Università degli Studi di Milano
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica e Comunicazione

A hybrid analysis framework for detecting web application vulnerabilities

Mattia Monga *Roberto Paleari* Emanuele Passerini

SESS 2009

Web applications

- many applications adopt the web paradigm: client-server model + HTTP protocol
- web servers are augmented with modules for the execution of server-side code

Security issues

- web applications are known to be subject to different attacks (*e.g.*, SQLI and XSS)
- $\sim$ 60% of software vulnerabilities are specific to web applications

Root cause

insufficient sanitization of user-supplied input

Introduction

Web applications

- many applications adopt the web paradigm: client-server model + HTTP protocol
- web servers are augmented with modules for the execution of server-side code

Security issues

- web applications are known to be subject to different attacks (*e.g.*, SQLI and XSS)
- $\sim$ 60% of software vulnerabilities are specific to web applications

Root cause

insufficient sanitization of user-supplied input

Taint analysis of web applications

How it works?

1. data from untrusted sources are marked as *tainted*
2. *propagation* of the “taint” attribute
3. alert if tainted data with malicious characters reach a *sink*
4. sanitization: tainted $\longrightarrow$ *untainted*

Static analysis

- complete
- no run-time overhead
- overly conservative: results can be imprecise

Dynamic analysis

- accurate results
- incomplete
- high overhead ($\sim$30%)

Taint analysis of web applications

How it works?

1. data from untrusted sources are marked as *tainted*
2. *propagation* of the "taint" attribute
3. alert if tainted data with malicious characters reach a *sink*
4. sanitization: tainted $\longrightarrow$ *untainted*

Static analysis

- complete
- no run-time overhead
- overly conservative: results can be imprecise

Dynamic analysis

- accurate results
- incomplete
- high overhead (∼30%)

A hybrid approach

Goal

design and develop a *hybrid analysis framework* in order to obtain:

- accurate results
- low run-time overhead

Our idea

1. **off-line analysis**
   - build a static model of the whole application
   - identify dangerous code statements
2. **on-line analysis**
   - dynamic taint-analysis over dangerous statements

A hybrid approach

Goal

design and develop a *hybrid analysis framework* in order to obtain:

- accurate results
- low run-time overhead

Our idea

1. **off-line analysis**
   - build a static model of the whole application
   - identify dangerous code statements
2. **on-line analysis**
   - dynamic taint-analysis over dangerous statements

Motivating example

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET[’product_id’])) {
  $a = $_GET[’product_id’];
  get_product($a);
} else {
  $msg = ’Invalid request’;
  echo $msg;
}
```

Motivating example

```
1  function get_product($id) {
2    $q = "SELECT ... WHERE id=$id";
3    mysql_connect(...);
4    $res = mysql_query($q);
5  }

6  if(isset($_GET['product_id'])) {
7    $a = $_GET['product_id'];
8    get_product($a);
9  }  else {
10   $msg = 'Invalid request';
11   echo $msg;
12 }
```

Vulnerability

- SQL injection
- control-dependent on condition at line 6

Motivating example

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

Off-line analysis

- identify dangerous statements

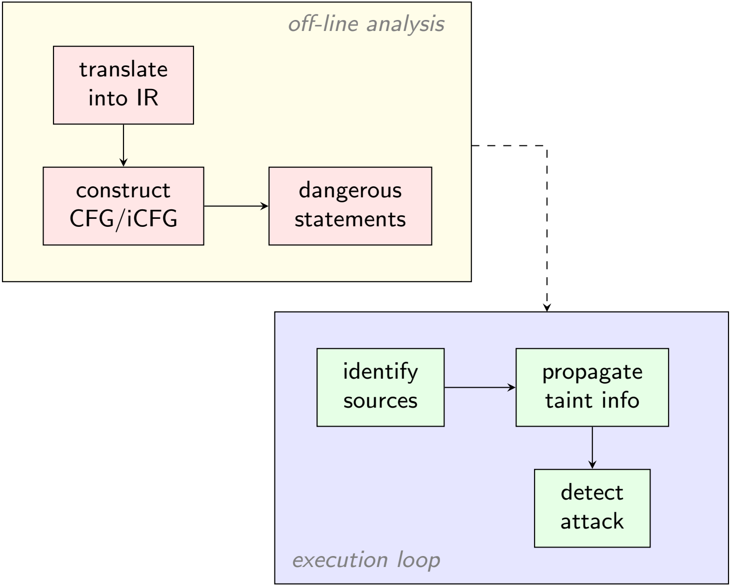
Motivating example

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

On-line analysis

- taint-propagation only over dangerous statements

Phan: PHP Hybrid Analyzer



Off-line analysis
Translation into IR

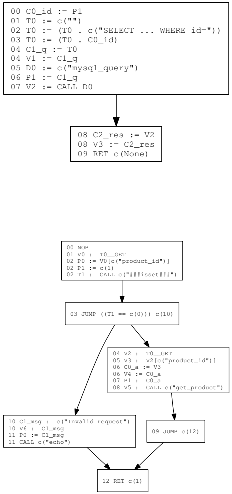
```
 6  if(isset($_GET['product_id'])) {
 7    $a = $_GET['product_id'];
 8    get_product($a);
 9  } else {
10    $msg = 'Invalid request';
11    echo $msg;
12  }
```

```
 6  V0 := T0__GET
 6  P0 := V0[c("product_id")]
 6  P1 := c(1)
 6  T1 := CALL c("isset")
 6  JUMP ((T1 == c(0))) c(10)
 7  V2 := T0__GET
 7  V3 := V2[c("product_id")]
 7  C0_a := V3
 7  V4 := C0_a
 8  P1 := C0_a
 8  V5 := CALL c("get_product")
 9  JUMP c(12)
10  C1_msg := c("Invalid...")
10  V6 := C1_msg
11  P0 := C1_msg
11  CALL c("echo")
12  RET c(1)
```

Intermediate language

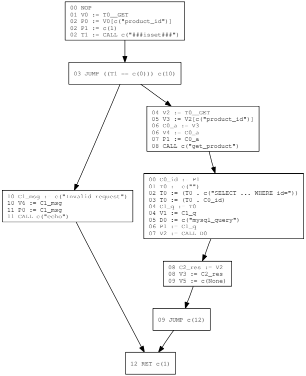
- RISC-like instructions
- 5 instruction types, 4 expression types

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```



Off-line analysis
iCFG construction

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```



- constant propagation to handle iCTI
- handling of inclusion statements

Off-line analysis

Identification of dangerous statements

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

- identify sources and sinks
- find paths from sources to sinks
- compute backward slice over sinks arguments
- flag only dangerous statements

Off-line analysis

Identification of dangerous statements

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

- identify sources and sinks
- find paths from sources to sinks
- compute backward slice over sinks arguments
- flag only dangerous statements

ignore sinks with constant input arguments

Off-line analysis

Identification of dangerous statements

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

- identify sources and sinks
- find paths from sources to sinks
- compute backward slice over sinks arguments
- flag only dangerous statements

Off-line analysis

Identification of dangerous statements

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

- identify sources and sinks
- find paths from sources to sinks
- compute backward slice over sinks arguments
- flag only dangerous statements

Off-line analysis
Identification of dangerous statements

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

- identify sources and sinks
- find paths from sources to sinks
- compute backward slice over sinks arguments
- flag only dangerous statements

On-line analysis
Dynamic taint analysis

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

On-line analysis

1. monitor only dangerous statements
2. taint-propagation
3. alert when tainted data reaches a sensitive sink

On-line analysis

Dynamic taint analysis

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

On-line analysis

1. monitor only dangerous statements
2. taint-propagation
3. alert when tainted data reaches a sensitive sink

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

On-line analysis

1. monitor only dangerous statements
2. taint-propagation
3. alert when tainted data reaches a sensitive sink

On-line analysis
Dynamic taint analysis

```
function get_product($id) {
  $q = "SELECT ... WHERE id=$id";
  mysql_connect(...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

On-line analysis

1. monitor only dangerous statements
2. taint-propagation
3. alert when tainted data reaches a sensitive sink

SQL injection

```
fun                                  ) {
  $q                                   id=$id";
  mysql_co          ...);
  $res = mysql_query($q);
}

if(isset($_GET['product_id'])) {
  $a = $_GET['product_id'];
  get_product($a);
} else {
  $msg = 'Invalid request';
  echo $msg;
}
```

On-line analysis

1. monitor only dangerous statements
2. taint-propagation
3. alert when tainted data reaches a sensitive sink

Implementation

Off-line module

- PHP extension module
- bytecode to IR translator
- IR analysis modules

▶ $\sim$ 6000 Python LoC + $\sim$ 1500 C LoC

On-line module

- hooks inside the Zend VM
- self-contained module (easily portable)

▶ $\sim$ 1000 C LoC

Preliminary evaluation

Application	Type	Opc	Path opc	Dangerous opc
Clean CMS 1.5	SQLI	221	104	56 (53.85%)
Google CMS 1.8.2	SQLI	62	58	17 (29.31%)
MyForum 1.3	SQLI	1102	651	141 (21.66%)
Pizzis CMS 1.5.1	SQLI	91	38	11 (28.95%)
W2B phpGreetCards	XSS	1078	814	221 (27.15%)
WordPress	XSS	612	26	10 (38.46%)

Experimental results

- open-source applications with known vulnerabilities
- high performance gain
- future improvements can further reduce run-time overhead

Conclusions

Contributions

- hybrid program analysis framework to detect input-driven security vulnerability in web application
- prototype implementation for PHP (at *bytecode* level)

Limitations

- 93/150 Zend opcodes
- limited support for aliasing and class constructs
- second-order injections

Future Work

- improve static analysis module (e.g., static taint analysis)
- support more Zend opcodes

Thank you for the attention!

Questions?