Vulnerability Discovery in Closed Source / Bytecode Encrypted PHP Applications

Stefan Esser

Power Of Community
November 2008
Seoul

SektionEins

Who am I?

Stefan Esser

- from Cologne/Germany
- 10 years in Information Security
- 8 years PHP Core Developer
- Month of PHP Bugs & Suhosin
- Head of Research & Development at SektionEins GmbH

What the talk is about...

- PHP Source Code Encryption
- PHP Bytecode Recovery & Visualisation
- Vulnerability Discovery in Bytecode
- Decompilation

Why PHP Source Code Encryption

closed source PHP applications

- as protection of intellectual property
- as protection of software license systems
- as protection against tampering
- to hide own IP violations

PHP Source Code Encryption (I)

This talk is not about user-space runtime encryption

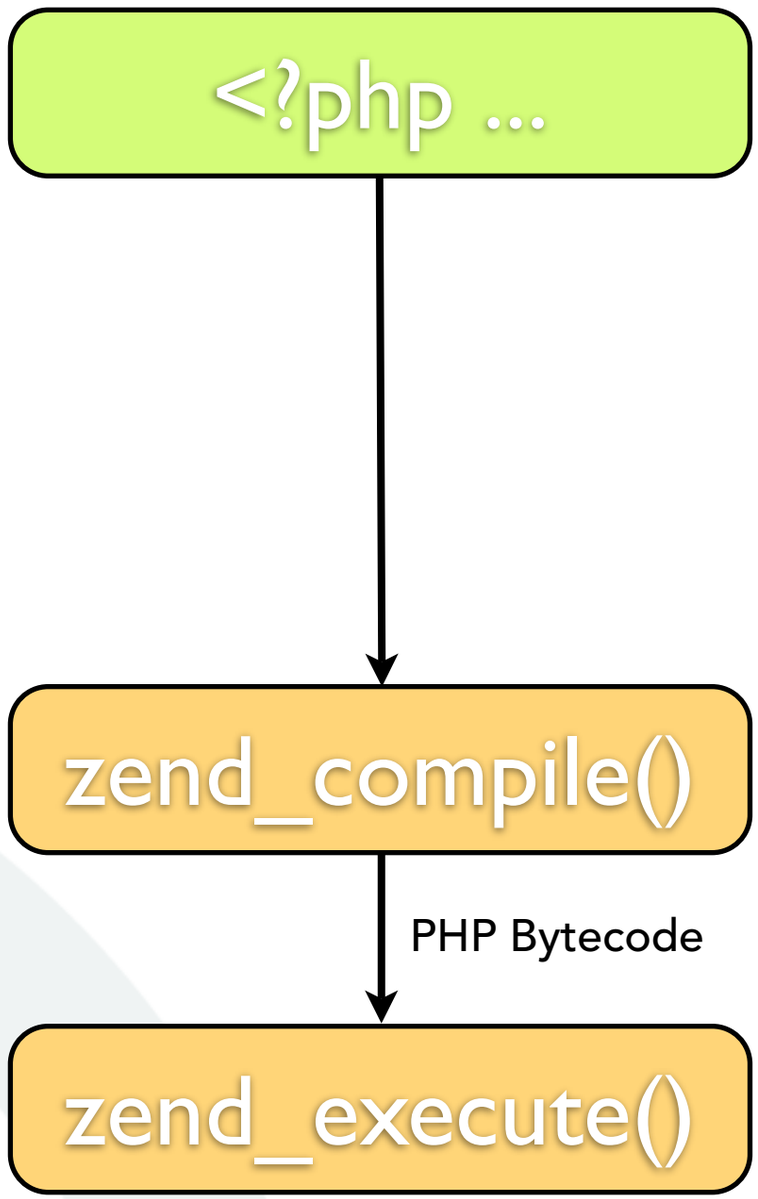
```
<?php $z=_;$x=32;$z.=decode;$y=base;$x<<=1;$f=$y.$x.
$z;eval($f(JHggPSJUaGUgZmlyc3Qgb25lIHdobyBkZWNyeXB0cyB0aGlzIGdldH
MgYSBib3R0bGUgb2Ygc29qdSI7ZWNobyAiSGVsbG8gV29ybGRcbiI7));?>
```

such encryption is defeated by hooking eval() and dumping the source code

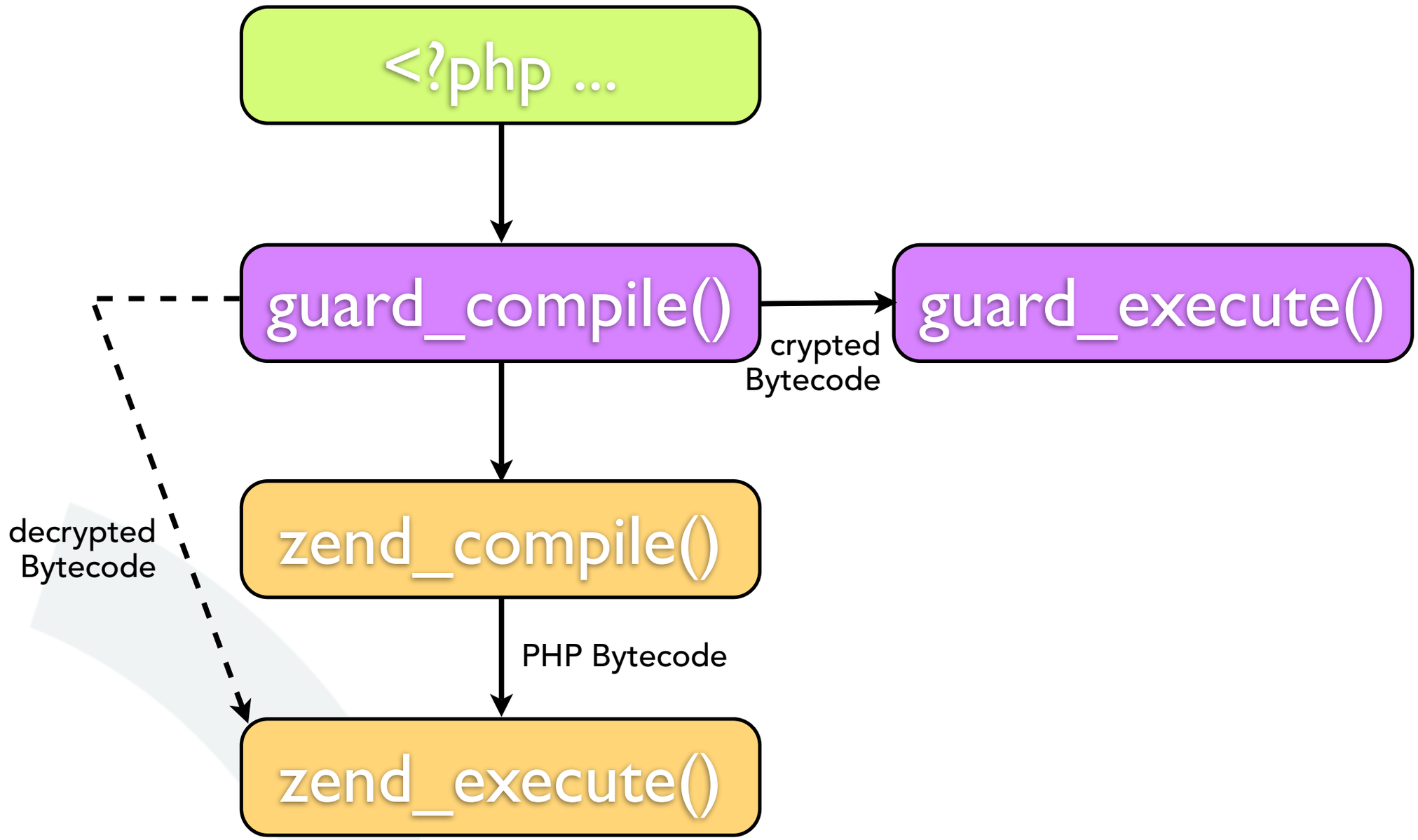
PHP Source Code Encryption (II)

- more secure encryption
  - needs to be implemented below user-space
  - should eliminate source code completely
  - requires loadable PHP modules
  - works directly on PHP bytecode

PHP Source Code Encryption (III)



PHP Source Code Encryption (IV)



PHP Source Code Encryption (V)

```
<?php @Zend;
4147;
/* ##Ò!This is not a text file!fiÌ#  */


print <<<EOM
<html><body><a href="http://www.zend.com/store/products/zend-safeguard-suite.php"><img border="0" src="http://
www.zend.com/images/store/safeguard_optimizer_img.gif" align="right"></a><center><h1>Zend Optimizer not installed</
h1></center><p>This file was encoded by the <a href="http://www.zend.com/products/zend_guard">Zend Guard</a>. In
order to run it, please install the <a href="http://www.zend.com/products/zend_optimizer">Zend Optimizer</a>
(available without charge), version 3.0.0 or later. </p><h2>Seeing this message instead of the website you
expected?</h2>This means that this webserver is not configured correctly. In order to view this website properly,
please contact the website's system administrator/webmaster with the following message:<br><br><tt>The component
"Zend Optimizer" is not installed on the Web Server and therefore cannot service encoded files. Please download and
install the Zend Optimizer (available without charge) on the Web Server.</tt><br><br><b>Note</b>: Zend Technologies
cannot resolve issues related to this message appearing on websites not belonging to <a href="http://
www.zend.com">Zend Technologies</a>. <h2>What is the Zend Optimizer?</h2><p>The Zend Optimizer is one of the most
popular PHP plugins for performance-improvement, and has been available without charge, since the early days of PHP
4. It improves performance by scanning PHP's intermediate code and passing it through multiple Optimization Passes
to replace inefficient code patterns with more efficient code blocks. The replaced code blocks perform exactly the
same operations as the original code, only faster. </p><p>In addition to improving performance, the Zend Optimizer
also enables PHP to transparently load files encoded by the Zend Guard. </p><p>The Zend Optimizer is a free product
available for download from <a href="http://www.zend.com">Zend Technologies</a>. Zend Technologies also developed
the PHP scripting engine, known as the <a href="http://www.zend.com/products/zend_engine">Zend Engine</a>.</p></
body></html>
EOM;
exit();
__halt_compiler();
?>

#2004102401##1##46#0,##s)#¡≤uoªÏ#ÈJî£P9óÍ=≈##HfÛC#!ßø0ŸZ# -#;ÔõÉ#17419##76188#x˜
ü2Ì}#ò\GuÊıh........
```

Available PHP Bytecode Encryptors

- ZendGuard - http://www.zend.com
- ionCube PHP Encoder - http://www.ioncube.com
- SourceGuardian - http://www.sourceguardian.com
- ... several white-labeled encryptors

PHP Source Code Encryption Levels

- Encryption
  - PHP bytecode is encrypted
- Obfuscation
  - PHP bytecode / function- and variable names are obfuscated
- Protection
  - anti hooking, anti decryption techniques are used

But what is PHP Bytecode?

PHP Bytecode - Opcode Arrays

- compiled execution unit (op_array)
- function name, filename and lines
- type, flags
- number and information about function parameters
- opcode lines (oplines)
- ...

PHP Bytecode - Oplines

- opcode
- result(-operand)
- operand 1
- operand 2
- extension
- PHP source code linenumber

```
SUB                ~0, 0, 4
ASSIGN             !0, ~0
SEND_VAR           !0
DO_FCALL           $2, 'abs'
MUL                ~3, $2, 3
ASSIGN             !0, ~3
POST_INC           ~5, !0
FREE               ~5
ECHO               !0
RETURN             1
```

PHP Bytecode - Operands

- five operand types
  - CONST - constant values
  - TMP - temporary variable (~)
  - VAR - variables ($)
  - CV - compiled variables (!)
  - UNUSED - unused operand

PHP Bytecode - Executor

- loops through oplines
- dispatches to opcode handlers
- uses lookup table for opcode handlers
- 25 opcode handlers per opcode
- for every combination of operand types

Weaknesses in PHP Bytecode Encrypters

- older crypters
  - do not replace executor
  - no obfuscation
  - everything decrypted at zend_execute()
- modern crypters
  - obfuscation requires full encryption
  - replaced executor ripped from PHP

Generic Unpacker - Idea

- find opcode handler table of replacement executor
- replace all opcode handlers with recording handler
- execute everything and record decrypted operands
- repair op_arrays
- dump op_arrays

Generic Unpacker - Find Opcode Handler Table

- start at replacement zend_execute() address
- table aligned in memory - NULL pointer infront
- starts with 25x NOP handler
- continues with something else
- search for NULL, 25x X followed by Y

Generic Unpacker - Opcode Handler Override

- detect end of table (currently hardcoded)
- backup original handlers
- replace with recording handler
  - that records decrypted operands
  - that continues with next opline

Generic Unpacker - Tracing

- enumerate all op_arrays of methods and functions
- trigger execution of every single op_array
- store decrypted recorded op_arrays
- restore backup of opcode handlers

Generic Unpacker - Repairing

- some encrypters implement own opcodes
  - e.g. to speed up function calls
- Generic Unpacker needs to
  - ignore and remove unknown opcodes
  - replace custom opcodes with plain PHP ones
    - ➡ requires reverse engineering

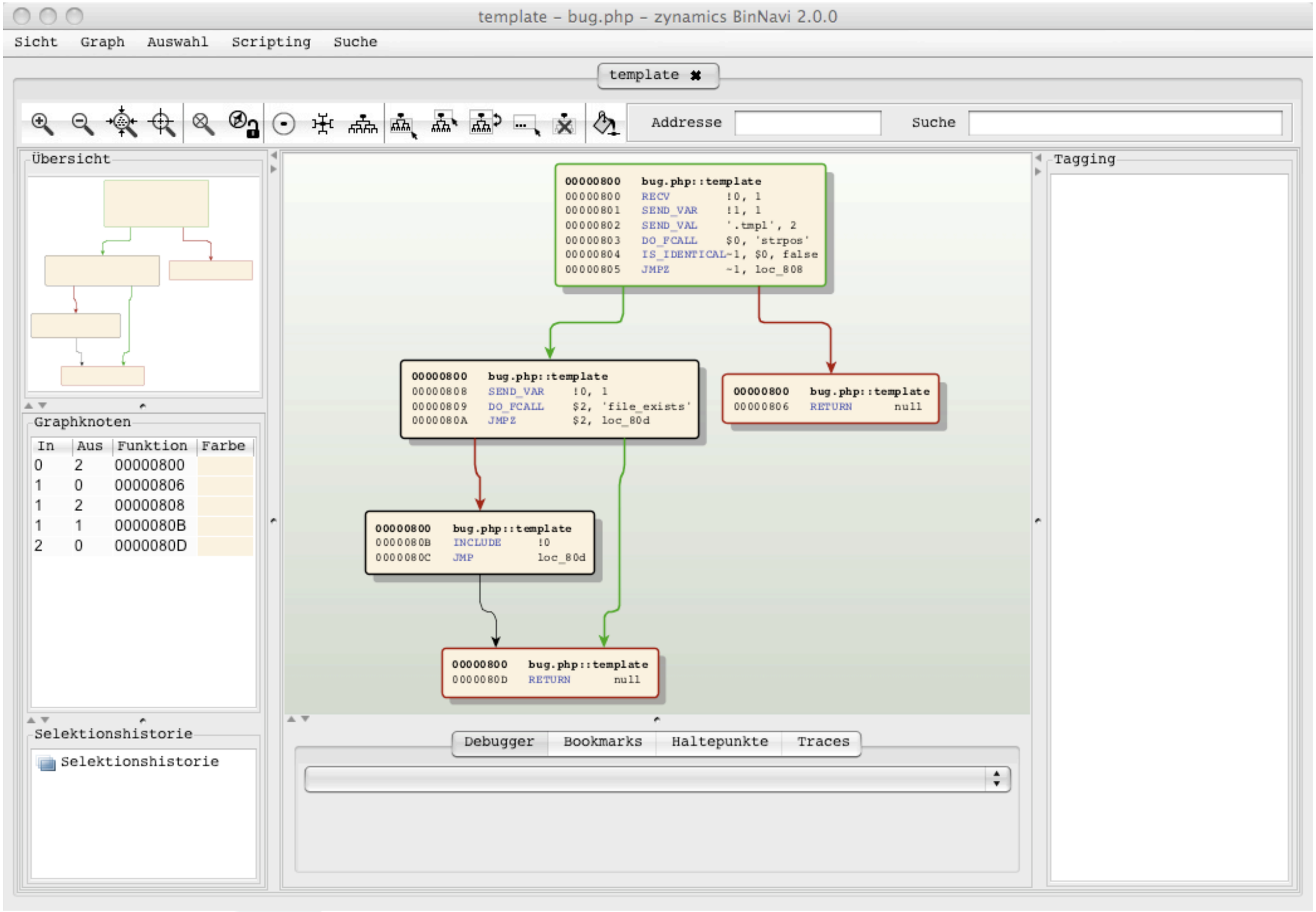
Vulnerability Discovery in PHP Bytecode

- PHP Bytecode Visualization and Navigation
- PHP Bytecode Decompilation
- PHP Bytecode Analysis

PHP Bytecode Visualization

- build up code flow graphs from bytecode
- build up callgraphs
- graphical representation of disassembly
- manual bytecode audit
- php2sql script to export bytecode to Binnavi

PHP Bytecode Visualization (in Binnavi 2.0)



PHP Bytecode Decompilation (I)

- PHP decompiler implemented based on Reverse Compilation Thesis by Cristina Cifuentes
- implementation is straight forward
- mainly consists of graph structuring
- and code generation algorithms

PHP Bytecode Decompilation (II)

- benefits from huge amount of Zend Engine meta data
- completely recovers non obfuscated code
- slight problems with protected members in classes
- obfuscated code still easier to read than bytecode

PHP Bytecode Decompilation - Example (I)

Original function with RFI

```
function template($tmpl)
{
    if (strpos($tmp, '.tmpl') === false) return;
    if (file_exists($tmpl)) {
        include $tmpl;
    }
}
```

PHP Bytecode Decompilation - Example (II)

Decompiled function with RFI

```
function template($tmpl = null)
{
    if (strpos($tmp, '.tmpl') === false) {
        return null;
    } else {
        if (file_exists($tmpl)) {
            include($tmpl);
        }
        return null;
    }
}
return 1;
```

PHP Bytecode Analysis

- manual analysis is time consuming
- wish to automate vulnerability discovery
- direct analysis of the PHP bytecode
- from simple pattern scanning to static or dynamic dataflow analysis

PHP Bytecode Analysis - Pattern Scanning

- example - find vulnerabilities by searching for
  - INCLUDE_OR_EVAL - RFI or eval() injection problems
  - PRINT / ECHO - cross site scripting problems
- eliminate occurrences with CONST operands
- scan for usage of FETCH_DIM_R on PHP's superglobals

➡ candidate for manual inspection

PHP Bytecode Analysis - Dataflow analysis (I)

- dataflow analysis to keep track of constant, value and type propagation
- simple approach considers global variables read only
- and ignores references
- sometimes heuristic class type detection necessary
- searches for user input in hotspots
  - dangerous Zend Engine opcodes - INCLUDE_OR_EVAL
  - dangerous functions - mysql_query()

PHP Bytecode Analysis - Dataflow analysis (II)

- algorithm starts with outermost functions
- user input in hotspots is marked as possible vulnerability
- function parameters in hotspots add the function to the dangerous function list
- repeated until all functions in callgraph are processed

Thank you for listening...

QUESTIONS ?