Dynamic PHP web-application analysis

Arthur Gerkis
Ruhr University Bochum
HackPra 2012/2013

Who am I?

- independent computer security researcher - bug-hunter
- in past doing web application security analysis and pentesting
- author of several articles in russian IT-security printed magazine "Хакер"
- senseless browser bug slayer

Outline

i. About dynamic analysis
ii. Dynamic analysis today
iii. PHP extension capabilities
iv. PVT extension

So, what was dynamic analysis about?

i. About dynamic analysis

Dynamic analysis

DA is a properties inspection of running application:

1. prepare environment & run application
2. collect run-time data
3. analyse data

In general used for: profiling, code coverage, tracing, etc.

Why dynamic analysis?

- operate with real data values - is known which function which arguments has, return values, etc.
- avoid static analysis false positives (more efficient is a combination of DA & SA)
- easier to analyse obfuscated code

Why not dynamic analysis?

- single dynamic analysis can not cover all code paths
- can be slow - depends on implementation, computing powers, LoC
- may depend on environment - OS, bits, PHP versions, etc.
- may be dangerous to execute code without knowing what results will be (e.g. malicious)

DA tools implementations (general)

- code instrumentation
  - source code
  - compile-time
  - execution/run-time
- patches and extensions for compiler or interpreter
- external (e.g. system) tools

DA tools implementations (PHP)

- code instrumentation
  - source code (web-application)
  - compile-time
  - execution/run-time
- patches and extensions for compiler or interpreter
- external (e.g. system) tools

State of PHP dynamic analysis as of today

ii. Dynamic analysis today

Tools - code instrumentation

- PHP Vulnerability Hunter

  autosectools.com/PHP-Vulnerability-Scanner

  Author: John Leitch

- Saner*

  iseclab.org/papers/oakland-saner.pdf

  Authors: Davide Balzarotti, Marco Cova, Vika Felmetsger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, Giovanni Vigna

- WAFA*

  research.cs.queensu.ca/~cordy/Papers/ACD_WSE09_WAFA.pdf

  Authors: Manar H. Alalfi, James R. Cordy, Thomas R. Dean

- PHP Aspis

  https://github.com/jpapayan/aspis

  Authors: Dr Peter Pietzuch, Dr Matteo Migliavacca, Ioannis Papagiannis

* Tool was not found in public access

Tools - PHP interpreters

- Taint support for PHP

  https://wiki.php.net/rfc/taint

  Author: Wietse Venema

Taint support for PHP

news.php.net/php.internals/26979

PHP.net

From:	(Wietse Venema)	**Date:**	Fri Dec 15 15:14:48 2006
Subject:	Run-time taint support proposal		
Groups:	php.internals		

Please allow me to introduce myself briefly: I'm Wietse Venema from IBM Research, also known as the creator of the open source Postfix mail system, co-author of the Coroner's toolkit and SATAN, and the original author of TCP Wrapper. That doesn't mean everything I touch becomes gold; if it did then I'd be a very rich man now :-)

This is a proposal to add basic Perl/Ruby like tainting support to PHP: an option that is turned off by default, and that programmers may turn on at runtime to alert them when they make the common mistake of using uncleansed input with include, echo, system, open, etc. This would work with unmodified third-party extensions.

Taint support is first of all an education tool. When an alert is raised that data needs cleansing, the programmer needs to make a conscious decision. It's their job to choose the right method. I'll discuss below why I think PHP shouldn't make the decision for them.

Tools - PHP interpreters

- Taint support for PHP

  https://wiki.php.net/rfc/taint

  Author: Wietse Venema
- CORE Grasp

  grasp.coresecurity.com/

  Author: CoreLabs
- PHPrevent*

  cs.virginia.edu/nguyen/phprevent/

  Authors: Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, David Evans

* Tool was not found in public access

Tools - PHP extension

- bytekit

  https://github.com/Tyrael/bytekit

  Author: Stefan Esser
- evalhook

  php-security.org/2010/05/13/article-decoding-a-user-space-encoded-php-script/index.html

  Author: Stefan Esser
- taint

  pecl.php.net/package/taint

  Author: Xinchen Hui
- Dtrace

  pecl.php.net/package/DTrace

  Author: Wez Furlong

Tools - external tools

- strace - Linux, truss - Solaris/BSD, ktrace - OS X (< 10.5)/BSD
- DTrace - OS X/Solaris/QNX/BSD
- SystemTap, LTTng - Linux
- gdb (use PHP's .gdbinit), any other debugger

Tools - gdb example

```
arthurgerkis@ubuntu:/var/www$ gdb php -q
Reading symbols from /usr/bin/php...Reading symbols from /usr/lib/debug/usr/bin/php5...done.
done.
Really redefine built-in command "frame"? (y or n) [answered Y; input not from terminal]
Really redefine built-in command "thread"? (y or n) [answered Y; input not from terminal]
Really redefine built-in command "start"? (y or n) [answered Y; input not from terminal]
gdb $ break php_trim
Breakpoint 1 at 0x5ebbb0: file /build/buildd/php5-5.3.5/ext/standard/string.c, line 719.
gdb $ run 1.php 12345
[Thread debugging using libthread_db enabled]
-----------------------------------------------------------------------------[ regs]
     eax:00F39A60 ebx:00D9DF28  ecx:00000000  edx:00000000     eflags:00000246
     esi:0000001F edi:00A99160  esp:FFFFDB18  ebp:00000015Error while running hook_stop:
Value can't be converted to integer.

Breakpoint 1, php_trim (c=0x15 <Address 0x15 out of bounds>, len=0xd9df28, what=0x0, what_len=0x1f, re
turn_value=0xddc620, mode=0x0) at /build/buildd/php5-5.3.5/ext/standard/string.c:719
719     {
gdb $ step
-----------------------------------------------------------------------------[ regs]
     eax:00000000 ebx:0000001F  ecx:00000000  edx:00000000     eflags:00000246
     esi:00000000 edi:00A99160  esp:FFFFD9B0  ebp:00A99160Error while running hook_stop:
Value can't be converted to integer.
724             if (what) {
gdb $ info args
c = 0xa99160 "^(text/|application/xhtml\\+xml)"
len = 0x1f
what = 0x0
what_len = 0x0
return_value = 0x7fffffffdb60
mode = 0x3
gdb $
```

Tools - miscellaneous

Xdebug +(KCachegrind, PHPUnit, php-code-coverage, NetBeans IDE), XHProf, pfff

As well there should exist unknown tools - small, private and unreleased.

Our choice - PHP extension

- transparent to web-application - no influence on code and execution path
- full control over web-application - dump values of variables, trace functions, taint variables, etc.

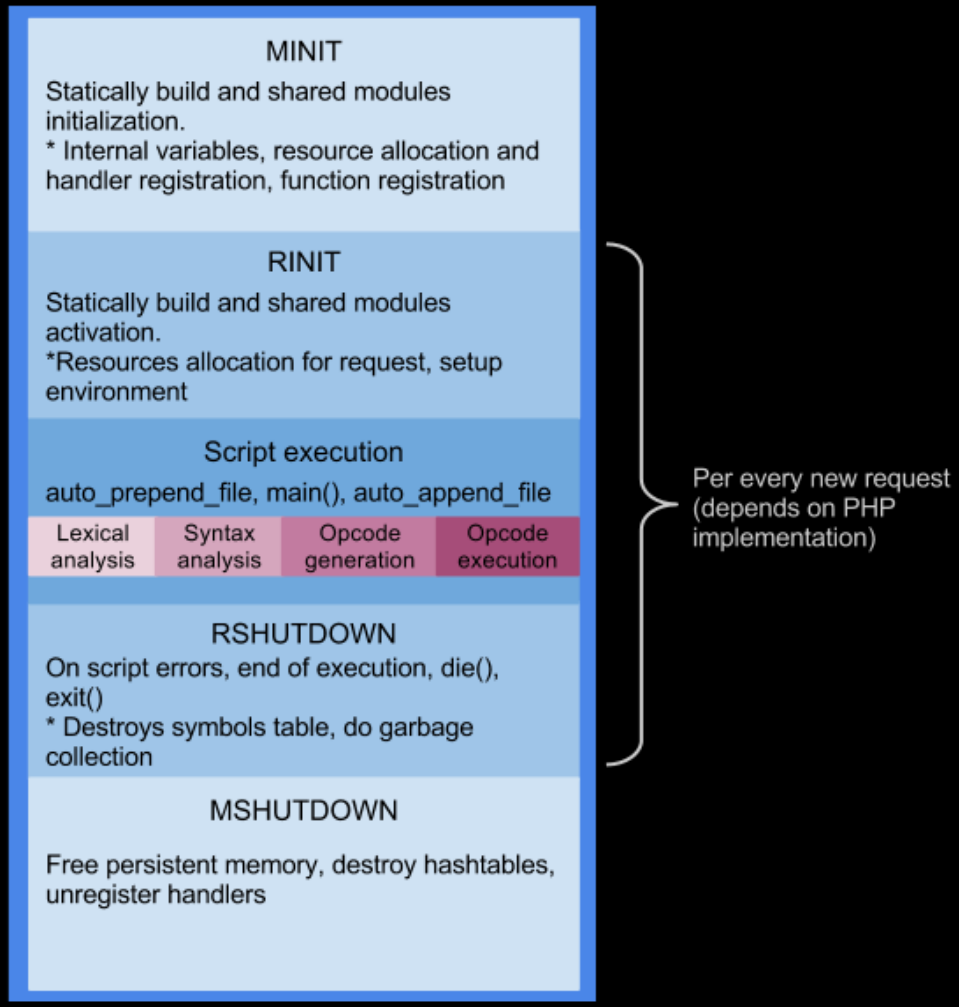
Developing PHP extension?

- no actual documentation - blog-posts, outdated book and couple of chapters:
  - Expert PHP and MySQL - Andrew Curioso, Ronald Bradford, Patrick Galbraith, 2010 (Chapter)
  - Extending and Embedding PHP - Sara Golemon, 2006
  - Advanced PHP Programming - George Schlossnagle, 2004 (Chapter)
- may be intimidating to follow PHP changes
- up-to-date source of information are another extensions source code (Suhosin, bytekit, XDebug)
  - http://lxr.php.net/ - PHP source code search via OpenGrok

What is a PHP extension capable of?

iii. PHP extension capabilities

Dissected PHP lifecycle




```
PHP_MINIT_FUNCTION(foobar)
{
    [...]
    orig_compile_string = zend_compile_string;
    zend_compile_string = foobar_compile_string;

    old_execute  = zend_execute;
    zend_execute = foobar_execute;

    old_zend_execute_internal = zend_execute_internal;
    zend_execute_internal = foobar_execute_internal;

    return SUCCESS;
}

PHP_MSHUTDOWN_FUNCTION(foobar)
{
    [...]
    zend_compile_string = orig_compile_string;
    zend_execute = old_execute;
    zend_execute_internal = old_zend_execute_internal;
    [...]
    return SUCCESS;
}

PHP_RINIT_FUNCTION(foobar)
{
    [...]
    return SUCCESS;
}

PHP_RSHUTDOWN_FUNCTION(foobar)
{
    [...]
    return SUCCESS;
}
```

Handle function entry and exit

Register every executing function and have access to all data functions works with:

- trace execution path and generate call graph later
  - understand application architecture
  - explore application executed paths and detect yet unexplored areas
- intercept identifiers when passing data to any function

Handle function entry and exit

```
// execute_internal() doesn't catch nested internal function calls (calllbacks)
static void foobar_execute_internal(zend_execute_data *execute_data_ptr, int return_value_used
TSRMLS_DC)
{
    [...]
    // Here Zend internal functions gets executed
    // Work with stuff like opcodes, variables, handle function entries.
    execute_internal(execute_data_ptr, return_value_used TSRMLS_CC);
    // Here is possible to handle function exits.
}

static void foobar_execute(zend_op_array *op_array TSRMLS_DC)
{
    [...]
    // Here user functions gets executed
    // Work with stuff like opcodes, variables, handle function entries.
    old_execute(op_array TSRMLS_CC);
    // Here is possible to handle function exits.
}
```

Implement functions and classes

Implementing own PHP internal classes and functions allow to extend PHP functionality for different use:

- fighting with bottlenecks, optimize execution time
- utilize OS-specific functionality
- extend web-application capabilities
- provide debugging and profiling facilities

Work with opcode

PHP allows complete control over every opcode:

- dump opcodes on the fly and observe them later for low-level analysis (e.g. for obfuscated code)
- set opcode handler for complete control over application

Work with opcode

```
static void php_taint_register_handlers(TSRMLS_D) {
      zend_set_user_opcode_handler(ZEND_ECHO, php_taint_echo_handler);
      [...]
}

static int php_taint_echo_handler(ZEND_OPCODE_HANDLER_ARGS) {
    zend_op *opline = execute_data->opline;
      [...]
      return ZEND_USER_OPCODE_DISPATCH;
}

PHP_MINIT_FUNCTION(foobar)
{
      [...]
      php_taint_register_handlers(TSRMLS_C);

      return SUCCESS;
}
```

Hook dynamically evaluated strings

Catch every dynamically executed string and log it - see what happens inside of

- eval(), create_function()
- assert(),
- preg_replace() with "e"

Hook dynamically evaluated strings

```
static zend_op_array *evalhook_compile_string(zval *source_string, char *filename TSRMLS_DC)
{
    [...]
    len = Z_STRLEN_P(source_string);
    copy = estrndup(Z_STRVAL_P(source_string), len);
    [...]
    return orig_compile_string(source_string, filename TSRMLS_CC);
}
```

Set Zend extension callbacks

Zend provides possibility to set various handlers for more fine-grained control:

- statement handler, allows:
  - single stepping through code
  - profiling
  - implement stepping debugger
- function entry/exit handlers
- op_array manipulation

Set Zend extension callbacks

```
// Set in our extension
ZEND_DLEXPORT zend_extension zend_extension_entry = {
    "Foobar extension",
    FOOBAR_VERSION,
    "Author",
    NULL,
    "Copyright (c)",
    NULL,   // foobar_zend_startup
    NULL,   // foobar_zend_shutdown
    NULL,   // activate_func_t
    NULL,   // deactivate_func_t
    NULL,   // message_handler_func_t
    NULL,   // op_array_handler_func_t
    statement_handler,   // statement_handler_func_t
    NULL,   // fcall_begin_handler_func_t
    NULL,   // fcall_end_handler
    NULL,   // op_array_ctor_func_t
    NULL,   // op_array_dtor_func_t
    STANDARD_ZEND_EXTENSION_PROPERTIES
};
```

```
// Defined in Zend/zend_extensions.h
struct _zend_extension {
    char *name;
    char *version;
    char *author;
    char *URL;
    char *copyright;

    startup_func_t startup;
    shutdown_func_t shutdown;
    activate_func_t activate;
    deactivate_func_t deactivate;

    message_handler_func_t message_handler;

    op_array_handler_func_t op_array_handler;

    statement_handler_func_t statement_handler;
    fcall_begin_handler_func_t fcall_begin_handler;
    fcall_end_handler_func_t fcall_end_handler;

    op_array_ctor_func_t op_array_ctor;
    op_array_dtor_func_t op_array_dtor;

    int (*api_no_check)(int api_no);
    int (*build_id_check)(const char* build_id);
    [...]
};
```

Tasks for PHP extension

- assist in debugging - XDebug, vld
- hardening PHP - Suhosin
- execution time optimization - APC, XCache
- web-application security evaluation - bytekit, evalhook, taint
- protective measures - Zend Guard, ionCube
- etc.

Introducing PVT

iv. PVT extension

New PHP dynamic analysis tool

Named PVT - PHP Vulnerability Tracer:

- the idea of PVT is to provide tools to assist in web-application security audit
- the aim of PVT is to be transparent to web-application, fully-automated, easy to use and highly customizable - achieved via being PHP extension.

PVT - Swiss knife

- draws dynamic code execution graphs (allows code navigation)
- hooks all eval'd strings
- catches your marker in arguments of function or just every argument in every function
- can dump chosen or all variables
- opcode dumper for low-level analysis
- settings for each module

PVT - Swiss knife

```
1 [pvt]
2 zend_extension=/var/www/htdocs/PVT/trunk/pvt/modules/pvt.so
3 pvt.log_file = "/var/www/htdocs/PVT/logs"
4 pvt.log_one_folder = On
5 ; Modes 'a' or 'w' only
6 pvt.log_write_mode = "a"
7 ; All the following values can be changed dynamically
8 ;
9 pvt.graph_fold = On
10 pvt.count_stat = Off
11 ; [1] Just trace all functions
12 pvt.trace_func = Off
13 ; [2] Dump opcodes
14 pvt.dump_ops = Off
15 ; [3] Just log changes of variables
16 pvt.dump_vars = On
17 pvt.dump_vars_list = ""
18 pvt.dump_vars_all = On
19 ; [4] Search for string in evaluated strings
20 pvt.eval_hook = Off
21 pvt.eval_hook_all = On
22 pvt.eval_marker = ""
23 pvt.eval_hook_len = 256
24 pvt.eval_unique = On
25 ; [5] Search for string in arguments passed to functions
26 pvt.catch_marker = Off
27 pvt.catch_all = On
28 pvt.catch_marker_val = ""
29 pvt.catch_len = 256
30 pvt.catch_funcs = ""
31
```

** namings may change later*

Sounds good?

- may be slow as hell if you switch too many modules or run it on heavy web-application
- works only on Linux*
- works only on PHP 5.2/5.3
- may be not very comfortable in use - logs are plain text files, needs *dot*, requires *Perl* ...

Statistics

Statistics shown are for Wordpress 3.4. (opening simple pages like index.php)

	Title	Time, seconds
1	Without PVT	0.14
2	All modules switched On	16.67
3	dump_ops = On	9.45
4	All modules On except dump_ops	6.45
5	catch_marker = On	4.79
6	dump_vars = On	1.18
7	trace_func = On	0.64
8	eval_hook = On	0.16

Demo

In perspective

- speed optimization (priority)
- add variable tracing
- connect logs and graph
- data tainting (?)
- opcode graphs
- convenient logs management
- find and eliminate bugs

You can help with ideas and bug reports!

Acknowledgments

- Stefan Esser
- Vladimir Vorontsov

References

- https://github.com/ax330d/pvt
- https://wiki.php.net/rfc/taint
- https://wiki.php.net/internals/references
- www.smartflow.org/aspis
- grasp.coresecurity.com/
- www.cs.virginia.edu/phprevent/
- https://github.com/Tyrael/bytekit
- http://www.cs.ucsb.edu/~rusvika/papers/ssp08_saner.pdf
- research.cs.queensu.ca/~cordy/Papers/ACD_WSE09_WAFA.pdf
- sebastian-bergmann.de/archives/871-bytekit-cli.html
- solaris.reys.net/dtrace-php-scripts/
- solaris.reys.net/debugging-php-with-dtrace-part-2/
- https://blogs.oracle.com/shanti/entry/dtrace_support_for_php
- wezfurlong.org/blog/2005/aug/dtracing-php-on-solaris/
- PHP Tainted variables: An idea whose time has come - Wietse Venema, 2008
- Static and Dynamic Analysis for PHP Security - V.C.Sreedhar, 2006
- Static and Dynamic Analysis at Ning - David Sklar, 2008
- Analysing PHP Code - Sebastian Bergmann, 2009
- PHP Extension Writing - Marcus Borger, Wez Furlong, Sara Golemon, 2005

Questions?

You can reach me on twitter.com/ax330d, or e-mail to user ax330d on the gmail server.