Towards Scalable Symbolic Execution in Interpreted Languages

George Argyros
argyros@cs.columbia.edu
Columbia University

Theofilos Petsios
theofilos@cs.columbia.edu
Columbia University

Dimitris Mitropoulos
dimitro@aueb.gr
Columbia University

Yunhui Zheng
zhengyu@us.ibm.com
IBM T.J. Watson Research Center

Angelos D. Keromytis
angelos@cs.columbia.edu
Columbia University

Junfeng Yang
junfeng@cs.columbia.edu
Columbia University

Abstract

Symbolic execution is a popular program analysis technique which is widely used in order to detect security vulnerabilities. The basic idea behind symbolic execution is to execute a program using a symbolic input instead of concrete values and replacing operations on concrete values with symbolic operations. Symbolically executing a program allows the generation of formulas which describe the behavior of different execution paths of the program under test. Solving these formulas using a constraint solver allows the generation of inputs which will exercise different paths in the program and the detection of violations of security properties such as memory safety. In the context of web applications, symbolic execution has being used in order to detect code injection and logic vulnerabilities and to generate exploits for security vulnerabilities. Nevertheless, to date, there is no complete system that is able to handle all the complex features present in modern web applications, and therefore being able to scale to large software projects.

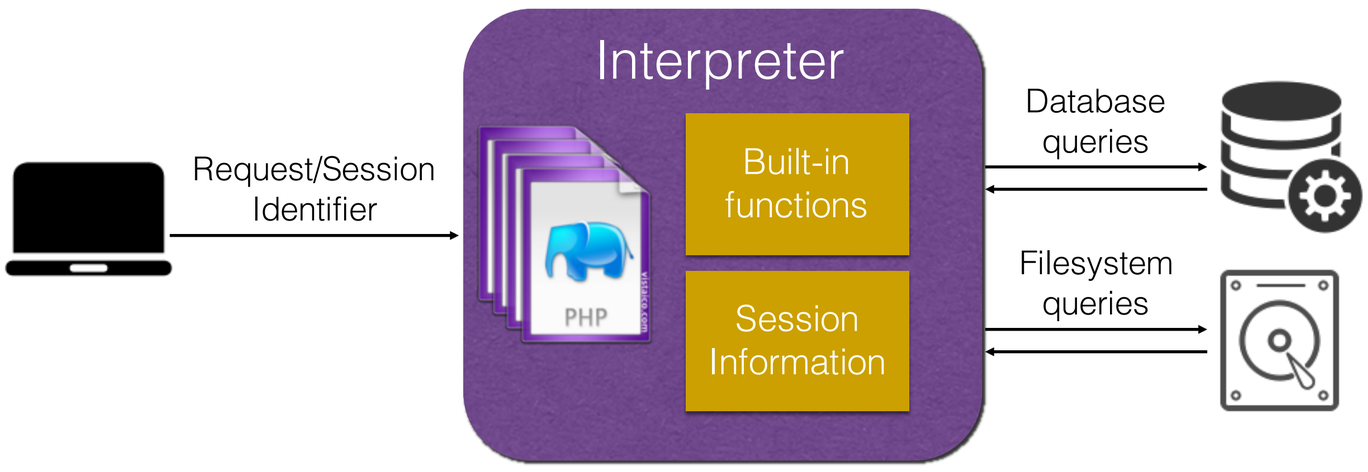
In this work, we present the design and implementation of Arpegio, a symbolic execution engine for PHP, which is designed to scale to large PHP applications. We describe how we address numerous challenges present in modeling complex web application features such as symbolically evaluating database queries, handling built-in functions and modeling complex data structures such as multidimensional hash tables. Another important contribution of our system is the introduction of symbolic types and operator overloading in the constraint solver which allows us to precisely model the dynamic types of variables and moreover, to exploit state-of-the-art string solvers in order to boost the efficiency of our system. Using Arpegio, we can test PHP applications for various security vulnerabilities ranging from code injection to logic vulnerabilities, as well as discover various other functionality problems.

RSAConference Security Scholar

Towards scalable symbolic execution in interpreted languages

George Argyros
Columbia University

Modern web applications




```php
<?php

/* i,j are symbolic variables */
$i = $_REQUEST['i'];
$j = $_REQUEST['j'];

$arr["foo"] = "bar";
$arr["bar"][10] = "secret";
$arr["bar"]["a"] = 30;
$arr[TRUE] = "100";

if ($arr[$i][$j] == "secret") {
    vulnerable_function();
}

?>
```

Arrays storing elements of different types.

Multidimensional arrays and hash tables.

Overloaded comparison operators.

How can we build a scalable symbolic execution engine by utilizing modern string solvers?

Database model

- SQL Queries are translated to PHP code and executed symbolically.
- Taint tracking used to precisely track symbolic values through parsing.

"SELECT name from users where age > 25;"

```php
<?php
$__arpegio_tmp_0 = array();
foreach ($__arpegio_db_array[$__arpegio_cur_db]["users"]
                as $value_0) {
        if ($value_0['age'] > 25) {
                array_push($tmp_0, $value_0);
        }
}

$tmp_result = array();
$tmp_result['name'] = array_column($tmp_0, 'name');
$result = array();
for ($idx = 0; $idx < count($tmp_result['name']);
       $idx++) {
        array_push($result,
           array("name" => $tmp_result['name'][$idx]));
}
return $result;
?>
```

Database converted to a PHP array

Multidimensional arrays

- Special optimizations for handling multidimensional arrays and hash tables.
- Support for hash table iterators.

Results

- Currently testing small CMS apps.
- Planning to expand on large apps (Wikimedia, Wordpress, etc.).

API for function modeling

- Simple interface with the solver in order to develop built-in function models.

```php
<?php
function arpegio_summary_str_replace($search, $replace, $str) {
        $ret_value = '';
        $replace_length = arpegio_z3_strlen($replace);
        $str_length = arpegio_z3_strlen($str);
        while (($index = arpegio_z3_indexof($str, $search)) != -1) {
                $repl_str = arpegio_z3_replace($str, $search, $replace);
                $ret_value .= arpegio_z3_substr($repl_str, 0,
                                $index + $replace_length);
        $str = arpegio_z3_substr($repl_str,
            $index + $replace_length,
            arpegio_z3_strlen($repl_str));
        }
        $ret_value .= $str;
        return $ret_value;
}
?>
```

arpegio_z3_* functions are directly supported by the solver.

Symbolic data types

- Variable type is symbolic and is set by the solver during constraint solving.
- Each variable is represented as a union of different types along with a type_selector variable within the solver.
- Use symbolic data types in order to implement operator overloading in the solver.

```
function is_equal(v1, v2) {
  if (v1.type == STR and v2.type == INT)
       return is_equal_str_int(v1.str, v2.int)
  if (v1.type == STR and v2.type == BOOL)
       return is_equal_str_bool(v1.str, v2.bool)
  ...
}
```

Pseudocode for constraints used for equality checking