Using Static Program Analysis to Aid Intrusion Detection

M. Egele M. Szydlowski
E. Kirda C. Kruegel

Secure Systems Lab
Vienna University of Technology

SIG SIDAR Conference on
Detection of Intrusions and
Malware & Vulnerability Assessment, 2006

outline

what we want to do and why

- PHP is arguably the most prominent language to develop web-based applications
- Many exploits for vulnerabilities in PHP-applications exist
- User/attacker can influence the application mainly via its parameters
- We employ interprocedural dataflow analysis to gain knowledge about used parameters
- Especially types and possible values of parameters are interesting

what we want to do and why

- PHP is arguably the most prominent language to develop web-based applications
- Many exploits for vulnerabilities in PHP-applications exist
- User/attacker can influence the application mainly via its parameters
- We employ interprocedural dataflow analysis to gain knowledge about used parameters
- Especially types and possible values of parameters are interesting

what we want to do and why

- PHP is arguably the most prominent language to develop web-based applications
- Many exploits for vulnerabilities in PHP-applications exist
- User/attacker can influence the application mainly via its parameters
- We employ interprocedural dataflow analysis to gain knowledge about used parameters
- Especially types and possible values of parameters are interesting

what we want to do and why

- PHP is arguably the most prominent language to develop web-based applications
- Many exploits for vulnerabilities in PHP-applications exist
- User/attacker can influence the application mainly via its parameters
- We employ interprocedural dataflow analysis to gain knowledge about used parameters
- Especially types and possible values of parameters are interesting

what we want to do and why

- PHP is arguably the most prominent language to develop web-based applications
- Many exploits for vulnerabilities in PHP-applications exist
- User/attacker can influence the application mainly via its parameters
- We employ interprocedural dataflow analysis to gain knowledge about used parameters
- Especially types and possible values of parameters are interesting

existing intrusion detection system

MAID tries to characterize web-requests through different models:

- Learning based intrusion detection system developed at the Secure Systems Lab TU Vienna.
- models include parameter presence/absence model, structural inference, token finder models . . .
- No *a priori* knowledge of applications it protects learning is basically done via logfile analysis
- In some cases unnecessary imprecision leads to many false positives → this is what we want to improve

existing intrusion detection system

MAID tries to characterize web-requests through different models:

- Learning based intrusion detection system developed at the Secure Systems Lab TU Vienna.
- models include parameter presence/absence model, structural inference, token finder models . . .
- No *a priori* knowledge of applications it protects learning is basically done via logfile analysis
- In some cases unnecessary imprecision leads to many false positives → this is what we want to improve

existing intrusion detection system

MAID tries to characterize web-requests through different models:

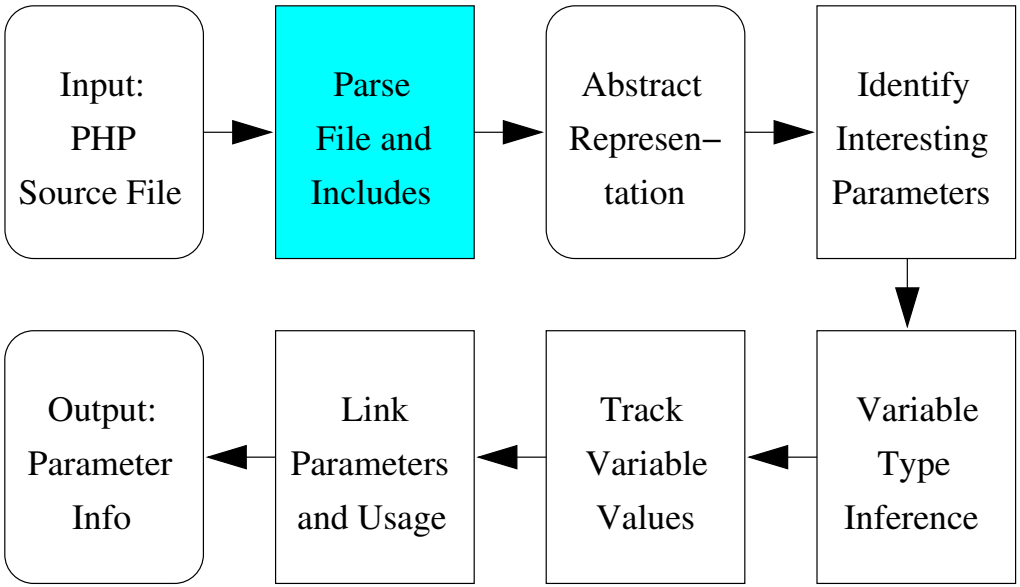
- Learning based intrusion detection system developed at the Secure Systems Lab TU Vienna.
- models include parameter presence/absence model, structural inference, token finder models . . .
- No *a priori* knowledge of applications it protects learning is basically done via logfile analysis
- In some cases unnecessary imprecision leads to many false positives → this is what we want to improve

existing intrusion detection system

MAID tries to characterize web-requests through different models:

- Learning based intrusion detection system developed at the Secure Systems Lab TU Vienna.
- models include parameter presence/absence model, structural inference, token finder models . . .
- No *a priori* knowledge of applications it protects learning is basically done via logfile analysis
- In some cases unnecessary imprecision leads to many false positives $\rightarrow$ this is what we want to improve

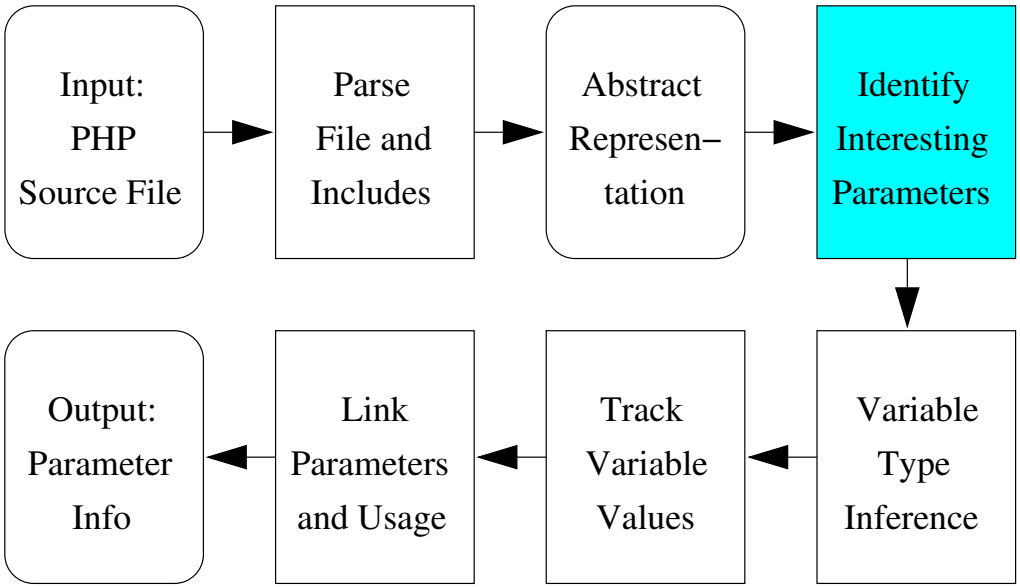
mode of operation



Action performed

parse the application source code

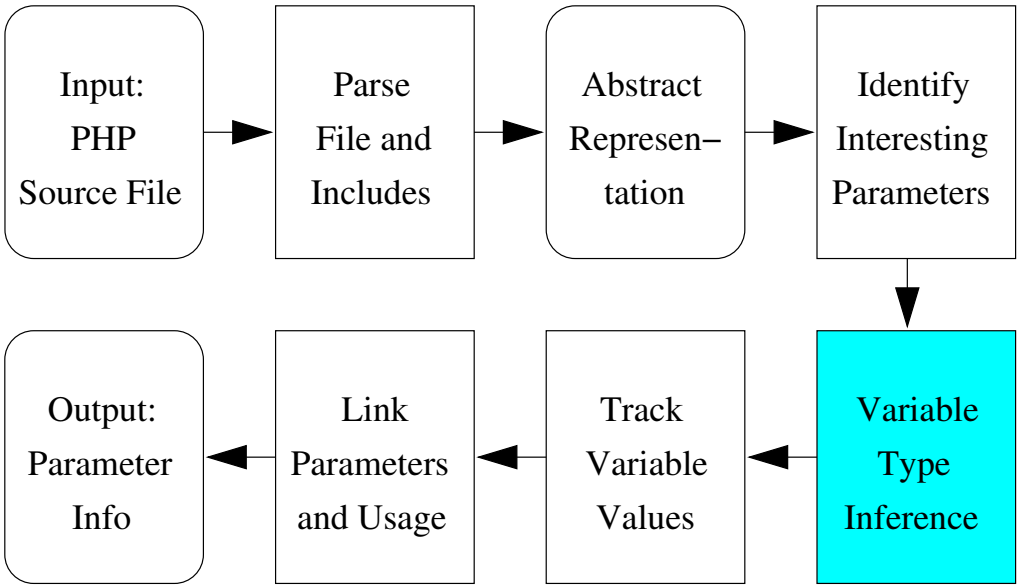
mode of operation



Action performed

identify the parameters the application accepts

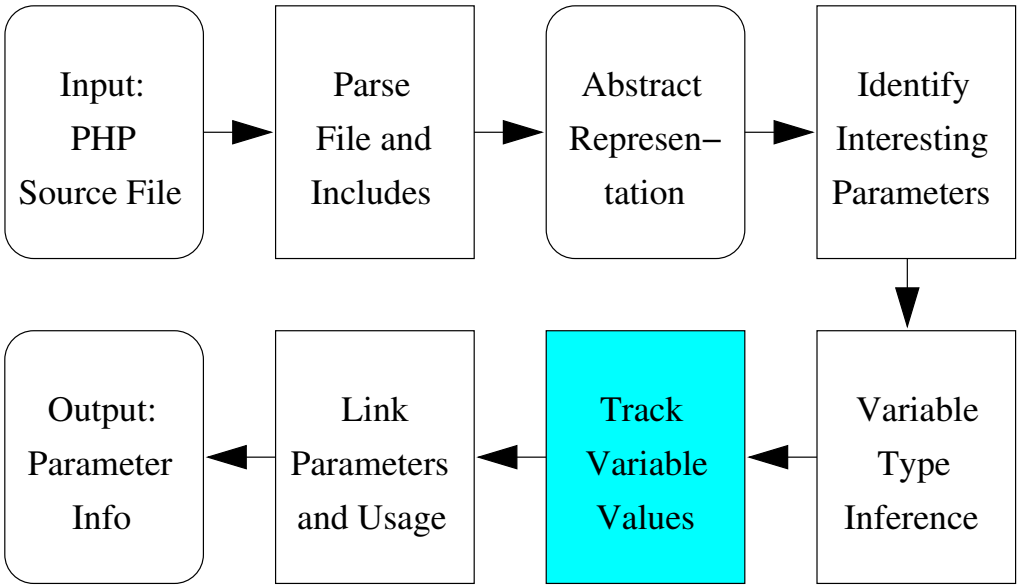
mode of operation



Action performed

type inference on variables

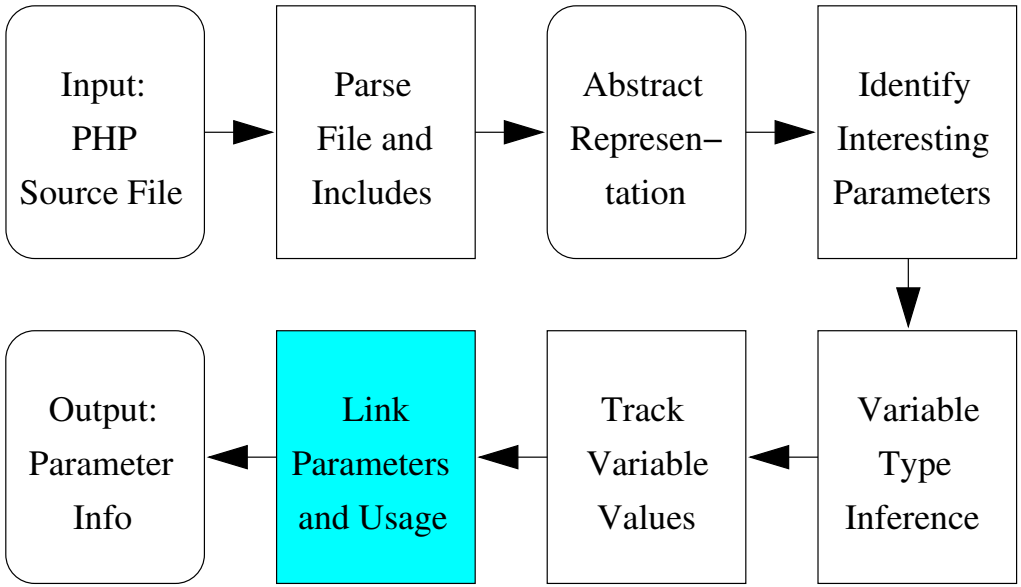
mode of operation



Action performed

variables values are tracked

mode of operation

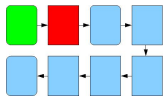


Action performed

parameter usage is examined

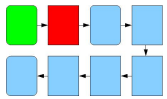
parsing

- Original Zend language parser was used
- Includes are resolved (constant expressions)
- Variables and functions are identified



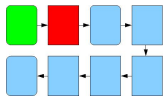
parsing

- Original Zend language parser was used
- Includes are resolved (constant expressions)
- Variables and functions are identified



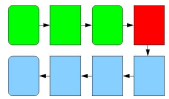
parsing

- Original Zend language parser was used
- Includes are resolved (constant expressions)
- Variables and functions are identified



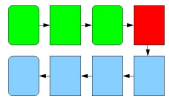
how parameters can be accessed in PHP

- Parameter name is index into a parameter array e.g., `$_GET`, `$_POST` superglobals
- Via `register_globals` risky



how parameters can be accessed in PHP

- Parameter name is index into a parameter array e.g., `$_GET`, `$_POST` superglobals
- Via `register_globals` risky



example on accessing parameters

```
 1  class Util {
 2    function getGet($var, $default = null) {
 3      return (isset($_GET[$var]))
 4        ? Util::dispelMagicQuotes($_GET[$var])
 5        : $default;
 6    }
 7    function getFormData($arg, $default = null) {
 8      return (($val = Util::getPost($arg)) !== null)
 9        ? $val
10        : Util::getGet($arg, $default);
11    }
12  }
13
14  $actionID = Util::getFormData('actionid')
```

example on accessing parameters

```
 1  class Util {
 2    function getGet($var, $default = null) {
 3      return (isset($_GET[$var]))
 4        ? Util::dispelMagicQuotes($_GET[$var])
 5        : $default;
 6    }
 7    function getFormData($arg, $default = null) {
 8      return (($val = Util::getPost($arg)) !== null)
 9        ? $val
10        : Util::getGet($arg, $default);
11    }
12  }
13
14  $actionID = Util::getFormData('actionid')
```

example on accessing parameters

```
 1  class Util {
 2    function getGet($var, $default = null) {
 3      return (isset($_GET[$var]))
 4        ? Util::dispelMagicQuotes($_GET[$var])
 5        : $default;
 6    }
 7    function getFormData($arg, $default = null) {
 8      return (($val = Util::getPost($arg)) !== null)
 9        ? $val
10        : Util::getGet($arg, $default);
11    }
12  }
13
14  $actionID = Util::getFormData('actionid')
```

example on accessing parameters

```
 1  class Util {
 2    function getGet($var, $default = null) {
 3      return (isset($_GET[$var]))
 4        ? Util::dispelMagicQuotes($_GET[$var])
 5        : $default;
 6    }
 7    function getFormData($arg, $default = null) {
 8      return (($val = Util::getPost($arg)) !== null)
 9        ? $val
10        : Util::getGet($arg, $default);
11    }
12  }
13
14  $actionID = Util::getFormData('actionid')
```

example on accessing parameters

```
 1  class Util {
 2    function getGet($var, $default = null) {
 3      return (isset($_GET[$var]))
 4        ? Util::dispelMagicQuotes($_GET[$var])
 5        : $default;
 6    }
 7    function getFormData($arg, $default = null) {
 8      return (($val = Util::getPost($arg)) !== null)
 9        ? $val
10        : Util::getGet($arg, $default);
11    }
12  }
13
14  $actionID = Util::getFormData('actionid')
```

example on accessing parameters

```
 1  class Util {
 2    function getGet($var, $default = null) {
 3      return (isset($_GET[$var]))
 4        ? Util::dispelMagicQuotes($_GET[$var])
 5        : $default;
 6    }
 7    function getFormData($arg, $default = null) {
 8      return (($val = Util::getPost($arg)) !== null)
 9        ? $val
10        : Util::getGet($arg, $default);
11    }
12  }
13
14  $actionID = Util::getFormData('actionid')
```

example on accessing parameters

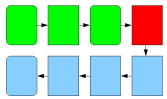
```
 1  class Util {
 2    function getGet($var, $default = null) {
 3      return (isset($_GET[$var]))
 4        ? Util::dispelMagicQuotes($_GET[$var])
 5        : $default;
 6    }
 7    function getFormData($arg, $default = null) {
 8      return (($val = Util::getPost($arg)) !== null)
 9        ? $val
10        : Util::getGet($arg, $default);
11    }
12  }
13
14  $actionID = Util::getFormData('actionid')
```

example on accessing parameters

```
 1  class Util {
 2    function getGet($var, $default = null) {
 3      return (isset($_GET[$var]))
 4        ? Util::dispelMagicQuotes($_GET[$var])
 5        : $default;
 6    }
 7    function getFormData($arg, $default = null) {
 8      return (($val = Util::getPost($arg)) !== null)
 9        ? $val
10        : Util::getGet($arg, $default);
11    }
12  }
13
14  $actionID = Util::getFormData('actionid')
```

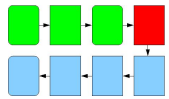
how we find the parameter names

1. Dataflow analysis at procedural level
2. Interprocedural analysis for function calls (recursive)
3. This information can be used for the parameter presence/absence model of our IDS



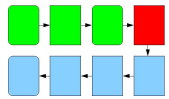
how we find the parameter names

1. Dataflow analysis at procedural level
2. Interprocedural analysis for function calls (recursive)
3. This information can be used for the parameter presence/absence model of our IDS



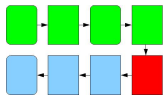
how we find the parameter names

1. Dataflow analysis at procedural level
2. Interprocedural analysis for function calls (recursive)
3. This information can be used for the parameter presence/absence model of our IDS



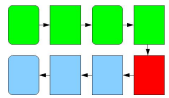
parameter types - valuable information

- Very useful for the IDS (e.g., integers {-}?[0-9]+)
- PHP only has dynamic types (e.g., no `int $x`) and every value is dynamically typecast to whatever type expected by a given operation
- Type inference through applied operations via type matrix, special cases include:
  - `&&, ||, xor, !` always return `boolean`
  - `.` always returns `string`
  - `&, |, ^` `string` iff operands are `string` - `integer` otherwise



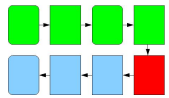
parameter types - valuable information

- Very useful for the IDS (e.g., integers {-}?[0-9]+)
- PHP only has dynamic types (e.g., no `int` $x) and every value is dynamically typecast to whatever type expected by a given operation
- Type inference through applied operations via type matrix, special cases include:
  - `&&, ||, xor, !` always return `boolean`
  - `.` always returns `string`
  - `&, |, ^` `string` iff operands are `string` - `integer` otherwise



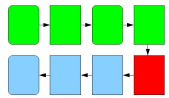
parameter types - valuable information

- Very useful for the IDS (e.g., integers {-}?[0-9]+)
- PHP only has dynamic types (e.g., no `int` $x) and every value is dynamically typecast to whatever type expected by a given operation
- Type inference through applied operations via type matrix, special cases include:
  - `&&,||,xor,!` always return `boolean`
  - `.` always returns `string`
  - `&,|,^` `string` iff operands are `string` - `integer` otherwise



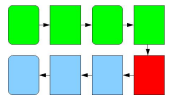
parameter types - valuable information

- Very useful for the IDS (e.g., integers {-}?[0-9]+)
- PHP only has dynamic types (e.g., no `int` $x) and every value is dynamically typecast to whatever type expected by a given operation
- Type inference through applied operations via type matrix, special cases include:
  - `&&,||,xor,!` always return `boolean`
  - `.` always returns `string`
  - `&,|,^` `string` iff operands are `string` - `integer` otherwise



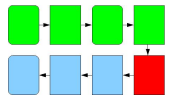
parameter types - valuable information

- Very useful for the IDS (e.g., integers {-}?[0-9]+)
- PHP only has dynamic types (e.g., no `int` $x) and every value is dynamically typecast to whatever type expected by a given operation
- Type inference through applied operations via type matrix, special cases include:
  - `&&,||,xor,!` always return `boolean`
  - `.` always returns `string`
  - `&,|,^` `string` iff operands are `string` - `integer` otherwise



parameter types - valuable information

- Very useful for the IDS (e.g., integers {-}?[0-9]+)
- PHP only has dynamic types (e.g., no `int` $x) and every value is dynamically typecast to whatever type expected by a given operation
- Type inference through applied operations via type matrix, special cases include:
  - `&&`, `||`, `xor`, `!` always return `boolean`
  - `.` always returns `string`
  - `&`, `|`, `^` `string` iff operands are `string` - `integer` otherwise

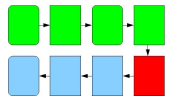


parameter types

1. Try to infer as many variables as possible via the operator-type matrix
2. If a parameter is linked to a variable, assume the parameter has the same type as the variable.

```
1 $x = $_GET['THE_X'];
2 if ($x == 42)
3   echo "x is 42";
4 else
5   echo "error x is not 42"
```

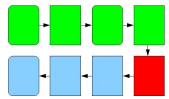
3. Use the more general type if there is a conflict



parameter types

1. Try to infer as many variables as possible via the operator-type matrix
2. If a parameter is linked to a variable, assume the parameter has the same type as the variable.

```
1 $x = $_GET['THE_X'];
2 if ($x == 42)
3   echo "x is 42";
4 else
5   echo "error x is not 42"
```



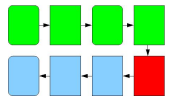
3. Use the more general type if there is a conflict

parameter types

1. Try to infer as many variables as possible via the operator-type matrix
2. If a parameter is linked to a variable, assume the parameter has the same type as the variable.

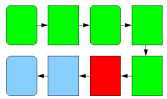
```
1 $x = $_GET['THE_X'];
2 if ($x == 42)
3   echo "x is 42";
4 else
5   echo "error x is not 42"
```

3. Use the more general type if there is a conflict



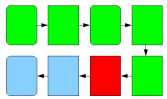
possible value sets

1. Direct comparison via literal
2. Indirect comparison via `switch-case` construct
3. sanitation code (e.g., regexp, built-in functions) – annotations possible
4. This information can be used for the structural models and character distribution models of our IDS



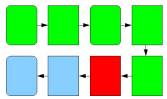
possible value sets

1. Direct comparison via literal
2. Indirect comparison via `switch-case` construct
3. sanitation code (e.g., regexp, built-in functions) – annotations possible
4. This information can be used for the structural models and character distribution models of our IDS



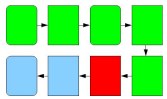
possible value sets

1. Direct comparison via literal
2. Indirect comparison via `switch-case` construct
3. sanitation code (e.g., regexp, built-in functions) – annotations possible
4. This information can be used for the structural models and character distribution models of our IDS



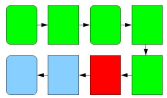
possible value sets

1. Direct comparison via literal
2. Indirect comparison via `switch-case` construct
3. sanitation code (e.g., regexp, built-in functions) – annotations possible
4. This information can be used for the structural models and character distribution models of our IDS



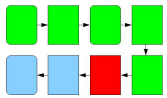
the annotation used for SquirrelMail

- SquirrelMail uses the `sqgetGlobalVar` function to retrieve its parameters.
  first argument: name of the parameter
  second argument: reference to the variable that should receive the value
- annotation: `sqgetGlobalVar:1:2`
  code: `sqgetGlobalVar('THE_X', &$x)`
  now the analyzer knows that `$x` holds the value of the parameter `THE_X`



the annotation used for SquirrelMail

- SquirrelMail uses the `sqgetGlobalVar` function to retrieve its parameters.
  first argument: name of the parameter
  second argument: reference to the variable that should receive the value
- annotation: `sqgetGlobalVar:1:2`
  code: `sqgetGlobalVar('THE_X', &$x)`
  now the analyzer knows that `$x` holds the value of the parameter `THE_X`



```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

```
 1  $otherparam = Util::getFormData("otherparam");
 2  $param = array(
 3    "name"  => "pizzaman",
 4    "value" => Util::getFormData("param"),
 5    "info"  => "something boring");
 6  $thirdparam = do_something($_POST["thirdparam"]);
 7
 8  $strippedparam = stripslashes($param["value"]);
 9  if ($strippedparam == "something")
10  ...
11  switch ($otherparam) {
12    case "something else":
13    ...
14  }
15  preg_match("/^([0-9]4).*", $thirdparam, $number);
```

how we evaluated our results

Evaluation is divided into two parts:

- Standalone Analysis of five real world web-applications
- Crosscheck of some of these results with actual log data
  only GET requests, since POST requests not logged
  We were able to find all parameters that actually appeared in the logs

how we evaluated our results

Evaluation is divided into two parts:

- Standalone Analysis of five real world web-applications
- Crosscheck of some of these results with actual log data
  only GET requests, since POST requests not logged
  We were able to find all parameters that actually appeared in the logs

web applications under examination

We analyzed the following popular web applications

Application	Parameters	Details	Percentage
Horde2/IMP3.1	153	47	31%
Squirrelmail 1.4.6-rc1	268	91	34%
phpBB 2.0.17	316	82	26%
Horde3/IMP4.0.2	298	64	21%
PHP iCalendar 2.1	23	15	65%

info on parameters found

Examples of our findings

- Details for parameters $\approx$ 35% $\rightarrow$ information for IDS
- Horde2: `actionID TYPE_INT` values: 0,1,101,102, ...
  Horde3: `actionID TYPE_STRING` values:
  'add_address', 'add_attachment', ...
- iCalendar: `getdate TYPE_STRING`
  `preg_match("/([0-9]4)([0-9]2)([0-9]2)/")`

info on parameters found

Examples of our findings

- Details for parameters $\approx$ 35% $\rightarrow$ information for IDS
- Horde2: `actionID TYPE_INT` values: 0,1,101,102, ...
  Horde3: `actionID TYPE_STRING` values:
  'add_address', 'add_attachment', ...
- iCalendar: `getdate TYPE_STRING preg_match("/([0-9]4)([0-9]2)([0-9]2)/")`

info on parameters found

Examples of our findings

- Details for parameters $\approx 35\% \rightarrow$ information for IDS
- Horde2: `actionID TYPE_INT` values: 0,1,101,102, ...
  Horde3: `actionID TYPE_STRING` values:
  'add_address', 'add_attachment', ...
- iCalendar: `getdate TYPE_STRING`
  `preg_match("/([0-9]4)([0-9]2)([0-9]2)/")`

crosscheck with real log data

Results for Horde2/IMP 3.1

- Timeframe: three months, 30.000 accesses
- name: `reason` type: `TYPE_STRING` values: `'failed'`, `'logout'`, `'session'` exact matches
- `to`, `cc`, `bcc` only information `TYPE_STRING`
- `f` filename to download-dialog (relict) replaced by MIME-Header parsing

crosscheck with real log data

Results for Horde2/IMP 3.1

- Timeframe: three months, 30.000 accesses
- name: `reason` type: `TYPE_STRING` values: `'failed', 'logout', 'session'` exact matches
- `to, cc, bcc` only information `TYPE_STRING`
- `f` filename to download-dialog (relict) replaced by MIME-Header parsing

crosscheck with real log data

Results for Horde2/IMP 3.1

- Timeframe: three months, 30.000 accesses
- name: `reason` type: `TYPE_STRING` values: `'failed', 'logout', 'session'` exact matches
- `to, cc, bcc` only information `TYPE_STRING`
- `f` filename to download-dialog (relict) replaced by MIME-Header parsing

crosscheck with real log data

Results for Horde2/IMP 3.1

- Timeframe: three months, 30.000 accesses
- name: `reason` type: `TYPE_STRING` values: `'failed', 'logout', 'session'` exact matches
- `to, cc, bcc` only information `TYPE_STRING`
- `f` filename to download-dialog (relict) replaced by MIME-Header parsing

crosscheck with real log data

Results for Squirrelmail

- Timeframe: three weeks, 13.000 accesses
- name: `smaction` type: `TYPE_STRING` values: `'draft'`, `'edit_as_new'`, `'forward'`, `'forward_as_attachment'`, `'reply'`, `'reply_all'` exact matches
- `what`, `where` type: `TYPE_STRING` dynamic search parameters

crosscheck with real log data

Results for Squirrelmail

- Timeframe: three weeks, 13.000 accesses
- name: `smaction` type: `TYPE_STRING` values: `'draft'`, `'edit_as_new'`, `'forward'`, `'forward_as_attachment'`, `'reply'`, `'reply_all'` exact matches
- `what`, `where` type: `TYPE_STRING` dynamic search parameters

crosscheck with real log data

Results for Squirrelmail

- Timeframe: three weeks, 13.000 accesses
- name: `smaction` type: `TYPE_STRING` values: `'draft'`, `'edit_as_new'`, `'forward'`, `'forward_as_attachment'`, `'reply'`, `'reply_all'` exact matches
- `what, where` type: `TYPE_STRING` dynamic search parameters

crosscheck with real log data

- Evaluation shows that we were able to find all parameters in the sourcecode that are actually used
- For about 35%, detailed information on these parameters could be deduced by our analyzer
- This additional information can be used to improve the precision of our IDS

crosscheck with real log data

- Evaluation shows that we were able to find all parameters in the sourcecode that are actually used
- For about 35%, detailed information on these parameters could be deduced by our analyzer
- This additional information can be used to improve the precision of our IDS

crosscheck with real log data

- Evaluation shows that we were able to find all parameters in the sourcecode that are actually used
- For about 35%, detailed information on these parameters could be deduced by our analyzer
- This additional information can be used to improve the precision of our IDS

details on phpBB

Scenario:

- December 2005 mass defacement of `phpBB 2.0.17`
- Exploit modifies the `GLOBALS` array via: `profile.php?GLOBALS[...]`
- Parameter presence and absence model → `GLOBALS` not a parameter of the web-application

details on phpBB

Scenario:

- December 2005 mass defacement of `phpBB 2.0.17`
- Exploit modifies the `GLOBALS` array via: `profile.php?GLOBALS[...]`
- Parameter presence and absence model → `GLOBALS` not a parameter of the web-application

details on phpBB

Scenario:

- December 2005 mass defacement of `phpBB 2.0.17`
- Exploit modifies the `GLOBALS` array via: `profile.php?GLOBALS[...]`
- Parameter presence and absence model $\rightarrow$ `GLOBALS` not a parameter of the web-application

Summary

- Parameter name detection via inter-procedural dataflow analysis
- Determine types and possible values of parameters based on their use by the application
- Would have been able to detect real-world exploit