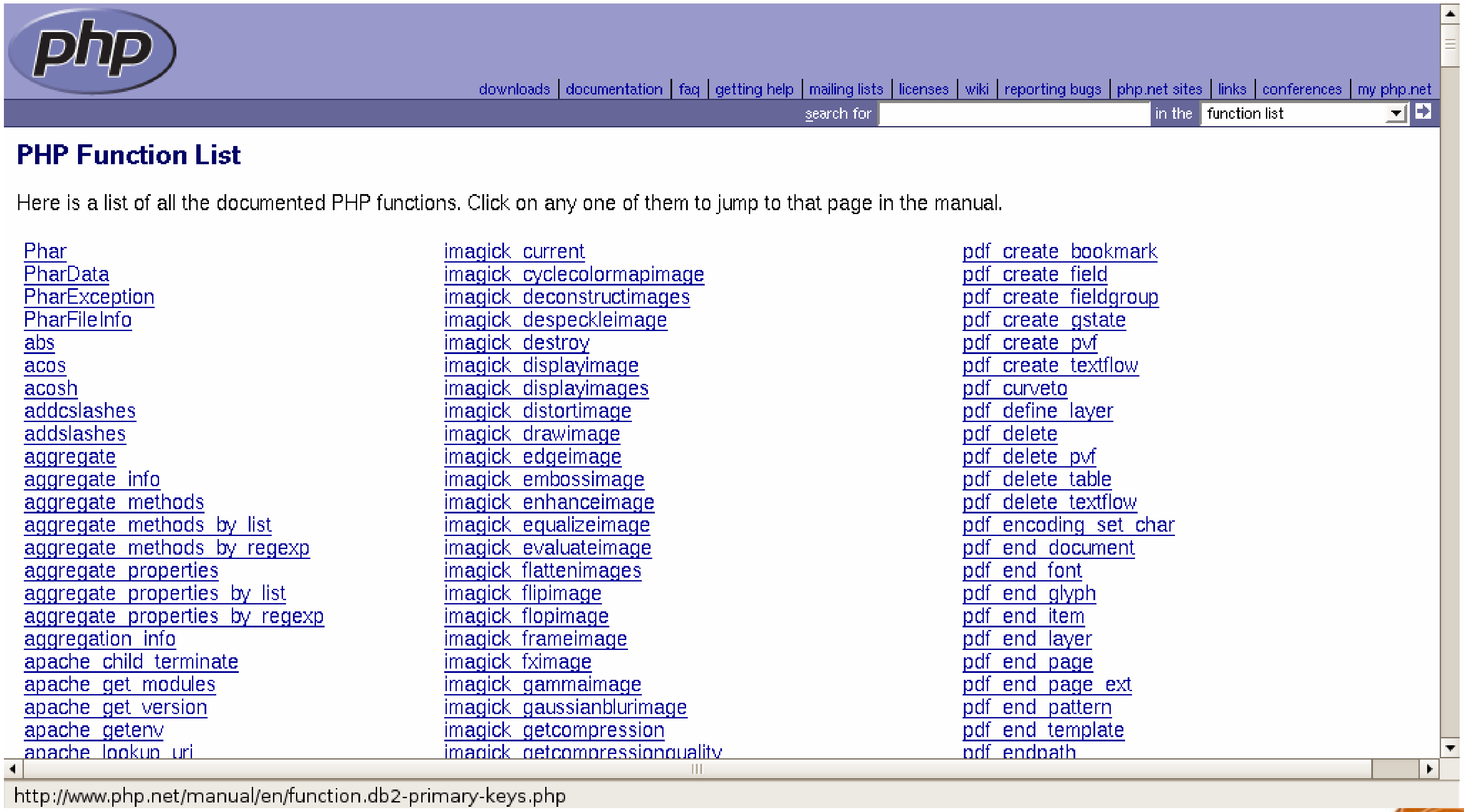
Gathering Information

- Determining software functionality
- Using software engineering diagrams
- Software documentations
- Comparing with other similar software
- Interaction of software with environment
  - Network activities
  - Input Files
  - The external API
  - etc
- Tools to gather information

Gathering Information in Action I

- PHP code is externally exposed via its APIs
- We would like to have a list of PHP APIs
- PHP APIs are also very easy to audit (loosely typed variables)
- Checking the PHP website we found the information we were looking for

Gathering Information in Action II



downloads | documentation | faq | getting help | mailing lists | licenses | wiki | reporting bugs | php.net sites | links | conferences | my php.net

search for [ ] in the function list

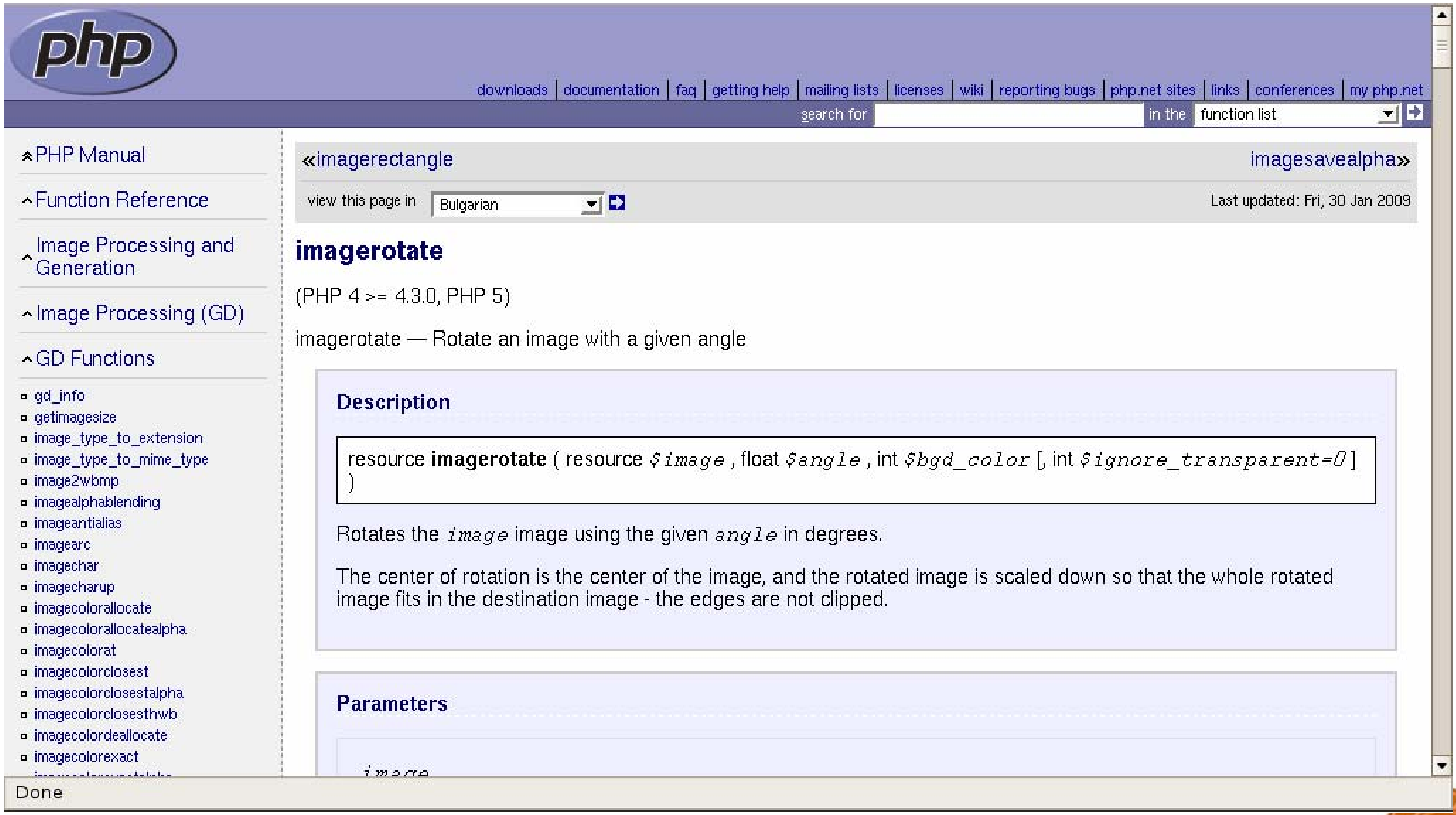
PHP Function List

Here is a list of all the documented PHP functions. Click on any one of them to jump to that page in the manual.

Phar	imagick_current	pdf_create_bookmark
PharData	imagick_cyclecolormapimage	pdf_create_field
PharException	imagick_deconstructimages	pdf_create_fieldgroup
PharFileInfo	imagick_despeckleimage	pdf_create_gstate
abs	imagick_destroy	pdf_create_pvf
acos	imagick_displayimage	pdf_create_textflow
acosh	imagick_displayimages	pdf_curveto
addcslashes	imagick_distortimage	pdf_define_layer
addslashes	imagick_drawimage	pdf_delete
aggregate	imagick_edgeimage	pdf_delete_pvf
aggregate_info	imagick_embossimage	pdf_delete_table
aggregate_methods	imagick_enhanceimage	pdf_delete_textflow
aggregate_methods_by_list	imagick_equalizeimage	pdf_encoding_set_char
aggregate_methods_by_regexp	imagick_evaluateimage	pdf_end_document
aggregate_properties	imagick_flattenimages	pdf_end_font
aggregate_properties_by_list	imagick_flipimage	pdf_end_glyph
aggregate_properties_by_regexp	imagick_flopimage	pdf_end_item
aggregation_info	imagick_frameimage	pdf_end_layer
apache_child_terminate	imagick_fximage	pdf_end_page
apache_get_modules	imagick_gammaimage	pdf_end_page_ext
apache_get_version	imagick_gaussianblurimage	pdf_end_pattern
apache_getenv	imagick_getcompression	pdf_end_template
apache_lookup_uri	imagick_getcompressionquality	pdf_endpath

http://www.php.net/manual/en/function.db2-primary-keys.php

Gathering Information in Action III



imagerotate

(PHP 4 >= 4.3.0, PHP 5)

imagerotate — Rotate an image with a given angle

Description

resource **imagerotate** (resource *$image* , float *$angle* , int *$bgd_color* [, int *$ignore_transparent=0*])

Rotates the *image* image using the given *angle* in degrees.

The center of rotation is the center of the image, and the rotated image is scaled down so that the whole rotated image fits in the destination image - the edges are not clipped.

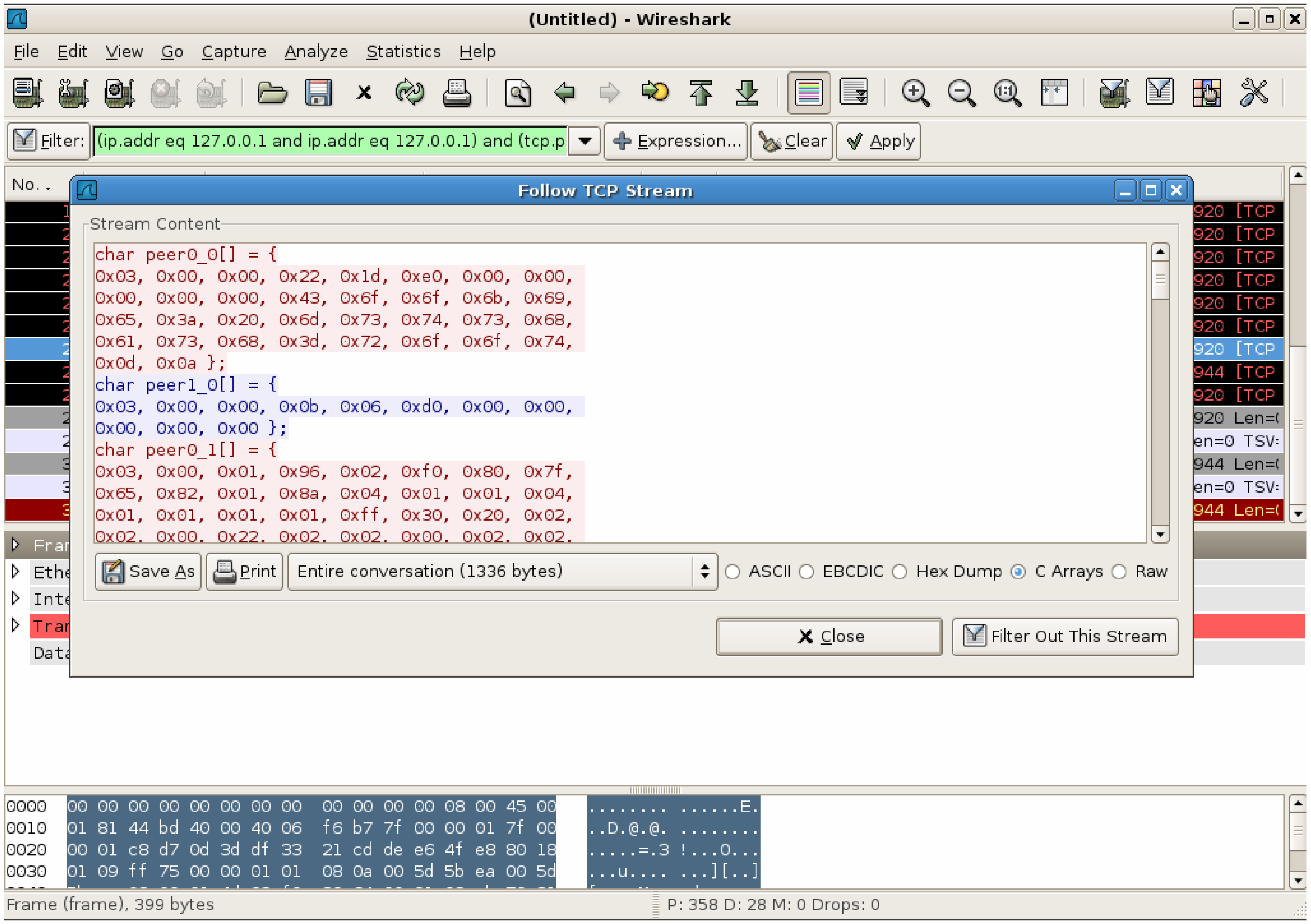
Parameters

image

Gathering Information in Action IV

- Gathering proper data for auditing xrdp would be a bit trickier
  - xrdp deploys a complex binary protocol
- Some samples from xrdp network traces is required
  - Wireshark
    - Beautiful GUI
    - Powerful filters
    - TCP stream following
  - Tcpdump
    - Fast setup
    - No GUI

Gathering Information in Action V



Filter: (ip.addr eq 127.0.0.1 and ip.addr eq 127.0.0.1) and (tcp.p

```
char peer0_0[] = {
0x03, 0x00, 0x00, 0x22, 0x1d, 0xe0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x43, 0x6f, 0x6f, 0x6b, 0x69,
0x65, 0x3a, 0x20, 0x6d, 0x73, 0x74, 0x73, 0x68,
0x61, 0x73, 0x68, 0x3d, 0x72, 0x6f, 0x6f, 0x74,
0x0d, 0x0a };
char peer1_0[] = {
0x03, 0x00, 0x00, 0x0b, 0x06, 0xd0, 0x00, 0x00,
0x00, 0x00, 0x00 };
char peer0_1[] = {
0x03, 0x00, 0x01, 0x96, 0x02, 0xf0, 0x80, 0x7f,
0x65, 0x82, 0x01, 0x8a, 0x04, 0x01, 0x01, 0x04,
0x01, 0x01, 0x01, 0x01, 0xff, 0x30, 0x20, 0x02,
0x02, 0x00, 0x22, 0x02, 0x02, 0x00, 0x02, 0x02,
```

Entire conversation (1336 bytes)

ASCII ( ) EBCDIC ( ) Hex Dump ( ) [✓ C Arrays] Raw ( )

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  01 81 44 bd 40 00 40 06  f6 b7 7f 00 00 01 7f 00   ..D.@.@. ........
0020  00 01 c8 d7 0d 3d df 33  21 cd de e6 4f e8 80 18   .....=.3 !...O...
0030  01 09 ff 75 00 00 01 01  08 0a 00 5d 5b ea 00 5d   ...u.... ...][..]
```

Frame (frame), 399 bytes   P: 358 D: 28 M: 0 Drops: 0

Designing/Implementing Test Case Generator

- Designing an effective test case generator is difficult and depends on the previous steps
  - Designing a set of powerful test cases for a specific software could be time consuming (effective)
  - Sending (executing) raw random data proven to be ineffective in many cases (easy to implement)
- Since time is an important factor, the design should be effective and easy to implement
- It should also be easy to trace what input causes a fault
- Randomness is powerful, make use of it whenever possible in the design of your test case generator

Designing/Implementing Test Case Generator in Action I

- All the functions (APIs) name from the PHP website are retrieved with a simple regular expression
- Since PHP variables are loosely typed, a variable could be anything
  - A long string (buffer overflow)
  - A negative number (integer overflow)
  - A zero or a very big number (division by zero, null pointer dereferences)
  - Some streams (required by some functions)
  - Some random characters (random behavior)
- The general method that we used for PHP is a combination of random testing and basic fuzzing

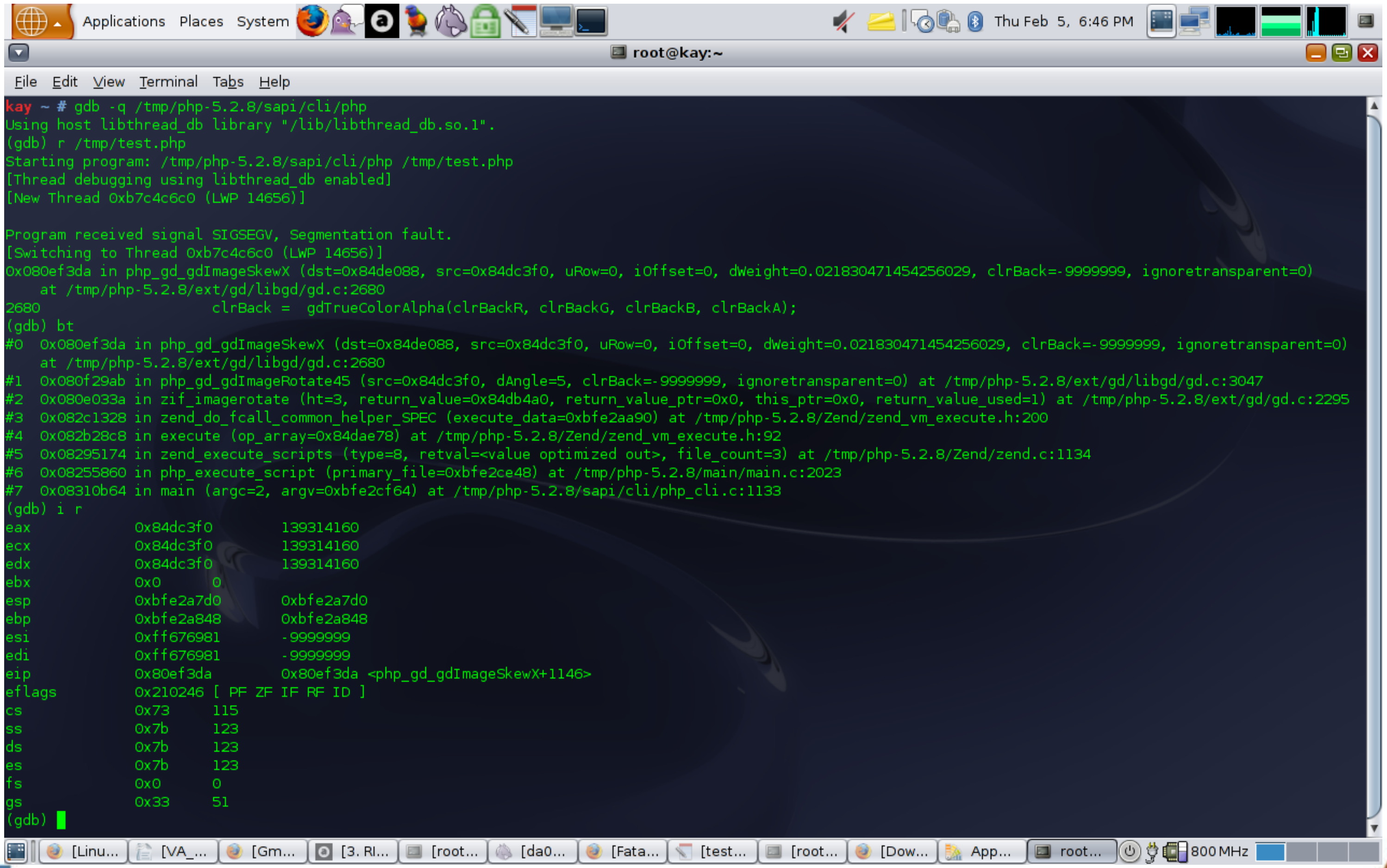
Designing/Implementing Test Case Generator in Action II

- We did not try to find the exact prototypes of PHP APIs
  - Needs more run time (less effective)
  - Faster implementation time
- It is a trade-off you might like to consider when designing test case generators
- Each function was executed multiple times
  - a random number of arguments
  - A random combination of the mentioned malformed variables
- Monitoring PHP during runtime
  - UNIX signals and signal handlers

Designing/Implementing Test Case Generator in Action III

- Packing it all together, we have
  - A list of functions
  - Some good variables that might crash PHP
  - It is not important for us how many arguments each function has
- 10,000 php files created containing
  - An include to the variables file
  - A call to a random function name
- A simple debugger was created to run /usr/bin/php with each of these files
  - The name of files that caused suspicious signals were logged
- It took us some (< 5h) hours to audit PHP
  - Retrieve the function names
  - Code a program to generate the php files
  - Code a simple debugger which ran the php binary with signal monitoring
- The run-time was less than a minute for generating the function.php files, we let each run take two seconds and the total run-time was about 8 hours
- PHP crashed a lot of times. Some of those were new security bugs (CVE-2008-5498)

Designing/Implementing Test Case Generator in Action IV



```
kay ~ # gdb -q /tmp/php-5.2.8/sapi/cli/php
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) r /tmp/test.php
Starting program: /tmp/php-5.2.8/sapi/cli/php /tmp/test.php
[Thread debugging using libthread_db enabled]
[New Thread 0xb7c4c6c0 (LWP 14656)]

Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0xb7c4c6c0 (LWP 14656)]
0x080ef3da in php_gd_gdImageSkewX (dst=0x84de088, src=0x84dc3f0, uRow=0, iOffset=0, dWeight=0.021830471454256029, clrBack=-9999999, ignoretransparent=0)
    at /tmp/php-5.2.8/ext/gd/libgd/gd.c:2680
2680                        clrBack =  gdTrueColorAlpha(clrBackR, clrBackG, clrBackB, clrBackA);
(gdb) bt
#0  0x080ef3da in php_gd_gdImageSkewX (dst=0x84de088, src=0x84dc3f0, uRow=0, iOffset=0, dWeight=0.021830471454256029, clrBack=-9999999, ignoretransparent=0)
    at /tmp/php-5.2.8/ext/gd/libgd/gd.c:2680
#1  0x080f29ab in php_gd_gdImageRotate45 (src=0x84dc3f0, dAngle=5, clrBack=-9999999, ignoretransparent=0) at /tmp/php-5.2.8/ext/gd/libgd/gd.c:3047
#2  0x080e033a in zif_imagerotate (ht=3, return_value=0x84db4a0, return_value_ptr=0x0, this_ptr=0x0, return_value_used=1) at /tmp/php-5.2.8/ext/gd/gd.c:2295
#3  0x082c1328 in zend_do_fcall_common_helper_SPEC (execute_data=0xbfe2aa90) at /tmp/php-5.2.8/Zend/zend_vm_execute.h:200
#4  0x082b28c8 in execute (op_array=0x84dae78) at /tmp/php-5.2.8/Zend/zend_vm_execute.h:92
#5  0x08295174 in zend_execute_scripts (type=8, retval=<value optimized out>, file_count=3) at /tmp/php-5.2.8/Zend/zend.c:1134
#6  0x08255860 in php_execute_script (primary_file=0xbfe2ce48) at /tmp/php-5.2.8/main/main.c:2023
#7  0x08310b64 in main (argc=2, argv=0xbfe2cf64) at /tmp/php-5.2.8/sapi/cli/php_cli.c:1133
(gdb) i r
eax            0x84dc3f0        139314160
ecx            0x84dc3f0        139314160
edx            0x84dc3f0        139314160
ebx            0x0      0
esp            0xbfe2a7d0       0xbfe2a7d0
ebp            0xbfe2a848       0xbfe2a848
esi            0xff676981       -9999999
edi            0xff676981       -9999999
eip            0x80ef3da        0x80ef3da <php_gd_gdImageSkewX+1146>
eflags         0x210246 [ PF ZF IF RF ID ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb)
```

Designing/Implementing Test Case Generator in Action V

- For xrdp the story was shorter
- We decided to change the normal traces a bit and send them to xrdp
- This method happens to be very effective when auditing a software which works with a binary protocol (explain why)
- This method is generally known as mutation based fuzzing
- Wireshark did us a nice work and the network packets are ready to be pasted to our little fuzzer

Designing/Implementing Test Case Generator in Action VI

- We started from the first packet, changed some bytes from some random places to some other random bytes
- Then the first and the second packet and after that the first three packets and so on
- We used gdb (The GNU Debugger) to monitor xrdp
- We let the auditing process take about an hour
- xrdp also crashed several times, almost all of them were new high risk vulnerabilities (CVE-2008-5902, CVE-2008-5903, CVE-2008-5904)

Vulnerability Analyzing

- The time required for debugging a vulnerability is based on:
  - Type of the vulnerability
  - Experience of the one who is responsible for this task
  - familiarity with the software under question
- Tools
  - Linux
    - gdb (or ddd)
    - IDA Pro
  - Windows
    - windbg
    - ollygdb
    - IDA Pro

Vulnerability Analyzing in Action

- It took us almost a day to understand the nature and impact of the vulnerabilities discovered in PHP which we finally came to categorize as information disclosure (e.g. read SSL private key of apache)
- Debugging xrdp vulnerabilities did not take as long, the cause was mainly the unchecked null pointers returned from malloc() and some buffer overflows (e.g. DoS or Unauthorized access)

Patching and Reporting

- Patching is normally a straight forward process when the nature of a vulnerability is discovered
- Sometimes a major change to the source code is required
- For formal reporting of the vulnerability, you should then inform the vendor (ask MITRE for a CVE)
- Normally vendors come with a newer version or a patch and ask if it is still vulnerable (and sometimes it still is!)
- Side effects might be introduced by patching a vulnerability

Patching and Reporting in Action

- In our cases we just added some checks to the program when handling input data
- On UNIX you can use diff –u source.c source_patched.c > source.patch and then distribute source.patch
- PHP guys were very cooperative and the vulnerability got patched in CVS 30 minutes later, they also used the same patch we sent to them
- xrdp developers were a bit harder to work with, after the formal reporting of the vulnerabilities to security sources, the major distributions patched it in a case basis.

Contents

- CERT and Vulnerability Discovery (**VD**)
- An Overview of Vulnerabilities
- Current Vulnerability Discovery Methods
- A New Procedure For Vulnerability Discovery
- The Proposed Procedure in Action
  – PHP (CVE-2008-5498)
  – xrdp (CVE-2008-5902 CVE-2008-5903 CVE-2008-5904)
- Conclusion

Conclusion

- There could be a lot of security bugs in a software which could be discovered and patched with a little effort
- Normally the produced testing tools could be used again with minor modifications
- Following the procedure we gave in this presentation, effective VD is possible even when there are time constraints
- It may not discover in depth vulnerabilities but the tested software could be given the minimums of security assurance

Q & A