Detecting Incorrect Wordpress Plugin Function Usage

Jens Thomas Vejlby Nielsen
jtvn10@student.aau.dk
Computer Science 10^{th} Semester Student Project
Aalborg University

Aalborg Universitet
Cassiopeia
Computer Science
Selma Lagerlöfs Vej 300
Phone 96 35 97 31
Fax 98 13 63 93
http://cs.aau.dk

Title: Detecting Incorrect Wordpress Plugin Function Usage

Project Period:
10^{th} semester

Project Group:
des1010f15

Written by:

Jens Thomas Vejlby Nielsen

Supervisor: René Rydhof Hansen

Number of prints: 2

Number of pages: 28

Total pages: 45

Appendices: 5

Completed on: 03.06.2015

Abstract:

This thesis presents the problem of incorrectly using either PHP build-in or homemade functions for WordPress plugin development. WordPress itself is created in a secure way, and vulnerabilities are quickly corrected. This is not the case for plugins, where there can be a multitude of vulnerabilities. WordPress supplies functions for correctly sanitisation of data, along with connecting to databases. WordPress allows the core functionality to be changed by using filters and actions, and if a developer forgets to close a filter this can have security and correctness implications.
A proof-of-concept solution using the nuXmv Model Checker on a WordPress plugin model for finding incorrect function usage and open filters is presented. Tests of the tool show that it is still clear that this is a proof-of-concept solution.

Summary

This thesis presents the problem of incorrectly using either PHP build-in or homemade functions for WordPress plugin development, along with using filters and actions incorrectly. WordPress itself is created in a secure way, and vulnerabilities are quickly corrected. This is not the case for plugins, where there can be a multitude of vulnerabilities. WordPress supplies functions for correctly sanitisation of data, along with connecting to databases,and guidelines for how data should be processed. WordPress allows the core functionality to be changed by using filters and actions, and if a developer forgets to close a filter this can have security and correctness implications.

Model Checking is where a model of a system is created, and a number of specification formulas is then used to verify the model. In this report Linear Temporal Logic and Computation Tree Logic is used. One of the prominent differences between the two logics is that CTL can reason about execution paths, whereas LTL cannot, as LTL works on the whole model.

A proof-of-concept solution using the nuXmv Model Checker, which implements CTL and LTL, on a WordPress plugin model for finding incorrect function usage and open filters is presented. The tool automatically generates specifications for checking if any incorrect function is used, and if there exists an open filter.

Tests of the tool show that it is still clear that this is a proof-of-concept solution, yet it still manages to detect open filters.

Preface

This project is the result of a 10^{th} semester Master thesis project conducted by a student from the Department of Computer Science, Aalborg University. The theme of this report is detecting incorrect function usage in WordPress plugins.

- Citations will be on the form [x], where the number represents the place in the bibliography in the back of the report.
- Code written in the report will be in the `ttfamily` font family.
- Broken lines are represented by a ↵.
- ... marks omitted code.

I would like to thank Kenneth Jepsen, Morten Nørtoft and Mikkel Vej for the code with which this project is based. Additionally, I would like to thank René Rydhof Hansen for supervision and guidance during the project.

Contents

1 Introduction

In recent years, a number of prominent web applications has been subjected to attacks by malicious attackers, namely Sony Entertainment [8], the Washington Post [12], and the danish company CSC [11]. This is possible as all these applications are connected to the Internet, and as such is reachable from anywhere in the world. This shows that web application security is very important, and since the applications are connected to the Internet, a malicious attacker can attack at his/her leisure. Failure to properly secure a web application can be extremely costly, as it was the case in the examples. Another example is if a malicious attacker manages to gain entry into an online bank, or gains unauthorised access to confidential information, such as Social Security numbers, as was the case in the CSC hack. This shows that it is important to ensure that only authorised access is allowed.

There exist a multitude of different languages for creating web applications, but the focus in this project will be on the PHP programming language, since it is widely used for web development [9, 6, 10]. Often the hosting providers supplies a stack containing the web server Apache, the language interpreter PHP and the database system MySQL. Additionally, there exist a number of frameworks for creating web applications in PHP, but in this project the focus will be on the Content Management System WordPress, as it is widely distributed, and estimated to power around 23 % of the top million websites, along with being used as Content Management System (CMS) on 60% of websites which uses CMS's [19].

As such, it is important that WordPress is created securely, as it can be quite costly if there is an error in the application. For example, a web shop vulnerable to a SQL injection could lose funds if a malicious attacker could place false orders, extract all customers, or change the prices.

WordPress is comprised of a core which in itself is created in a secure way, with a lot of focus on security [19]. WordPress can be extended by using custom plugins and themes, and these are often not as secure as WordPress itself. Not only that, but often plugin developers does not use the supplied WordPress functions, for example for database access, but instead either uses custom functions, or the PHP supplied ones. This can have security implications, and can cause incorrect usage, for example if a developer wishes to allow some user-supplied HTML, but uses an incorrect sanitisation function, then a Cross-Site-Scripting attack might be possible.

To detect the incorrect function usage, Model Checking can be used. The goal of a Model Checker is to verify whether a given specification is satisfies by a model. In this case, the Model is the WordPress plugin, and the specification is written in Linear Temporal Logic (LTL) and/or Computation Tree Logic (CTL). These logics can be used to reason about paths in a program, for example by checking "is there a path to a `htmlspecialchars`, and for checking if there is a path from an opening of a filter to a closing of a filter. For WordPress, the wrong function usage could be to check if the escape methods `htmlspecialchars`,`htmlentities` `etc.` is reachable. In WordPress, there is filters and actions, which can be used to modify an existing function within WordPress, for example the `upload_dir` filter, which specifies where a file is uploaded. Additionally, WordPress supplies functions for uploading and checking files, using the filter `upload_mimes` for the allowed filters. This can lead to a vulnerability, for example if a plugin enables the uploading of PHP files, and forgets to close the filter after usage, in which case every upload using the WordPress upload functions will allow PHP files.

WordPress will be further described in Chapter 2, and Model Checking will be described in Chapter 3, and a proof-of-concept solution for finding incorrect functions and open filters will be presented in Chapter 4. Finally, Chapter 5 concludes the project.

2 WordPress

WordPress is a popular, free, open-source Content-Management System (CMS), used by 23 % of the top 10 million websites, with an estimated market share of 60 % of all websites using a CMS [19].

WordPress is run by a Core Leadership Team, consisting of five lead developers and a dozen core developers, along with a community of developers, whom contribute to the development of WordPress. However, only the lead developers and a selected number of community developers has commit access.

To keep WordPress secure, there is a WordPress Security Team, consisting of 25 experts, lead developers and security researchers, which manage the WordPress security. To promote the detection of vulnerabilities in WordPress, it is possible to contact the WordPress security team, and any authors of vulnerabilities will be given proper credit. If the vulnerability is severe, the fix can be distributed immediately, or in an upcoming release.

To increase the security, WordPress has, in version 3.7, introduced automatic updating of minor releases, such as 3.7.1 to 3.7.2. In this way, updates to the WordPress core can be automatically distributed.

As the above describes, WordPress takes security very serious, and is quick to patch any errors found in WordPress itself. However, this is not the case for Plugins and Themes, where the response time can be longer. Not only that, but often plugins are not created with security in mind, and as such there can be quite a few vulnerabilities in the same plugin, and the system is only as safe as the weakest link.

To assist the plugin developers in increasing the security of the plugins, the WordPress core supplies sanitisation and escape functions for the OWASP

Top 10 [4], along with the best practices for how it is done in WordPress [19, 15]. In the case where the WordPress Security Team fails to reach a plugin developer, they might either patch the plugin themselves, or remove it from the repository.

2.1 Securing WordPress

WordPress supplies API's to protect, validate and sanitise data, along with protection from the most popular attacks. In this section, it is described how to prevent against the most popular attacks using WordPress. The vulnerabilities is further described in my previous report [18].

2.1.1 Preventing Cross Site Scripting in WordPress

Cross Site Scripting is an attack in which a malicious piece of code is executed on the client site. This can, for example, be used to steal the authentication cookies for WordPress, or transfer data about the client(s).

For prevention, the PHP functions `htmlspecialchars` and `htmlentities` can be used, but WordPress recommends using the WordPress specific escape functions;

1. `wp_kses` is used to remove all untrusted HTML, and is used for data validation before the data is added to the database.

2. `esc_(html/textarea/attr)` and `sanitize_text_field` is used to escape when extracting the data from the database, and recommended right before printing.

The benefit of using the WordPress specific functions as opposed to the functions supplied by PHP, is that WordPress allows some HTML tags to be used. As such, the client is not completely restricted from using HTML, whereas this can cause security problems when the PHP functions is used, if a mistake is made and the escaping is not done properly. Additionally, the WordPress functions allows the direct printing and localisation of the results, so that if a plugin author has added support for additional languages, then the translation will be done seamlessly.

2.1.2 SQL Injections

An SQL injection is an attack in which the database query has been altered by a malicious attacker to serve the attackers purpose. This could for example

be used to bypass login authentication, delete from the database, or extract information.

For PHP, preventing SQL injections can be done either by escaping the input, for example by using `mysqli_real_escape_string`. Another approach is to use prepared statements. For WordPress, the `esc_sql` function can be used to escape the input, but WordPress encourages the usage of the `$wpdb` object for any and all database access. The `$wpdb` object contains a number of functions which will automatically escape all input, and a number of functions in which the user will have to escape the input.

Functions that automatically escape input

The functions supplied by the `$wpdb` object is the recommended way for interacting with the WordPress database in a secure way. The functions which automatically sanitises input are;

1. `$wpdb->insert( $table, (array)$data, $format )`
2. `$wpdb->update( $table, (array)$data, (array)$where )`
3. `$wpdb->replace( $table, $data, $format )`
4. `$wpdb->delete( $table, $where )`

Where for example the `$wpdb->insert` can be used as seen in Listing 2.1. In this example the post variables are input directly, as WordPress will ensure that no SQL Injection happen.

In addition to the functions supplied by the `$wpdb` object, there exist a number of functions for example for creating posts, getting/setting/deleting users, options, posts, metadata and so on. As such, in most cases it is not necessary to use the functions which does not automatically sanitise.

```
1 $wpdb->insert($wpdb->postmeta,
2 array( 'post_id' => $_POST['id'],
3       'meta_key' => $_POST['key'])
4 );
```

Listing 2.1: Automatic Escaping

Functions that the user needs to escape

In addition to the functions that WordPress automatically escapes, there is a number of functions in which it is the developers responsibility to properly

escape input, e.g. using `esc_sql`, `$wpdb->prepare` and/or `esc_like`. The first function is used to escape a query, whereas the second one creates a prepared statement from the `wpdb` object. The last function is used to escape like clauses in queries. Some of the functions that the programmer has to escape is;

1. `$wpdb->get_var( 'query', column_offset, row_offset )`, used to extract a single variable.
2. `$wpdb->get_row('query', output_type, row_offset )`, used to extract a row.
3. `$wpdb->get_col( 'query', column_offset )`, used to extract the columns.
4. `$wpdb->get_results( 'query', output_type )`, used for queries which return a result.
5. `$wpdb->query('query')`, used for queries.

An example of unsafe function usage can be seen in Listing 2.2. In this case, the ID variable can be set to what the attacker decides, and as such be used for an SQL Injection attack. For securing the query, either the `esc_sql` function can be used, or everything can be wrapped in a prepared statement, as seen in Listing 2.3.

```
1 $wpdb->get_var("SELECT * FROM $wpdb->postmeta WHERE ID=$_POST↩
      ['ID']" );
```

Listing 2.2: Unsafe SQL Query

```
1 $wpdb->get_var($wpdb->prepare("SELECT * FROM $wpdb->postmeta ↩
      WHERE ID=%s", $_POST['ID'] ));
```

Listing 2.3: Prepared Statement SQL Query

2.2 Actions and filters

Actions in WordPress allows one to hook into existing actions in WordPress, for example the action which prints the title. This is accomplished using the `add_action` function. Another example could be to hook into the published posts and send an e-mail whenever a post is published, as seen in Listing 2.4.

The action is performed whenever the function with which it is registered is executed, so in this case, it is executed whenever a user published a blog post. In addition to adding actions, it is also possible to remove actions, using the `remove_action` function. It is possible to create custom actions, and these are executed using the `do_action` function.

```
1  function email_admin( $post_ID ) {
2    $admin = 'admin@example.org';
3    wp_mail( $admin, "Blog updated", 'Please verify the new blog ↩
         content: http://blog.example.com' );
4    return $post_ID;
5  }
6  add_action( 'publish_post', 'email_admin' );
```

Listing 2.4: Action example

Filters differ from actions in that they are used to modify existing functionality, for example by replacing an existing hook. An example could be to replace the allowed mime filter that WordPress uses when a file is uploaded. Filters are, like actions, added with the `add_filter` function, and removed with the `remove_filter` function. Whenever a filter is added, once the plugin is done with the filter it should remove the filter to avoid any side-effects. Consider for example the plugin in Listing 2.5. The filter in this case changes the allowed mime types to allow uploading of PHP files, which by adding the filter will replace it for every upload operation in any part of the site (that is, if it correctly uses the WordPress functions for uploading media). This is an unfortunate side effect that can have major security implications, as it has just become possible to upload any PHP file. The solution to this is to remember to remove the filter after usage, in which case the default filter will be used, so in the example this can be done by adding `remove_filter('upload_mimes, 'openfilter_upload_mimes')` after line 17.

```
1  <?php
2  /*
3    Plugin Name: Open Filter
4    Description: Opens a filter for uploading PHP files - very ↩
         vulnerable!
5    Version: 1.0
6    Author: Thomas Vejlby Nielsen
7    Author URI: http://jtvn.dk
8  */
9
10 function openfilter_upload_mimes ( $existing_mimes=array() ) {
```

```
11    $existing_mimes['php'] = 'file/php';
12    return $existing_mimes;
13  }
14
15  add_filter('upload_mimes', 'openfilter_upload_mimes');
16
17  //Do stuff that requires uploading PHP files using the WordPress↩
        upload functions..
```

Listing 2.5: Filter example

3 Model Checking

For detecting the incorrect function usage, Model Checking can be used. For constructing the model nuXmv [3] is used, and for verifying the model Computation Tree Logic and Linear Temporal Logic is used. This chapter is heavily based on Huth and Ryan [16].

When verifying models, there are a number of different approaches [16].

- **Proof/Model based** - a proof based approach is a set of formulas Γ, and a specification ϕ, where the goal is to find a proof that $\Gamma \vdash \phi$. This often requires user-supplied assistance. A model based approach models a system M and a specification ϕ that needs to be checked if it satisfies the model, that is, $M \models \phi$. This can be automatic for finite models.
- **The degree of automation** - how automatic the verification approach can be, ranging from fully automatic to fully manual. Often computer-assisted verification tools are somewhere in the middle.
- **Full vs property-verification** - the specification may describe a single property, or the whole system, which can be quite expensive to verify.
- **Domain of application** - whether it is hardware or software; sequential or concurrent, and whether it is reactive, which means that the system is made to not terminate, or terminating, in which case the system eventually terminates.
- **Pre and post-development** - The verification approach has a greater advantage the earlier it is introduced, as errors are more expensive the

later they are caught.

A Model Checker is a model based, automatic, property-verification approach, intended to be uses for reactive concurrent systems in a post-development methodology. Model Checkers are based on temporal logic, which means that the the model consists of several states, and a formula which can be true in some states, and false in others as the model transitions between states. The temporal logics used in this thesis is the Linear Time Temporal Logic, described in section 3.1 and Computation Tree logic, described in section 3.2.

The models M are transition systems, and the properties ϕ are formulas in temporal logic, and so to verify that a system satisfies a property, that is, if it is the case that a model satisfies a specification ($M \models \phi$), then a Model Checker will respond with a yes, and if it is not the case, the model checker will answer with a no, and often supply a counter-example. To conduct the verification in itself it is necessary to:

- Use the Model Checkers descriptive language to create a model M of a system.
- Use the Model Checkers specification language to create the temporal logical formulas ϕ.
- Supply M and ϕ as input to a Model Checker.

The model checker used in this thesis is the nuXmv model checker [3], which extends the well-known NuSMV model checker [2].

3.1 Linear Time Temporal Logic

Linear Time Temporal Logic (LTL) is a logic with connectives for modelling the future, by modelling time as a sequence of states. In LTL, the future is not determined, and as such it is necessary to consider several different paths.

LTL works with atomic formulas, such as *p,q,r...*, where each of these atoms can represent an atomic fact, for example *"Add_filter is called", "The Plugin is active"* etc. LTL, however, cannot reason about the existence of paths, this can be done in CTL (described in section 3.2).

3.1.1 Syntax

The syntax for LTL given in Backus Naur form is as follows [16]

$$\phi := \top \mid p \mid \bot \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (X\phi) \mid (F\phi) \mid (G\phi) \mid (\phi U \phi) \mid (\phi W \phi) \mid (\phi R \phi)$$

where p is any propositional atom.

The symbols $\top$, $\bot$ and ϕ (if $\neg\phi$) are all LTL formulas, as well as atoms. The temporal connectives X, F, G, U, R and W represents the neXt state, some Future state, Globally (all future states), Until, Release and Weak-until.

An example of a LTL formula is $G\ r \vee \neg q\ U\ p$, which has been illustrated in Figure 3.1. This implies a binding where unary connections binds the most tightly ($\neg$, X, F and G), then comes U, R and W. Next is $\vee$ and $\wedge$, and lastly $\rightarrow$ which has the loosest binding.

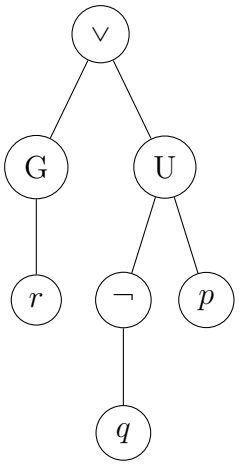

Figure 3.1: Figure of the LTL formula $G\ r \vee \neg q\ U\ p$.

3.1.2 Semantics

Formally, LTL is used to model transition systems with states and transitions, where

Definition 1. A transition system $M = (S, \rightarrow, L)$ is a model consisting of a set of states S, a transition relation *to*, such that for every $s \in S$ there exists some $s' \in S$ with $s \rightarrow S$. Finally there is a labelling function $L : S \rightarrow P(Atoms)$, where $P(Atoms)$ is the power set of atoms (for example the power set of $\{p, q\}$ is $\{\varnothing\}, \{p\}, \{q\}, \{p, q\}$).

The transition system M is modelled as a directed graph contain all propositional atoms that is true in a given state, such as for example in

??. In the figure, there are three states; s_1, s_2 and s_3, while the labels are $L(s_0) = p$, $L(s_1) = r, q$ and $L(s_2) = q, r$. Formally,

Definition 2. In the model $M = (S, \rightarrow, L)$ a path is an infinite sequence of states $s_1, s_2, s_3.... \in S$ such that for each $i \geqslant 1, s_i \rightarrow s_{i+1}$. A path is denoted as π.

With the above definition, the definition for satisfaction ($\models$) is

Definition 3. The model $M = (S, \rightarrow, L)$ and a path $\pi = s_1, s_2, ...$ is satisfied by the satisfaction relation $\models$ as follows [16]:

1. $\pi \models \top$
2. $\pi \not\models \bot$
3. $\pi \models p$ iff $p \in L(s_1)$
4. $\pi \models \neg\phi$ iff $\pi \not\models \phi$
5. $\pi \models \phi_1 \wedge \phi_2$ iff $\pi \models \phi_1$ and $\pi \models \phi_2$
6. $\pi \models \phi_1 \vee \phi_2$ iff $\pi \models \phi_1$ or $\pi \models \phi_2$
7. $\pi \models \phi_1 \rightarrow \phi_2$ iff $\pi \models \phi_2$ when $\pi \models \phi_1$
8. $\pi \models X\phi$ iff $\pi^2 \models \phi$
9. $\pi \models G\ \phi$ iff $\pi^i \models \phi$ for all $i \geqslant 1$
10. $\pi \models F\ \phi$ iff $\pi^i \models \phi$ for some $i \geqslant 1$
11. $\pi \models \phi\ U\ \psi$ iff there exist an $i \geqslant 1$ such that $\pi^i \models \psi$ and for all $j = 1, ..., i-1$ it is the case that $\pi^j \models \phi$
12. $\pi \models \phi\ W\ \psi$ iff either there exist an $i \geqslant 1$ such that $\pi^i \models \psi$ and for all $j = 1, ..., i-1$ it is the case that $\pi^j \models \phi$; or for all $k \geqslant 1$ it is the case that $\pi^k \models \psi$
13. $\pi \models \phi\ R\ \psi$ iff either there exist an $i \geqslant 1$ such that $\pi^i \models \phi$ and for all $j = 1, ..., i-1$ it is the case that $\pi^j \models \psi$; or for all $k \geqslant 1$ it is the case that $\pi^k \models \psi$

The first two clauses reflect that $\top$ is always true and $\bot$ is always false. Clause 3-7 is the clauses for the propositional logic. Clause 8 starts the path at the second state, and clause 9 states that it is reachable from all other paths, whereas clause 10 states that it is reachable from some path. Clause 11 states that ϕ_1 holds until ϕ_2 holds, and ϕ_2 must hold in some future state, whereas this last requirement is not present in clause 12. Clause 13 states that ψ must remain true until ϕ becomes true.

To say that a system as a whole is satisfied, all execution paths satisfies the LTL formula

Definition 4. If $M = (S, \rightarrow, L)$ is a model, $s \in S$ and ϕ is an LTL formula. Then $M, s \models \phi$ if , for every execution path π of M, which starts at s, it is the case that $\pi \models \phi$.

3.2 Computation Tree Logic

Computation Tree Logic (CTL) is a branching-time logic, meaning that CTL reasons about individual paths, as opposed to LTL which only reasons about all paths. This means that CTL can be used to reason about for example in which order functions in a program was called, whereas LTL can only detect that the function is called. As for LTL, the work is with a fixed set of formulas.

Definition 5. CTL is defined in Backus Naur Form as follows:

$$\phi := \top \mid \bot \mid q \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid AX\phi \mid \\ EX\phi \mid AF\phi \mid AG\phi \mid EG\phi \mid A[\phi U \phi] \mid E[\phi U \phi]$$

where p is the atomic formulas, as in LTL.

As opposed to LTL, each of the formulas consists of two symbols; A means "along All paths", whereas E means "there Exist some path". The other part of the pair is, like in LTL, X, F, G, U means neXt state, some Future state, Globally and Until.

3.2.1 Semantics

Much like LTL, the semantics of CTL is;

Definition 6. Let the model $M = (S, \rightarrow, L)$ be a CTL model, $s \in S$ and ϕ a CTL formula. $M, s \models \phi$ is defined as the induction on ϕ [16];

1. $M, s \models \top$ and $M, s \not\models \bot$

2. $M, s \models q$ iff $p \in L(s)$
3. $M, s \models \neg\phi$ iff $M, s \not\models \phi$
4. $M, s \models \phi_1 \wedge \phi_2$ iff $M, s \models \phi_1$ and $M, s \models \phi_2$
5. $M, s \models \phi_1 \vee \phi_2$ iff $M, s \models \phi_1$ or $M, s \models \phi_2$
6. $M, s \models \phi_1 \rightarrow \phi_2$ iff $M, s \not\models \phi_1$ or $M, s \models \phi_2$
7. $M, s \models AX\ \phi$ iff we have that $s \rightarrow s_1$ we have $M, s_1 \models \phi$ for all s_1
8. $M, s \models EX\ \phi$ iff we have that $s \rightarrow s_1$ we have $M, s_1 \models \phi$ for some s_1
9. For $M, s \models AG\ \phi$ we have that $M, s_i \models \phi$ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3...$, where s_1 equals s, and all s_i along the path
10. For $M, s \models EG\ \phi$ we have that $M, s_i \models \phi$ holds iff for some path $s_1 \rightarrow s_2 \rightarrow s_3...$, where s_1 equals s, and all s_i along the path
11. For $M, s \models AF\ \phi$ we have that $M, s_i \models \phi$ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3...$ where s_1 equals s, and for some s_i along the path
12. For $M, s \models AF\ \phi$ we have that $M, s_i \models \phi$ holds iff for some path $s_1 \rightarrow s_2 \rightarrow s_3...$ where s_1 equals s, and for some s_i along the path
13. For $M, s \models A[\phi_1\ U\ \phi_2]$ we have that $M, s_i \models \phi$ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3...$ where s_1 equals s, that path satisfies $\phi_1\ U\ \phi_2$
14. For $M, s \models E[\phi_1\ U\ \phi_2]$ we have that $M, s_i \models \phi$ holds iff for some path $s_1 \rightarrow s_2 \rightarrow s_3...$ where s_1 equals s, that path satisfies $\phi_1\ U\ \phi_2$

The difference between LTL and CTL is that LTL cannot reason about execution paths, whereas CTL can. LTL can, however, express some constructs that cannot be done in CTL, and vice-versa. Neither of the two logics are able to track variables, although there has been an extension for CTL which serves this purpose.

3.3 nuXmv

nuXmv is a continuation of the widely used NuSMV model checker. nuXmv provides a language for describing the system model, along with support for writing CTL and LTL specifications. When nuXmv finds a specification where the model does not adhere to the specification it will print a counter example.

4 Proof-of-concept solution

For detecting the incorrect function usage, a proof-of-concept solution has been constructed. The solution parses a PHP Plugin, generates a nuXmv file, with generated CTL and LTL formulas for detecting unclosed filters, and if the following functions are being used;

- `move_uploaded_file`
- `htmlentities`
- `htmlspecialchars`
- `preg_match`
- `mysqli_real_escape_string`
- `mysql_real_escape_string`

The code is based on the C# implementation created by Jepsen, Nørtoft and Vej[17], who have created a tool called Eir, which is an extendable analysis tool. They implemented Taint Analysis to detect reflected XSS and SQL Injections, while being able to track the data flow through the database. Additionally, they made it possible to scan WordPress plugins, and at present they have a module for managing `get_options` and `set_options` and WordPress hooks.

4.1 Parsing

At first, a plugin is parsed from files to AST representations, which is then used to create a Super-CFG, wherein the function calls are inlined. An example can be seen in Listing 4.1, which is a simple PHP file that will print out ”test”. The constructed CFG does not ensure that the code is actually runnable, such as for example if too few arguments is supplied to a function. The graph representation of the file can be seen in Figure 4.1, where the first state is in the entry state, which will always be `0`. The second state is the assign statement from line 14, followed by state 3 as the variable declaration, finally with the construction of the class with the new keyword. State 5 is the method call in line 15, which will be inlined as line 8-13.

```
1  <?php
2  class ClassOne {
3    public $var1;
4    function __construct($var) {
5      $this->var1 = $var;
6    }
7
8    function printOut($extra) {
9      if ($extra)
10       $var = mysql_real_escape_string($extra);
11       echo $var;
12   }
13 }
14 $tmp = new ClassOne();
15 $tmp->printOut('text');
```

Listing 4.1: Test PHP file

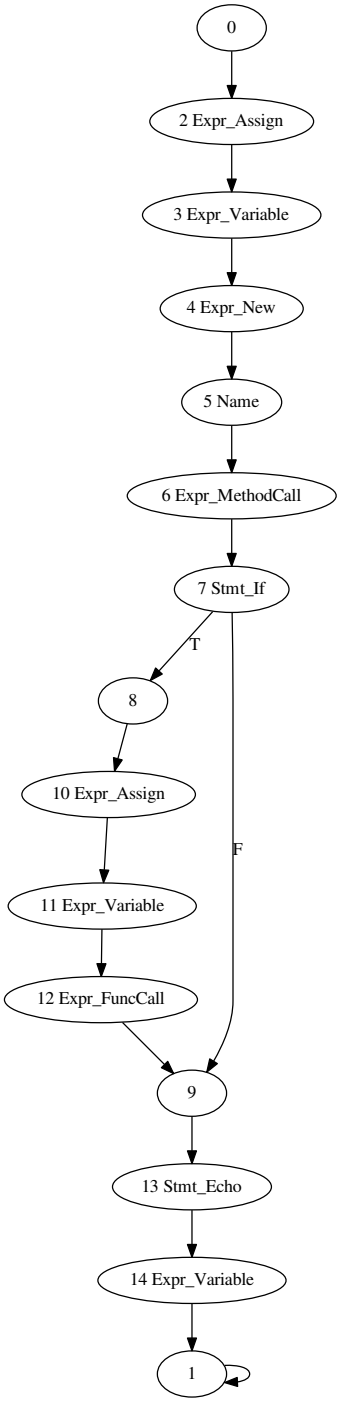

Figure 4.1: Super-CFG representation of the program in Listing 4.1

To handle function calls within different files, the parser will start by parsing all the plugin files, and construct the list of functions and the AST representation of the files. After this step, the parser will construct a Super-CFG from each plugin file, and all function and method calls will be inlined, including basic object oriented calls, as shown in Listing 4.1. In addition, the parser is able to resolve variables, such that if a class function is called, it will be possible to inline the called function. File inclusions are possible by adding the Super-CFG of the file which has been included, and this will be done when the nuXmv model is generated.

The parser is able to parse and inline the following WordPress specific function calls;

- `add_action`
- `add_filter`
- `add_menu_page`
- `add_submenu_page`
- `add_dashboard_page`
- `add_posts_page`
- `add_media_page`
- `add_links_page`
- `add_pages_page`
- `add_comments_page`
- `add_theme_page`
- `add_plugins_page`
- `add_users_page`
- `add_management_page`
- `add_options_page`
- `register_sidebar_widget`
- `wp_register_sidebar_widget`

This is done by resolving the hook that the function hooks into, and then inline the hooking function in the Super-CFG.

Finally, CTL and LTL requires all child nodes to have a self-loop, and the code for this can be seen in Appendix A.

4.2 Creating the nuXmv model and formulas

The code for generating the nuXmv model can be seen in Appendix B and Appendix C. The generator will build the model in a breath-first manner, ensuring that nodes are only visited once to avoid loops. When a node is processed, the node will create the CTLLTLNode object, resolve the names of the node, and whether or not it is a function call. If it is a function call, the call is stored, and it is checked if it is a call to `apply_filter`. If it is, the filter is added to a list of detected filters. Each node will be given an unique number corresponding to the node's state. Once the node has been generated, it will be added to a list of nodes, which is then ready for further processing.

After node generation, the BFS checks if it is an include statement, and if it is, another BFS search is launched recursively with the root of the included file. This ensures that the model will correctly model plugins with included files. Finally, the BFS will visit all child nodes of the node, and add them to the queue for processing.

After the model has been created, the file is written to disc. This is done by first writing a state called `state`, along with a boolean value for each filter and incorrect function. The incorrect functions covered are;

- `move_uploaded_file`
- `move_uploaded_file`
- `htmlspecialchars`
- `preg_match`
- `mysqli_real_escape_string`
- `mysql_real_escape_string`

After the generation of the mode, the CTL specifications are generated, and an example of the generated formulas are below, based on the analysis of the Post Types Order[5] plugin. The first 6 are always included, whereas 7-13

is generated according to the created filters. The first 6 specifications check if there exist a case in which one of the functions above has been found, that is, the boolean corresponding to the function has been set to true. For the filters, it is checked if every filter that has been opened is successfully closed. In the formulas `af` means "apply_filter" and `rf` means "remove_filter".

1. `SPEC AG(move_uploaded_file = FALSE);`
2. `SPEC AG(htmlentities = FALSE);`
3. `SPEC AG(htmlspecialchars = FALSE);`
4. `SPEC AG(preg_match = FALSE);`
5. `SPEC AG(mysqli_real_escape_string = FALSE);`
6. `SPEC AG(mysql_real_escape_string = FALSE);`
7. `SPEC AG(EF(af_pre_get_posts = TRUE -> rf_pre_get_posts = TRUE));`
8. `SPEC AG(EF(af_posts_orderby = TRUE -> rf_posts_orderby = TRUE));`
9. `SPEC AG(EF(af_init = TRUE -> rf_init = TRUE));`
10. `SPEC AG(EF(af_get_previous_post_where = TRUE -> rf_get_previous_post_where = TRUE));`
11. `SPEC AG(EF(af_get_previous_post_sort = TRUE -> rf_get_previous_post_sort = TRUE));`
12. `SPEC AG(EF(af_get_next_post_where = TRUE -> rf_get_next_post_where = TRUE));`
13. `SPEC AG(EF(af_get_next_post_sort = TRUE -> rf_get_next_post_sort = TRUE));`

After the file has been generated nuXmv is run on the file, and if some of the formulas are false, the tool will print a counter-example.

4.2.1 Example

An example of the nuXmv model for Listing 4.1 can be seen in Listing 4.2. The first line is the Module declaration, and the `main` module is the main entry point. Line 3-7 is the variable declarations, with the `state`'s first, followed by every function as a boolean value. Line 9-12 initialises the variables, starting at state 1, and setting each boolean value to `FALSE`, as no functions has been detected. Line 14-18 specifies the state transitions, and line 21-24 is generated based on which state the function call for the incorrect function is in, in this case `mysql_real_escape_string`. Since an incorrect function was detected, the value of the boolean is set to `TRUE`. nuXmv requires all paths to be exhaustive, which is the reason for line 23, which will keep the value of the boolean as-is. Line 26-28 is the specifications.

```
1  MODULE main
2
3  VAR
4     state: {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
5     move_uploaded_file : boolean;
6     htmlentities : boolean;
7     ...
8
9  ASSIGN
10    init(state) := 1;
11    init(move_uploaded_file) := FALSE;
12    ...
13
14 next(state) := case
15    (state=1) : {2};
16    (state=2) : {3};
17    ...
18    esac;
19 ...
20
21 next(mysql_real_escape_string) := case
22 state=14 : TRUE;
23 TRUE : mysql_real_escape_string;
24 esac;
25
26 ...
27 SPEC AG(mysqli_real_escape_string = FALSE);
28 ...
```

Listing 4.2: nuXmv model for Listing 4.1

Running nuXmv on the model and specification from Listing 4.2 yields the result from Listing 4.3. Line 1 and 2 is the specifications which are true

on the model. Line 3-18 is the proof for the specification being false, that is, there exist an `mysql_real_escape_string` function call in the program, which is in Listing 4.1 line 10.

```
1  ...
2  -- specification AG mysqli_real_escape_string = FALSE is true
3  -- specification AG mysql_real_escape_string = FALSE is false
4  -- as demonstrated by the following execution sequence
5  Trace Description: CTL Counterexample
6  Trace Type: Counterexample
7    -> State: 1.1 <-
8      state = 1
9      move_uploaded_file = FALSE
10     htmlentities = FALSE
11     htmlspecialchars = FALSE
12     preg_match = FALSE
13     mysqli_real_escape_string = FALSE
14     mysql_real_escape_string = FALSE
15   -> State: 1.2 <-
16     ...
17     state = 9
18     mysql_real_escape_string = TRUE
```

Listing 4.3: nuXmv output for Listing 4.2

4.3 Testing

The implementation is tested on 5 WordPress plugins; OpenFilter (see Appendix D), Form Maker [1], Post Types Order [5], Shortcodes Ultimate[7] and WordPress SEO by Yoast [14]. These plugins were selected randomly among the most popular plugins. The execution time has not been measured, as it finishes within a few minutes. None of the analysed plugins contained security errors apart from filters. The test files and graphs can be found on the attached CD.

4.3.1 OpenFilter

OpenFilter is a plugin that adds a filter which allows the upload of PHP files, but does not close it. This is a vulnerability, and it is detected, as seen in Appendix D. This can be quite a severe vulnerability, since every upload function using the WordPress mime filter will allow the uploading of PHP files.

4.3.2 Form Maker

Form Maker is a plugin for creating forms [1]. The plugin makes use of 5 filters, and none of these are closed, however only 4 of the open filters are detected. 4 of the filters are used internally by the plugin only, but if another plugin emerges with the same filter names (which should never happen), this can cause a conflict. The last filter modifies one of the WordPress standard filters; the `the_content` filter for content. Depending on what the opening of the filter does, this can cause issues.

4.3.3 Post Types Order

Post Types Order is a plugin for ordering posts and post types [5]. The plugin makes use of 7 filters, and does not close any of the filters. Most of these filters has generic names such as `_init`, so this can cause a potential clash with other plugins, which can cause unintended side effects.

4.3.4 Shortcodes Ultimate and WordPress SEO by Yoast

Shortcodes Ultimate provides shortcodes for a lot of different tasks [7]. WordPress SEO is a plugin for doing search engine optimisation [14]. None of these plugins had forgotten open filters, but a clash happened between WordPress SEO and a plugin called BBPress, because it hooked into a WordPress function too early [13]. This might have been possible to avoid if proper filtering had been conducted.

5 Conclusion

This thesis explores the creation of WordPress plugins that makes use of the WordPress functions instead of the PHP supplied ones, or home-made sanitisation functions. WordPress itself consists of a secure core, and the WordPress developers focuses a lot on security. However, this is not always the case for plugins and themes, where these can be created insecurely.

WordPress supplies functions for sanitisation of data, database access, file manipulation, and so on. Yet, not all developers make use of these functions correctly. Additionally, WordPress allows developers to change the internal workings of WordPress by overwriting the WordPress filters, and if these are not closed once the plugin developer is done, this might have unintended side effects.

Model Checking is used to detect incorrect function usage, along with open filters, and Computation Tree Logic and Linear Temporal Logic is used to reason about the model. The Model Checker nuXmv is used, and the model is generated using a proof-of-concept that is aware of WordPress specific function calls and workings, and able to generate CTL and LTL specifications for checking for incorrect functions, and open filters.

Testing of the tool shows that it does find some open filters, but that it is still clear that it is a proof-of-concept solution.

Bibliography

[1] *Form-Maker version 1.7.47*, Accessed 2. June, 2015. URL https://wordpress.org/plugins/form-maker/.

[2] *NuSMV: a new symbolic model checker*, Accessed 2. June, 2015. URL http://nusmv.fbk.eu/.

[3] *The nuXmv model checker*, Accessed 2. June, 2015. URL https://nuxmv.fbk.eu/.

[4] *OWASP project 2013 Top 10*, Accessed 2. June, 2015. URL https://www.owasp.org/index.php/Top_10_2013-Top_10.

[5] *post-types-order version 1.7.9*, Accessed 2. June, 2015. URL https://wordpress.org/plugins/post-types-order.

[6] *Programming Language Popularity*, Accessed 2. June, 2015. URL http://langpop.com/.

[7] *Shortcodes Ultimate version 4.9.7*, Accessed 2. June, 2015. URL https://wordpress.org/plugins/shortcodes-ultimate/.

[8] *Sony Pictures Hacked: the full story | The Verge*, Accessed 2. June, 2015. URL https://www.theverge.com/2014/12/8/7352581/sony-pictures-hacked-storystream.

[9] *TIOBE Software: Tiobe Index*, Accessed 2. June, 2015. URL http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[10] *Usage Statistics and Market Share of PHP for Websites*, Accessed 2. June, 2015. URL http://w3techs.com/technologies/details/pl-php/all/all.

[11] *Version2: CSC hack (Danish)*, Accessed 2. June, 2015. URL http://www.version2.dk/artikel/rigspolitiet-koerekort-register-hos-csc-er-blevet-hacket-52368.

[12] *Washington Post*, Accessed 2. June, 2015. URL http://www.washingtonpost.com/blogs/the-switch/wp/2015/05/14/the-syrian-electronic-army-just-hacked-the-washington-post-again/.

[13] *WordPress SEO by Yoast Changelog for 2.1.1*, Accessed 2. June, 2015. URL https://wordpress.org/plugins/wordpress-seo/changelog.

[14] *WordPress SEO by Yoast version 2.1.1*, Accessed 2. June, 2015. URL https://wordpress.org/plugins/wordpress-seo/.

[15] *WordPress › Plugin Security | Plugin Developer Handbook | WordPress Developer Resources*, Accessed 2. June, 2015. URL https://developer.wordpress.org/plugins/security/.

[16] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press, New York, NY, USA, 2004. ISBN 052154310X.

[17] Kenneth Jepsen, Morten Nørtoft, and Mikkel Vej. *Eir - Static Vulnerability Detection in PHP Applications*. 2015.

[18] Jens Thomas Vejlby Nielsen. *Securing Web Applications*. 2015.

[19] Sara Rosso, with contributions from Barry Abrahamson, Michael Adams, Jon Cave, Helen Hou-Sandí, Dion Hulse, Mo Jangda, and Paul Maiorana. *WordPress Security White Paper*. https://wordpress.org/-about/security/, Version 1.0, March 2015.

A

Adding self-loops

```
using QuickGraph;
using PHPAnalysis.Data.CFG;
using System.Linq;

namespace PHPAnalysis
{
  public interface ICFGCTLPreparation
  {
    void AddSelfLoops (BidirectionalGraph<CFGBlock, TaggedEdge<↩
        CFGBlock, EdgeTag>> graph);
  }

  public sealed class CFGCTLPreparation : ICFGCTLPreparation
  {
    //Preparation for CTL requires all nodes without an outgoing ↩
        edge to have a loop to self

    public void AddSelfLoops (BidirectionalGraph<CFGBlock, ↩
        TaggedEdge<CFGBlock, EdgeTag>> graph)
    {
      foreach (var vertex in graph.Vertices)
      {
        if (!graph.OutEdges(vertex).Any())
        {
          graph.AddEdge (new TaggedEdge<CFGBlock, EdgeTag> (↩
              vertex, vertex, new EdgeTag (EdgeType.Normal)));
        }
      }
    }
  }
```

```
27 }
```

Creating the nuXmv model

```
1  using System;
2  using QuickGraph;
3  using PHPAnalysis.Data.CFG;
4  using System.Threading;
5  using System.Threading.Tasks;
6  using QuickGraph.Algorithms;
7  using System.Linq;
8  using PHPAnalysis.Utils;
9  using System.Collections;
10 using System.Collections.Generic;
11 using PHPAnalysis.Data;
12 using PHPAnalysis.Utils.XmlHelpers;
13 using PHPAnalysis.Analysis.AST;
14 using PHPAnalysis.Data;
15 using File = PHPAnalysis.Data.File;
16 using System.Diagnostics;
17
18
19 namespace PHPAnalysis.Analysis.CTLLTL
20 {
21   public class CTLLTL
22   {
23
24     private BidirectionalGraph<CFGBlock, TaggedEdge<CFGBlock, ↩
           EdgeTag>> graph;
25     private List<CTLLTLNode> nodeList = new List<CTLLTLNode> ();
26     private IncludeResolver resolver;
27
28
29     //Simple proof-of-concept list of functions to watch out for
```

```
30    private readonly string[] _AlertFunctions = {
31      //Filters
32  //  "add_filter",
33  //  "remove_filter",
34
35      //Upload
36      "move_uploaded_file",
37
38      //XSS
39      "htmlentities",
40      "htmlspecialchars",
41      "preg_match",
42
43      //SQL Injection
44      "mysqli_real_escape_string",
45      "mysql_real_escape_string"
46    };
47
48    public CTLLTL ()
49    {
50    }
51
52    private HashSet<string> filterList = new HashSet<string> ();
53
54    private void GetBlockName (CTLLTLNode block)
55    {
56      string nodeName = (nodeList.Count + 1).ToString ();
57      if (block.block.IsSpecialBlock || block.block.AstEntryNode ↩
            == null) {
58
59      } else {
60        switch (block.block.AstEntryNode.LocalName) {
61        case (AstConstants.Nodes.Expr_FuncCall):
62        case (AstConstants.Nodes.Expr_MethodCall):
63          string methodName = "";
64          try{methodName = block.block.AstEntryNode.GetSubNode (↩
                AstConstants.Subnode + ":" + AstConstants.Subnodes.↩
                Name).FirstChild.GetSubNode (AstConstants.Subnode + ↩
                ":" + AstConstants.Subnodes.Parts).InnerText;} catch↩
                 (Exception e){};
65          if (methodName == "add_filter") {
66            var filterName = block.block.AstEntryNode.GetSubNode (↩
                  AstConstants.Subnode + ":" + AstConstants.Subnodes↩
                  .Args)
67              .FirstChild.FirstChild
68              .GetSubNode (AstConstants.Subnode + ":" + ↩
                    AstConstants.Subnodes.Value).FirstChild.↩
                    LastChild.InnerText;
69            block.methodName = methodName;
```

```
70            block.filterName = filterName;
71            filterList.Add (filterName);
72          } else
73            block.methodName = methodName;
74          nodeName = nodeName + "Call_" + block.methodName;
75          break;
76        default:
77          nodeName = nodeName + block.block.AstEntryNode.↩
                LocalName.ToLower ();
78          break;
79        }
80        //Handle the harder cases!
81      }
82      block.nodeName = nodeName;
83    }
84
85    //HashSet<CFGBlock> CnodesVisited = new HashSet<CFGBlock>();
86    private CTLLTLNode MakeCNode (CFGBlock block)
87    {
88
89      var v = Stopwatch.StartNew();
90      if (nodeList.Exists (x => x.block == block))
91        return null;
92      v.Stop();
93
94      var cNode = new CTLLTLNode ();
95      cNode.block = block;
96      GetBlockName (cNode);
97      cNode.nodeName = nodeList.Count + 1 + "";
98      nodeList.Add (cNode);
99      //CnodesVisited.Add(cNode.block);
100     if (cNode.block.AstEntryNode != null && (nodeList.Count % ↩
            1000) == 0)
101       Console.WriteLine("Added node: " + cNode.block.↩
              AstEntryNode.LocalName + " total in list: " + ↩
              nodeList.Count + " Loop took: " + v.Elapsed);
102     return cNode;
103   }
104
105
106   //Really not a pretty way to make this, but it works
107   private void WriteToFile (string path)
108   {
109     using (System.IO.StreamWriter file = new System.IO.↩
            StreamWriter (path)) {
110       string s = " ";
111       string states = "";
112       Console.WriteLine("Writing var declarations...");
113       foreach (var v in nodeList)
```

```
114           states += v.nodeName + ",";
115         states = states.Remove (states.Length - 1); //Remove the ↩
              last ,
116
117         file.WriteLine ("MODULE main");
118         file.WriteLine ();
119         file.WriteLine ("VAR");
120         file.WriteLine (s + "state: {" + states + "};");
121
122         foreach (var v in _AlertFunctions) {
123           file.WriteLine (s + v + " : boolean;");
124         }
125
126         foreach (var v in filterList) {
127           file.WriteLine (s + "af_" + v + ": boolean;");
128           file.WriteLine (s + "rf_" + v + ": boolean;");
129         }
130
131         Console.WriteLine("Writing Assignments...");
132         file.WriteLine ();
133         file.WriteLine ("ASSIGN");
134         file.WriteLine (s + "init(state) := " + nodeList [0].↩
              nodeName + ";");
135         foreach (var v in _AlertFunctions) {
136           file.WriteLine (s + "init(" + v + ") := FALSE;");
137         }
138
139         foreach (var v in filterList) {
140           file.WriteLine (s + "init(af_" + v + ") := FALSE;");
141           file.WriteLine (s + "init(rf_" + v + ") := FALSE;");
142         }
143
144         Console.WriteLine("Writing state if's");
145         file.WriteLine ();
146         file.WriteLine ("next(state) := case");
147         foreach (var v in nodeList) {
148           file.WriteLine (s + v.nodeText);
149         }
150
151         file.WriteLine (s + "esac;");
152
153         Console.WriteLine("Writing filters and alert functions...↩
              ");
154         WriteSpecialStuff (file);
155
156         Console.WriteLine("Generating Specification..");
157         GenSpec (file);
158       }
159     }
```

```
160
161     private void GenSpec (System.IO.StreamWriter file)
162     {
163       file.WriteLine();
164       foreach (var s in _AlertFunctions)
165       {
166         file.WriteLine("SPEC AG(" + s +" = FALSE);");
167       }
168
169       foreach (var filter in filterList)
170       {
171         file.WriteLine("SPEC AG(EF(af_" + filter + " = TRUE -> ↩
              rf_" +filter+" = TRUE));");
172       }
173     }
174     private void WriteSpecialStuff (System.IO.StreamWriter file)
175     {
176       Console.WriteLine("Functions...");
177       foreach (string node in _AlertFunctions) {
178         List<CTLLTLNode> list = nodeList.FindAll (v => v.↩
              methodName == node);
179         file.WriteLine ();
180         file.WriteLine ("next(" + node + ") := case");
181         foreach (var q in list) {
182           file.WriteLine ("state" + "=" + q.nodeName + " : TRUE;");
183         }
184         file.WriteLine ("TRUE : " + node + ";");
185         file.WriteLine ("esac;");
186       }
187
188       Console.WriteLine("Filters...");
189       foreach (string filtername in filterList) {
190         List<CTLLTLNode> nlist = nodeList.FindAll (v => v.↩
              filterName == filtername && v.methodName == "↩
              add_filter");
191         file.WriteLine ("next(af_" + filtername + ") := case");
192         foreach (var node in nlist) {
193           file.WriteLine ("state =" + node.nodeName + " : TRUE;");
194         }
195         file.WriteLine ("TRUE : af_" + filtername + ";");
196         file.WriteLine ("esac;");
197
198         nlist = nodeList.FindAll (v => v.filterName == filtername↩
               && v.methodName == "remove_filter");
199         file.WriteLine ("next(rf_" + filtername + ") := case");
200         foreach (var node in nlist) {
201           file.WriteLine ("state =" + node.nodeName + " : TRUE;");
202         }
203         file.WriteLine ("TRUE : rf_" + filtername + ";");
```

```
204         file.WriteLine ("esac;");
205       }
206     }
207
208     private void GenerateIf ()
209     {
210       //This was slow before... now it is significantly quicker
211       Parallel.ForEach(nodeList, cNode => {
212         var v = Stopwatch.StartNew();
213         cNode.nodeText = "(state=" + cNode.nodeName + ") : {";
214         foreach (var edge in cNode.graph.OutEdges(cNode.block)) {
215           cNode.nodeText += nodeList.Find (x => x.block == edge.↩
                Target).nodeName + ",";
216         }
217         cNode.nodeText = cNode.nodeText.Remove (cNode.nodeText.↩
              Length - 1); //Remove last ,
218         cNode.nodeText += "};";
219
220         v.Stop();
221         int number;
222         bool result = Int32.TryParse(cNode.nodeName, out number);
223         if (result && (number % 1000) == 0)
224           Console.WriteLine("Generated if for node: " + cNode.↩
                nodeName + " of " + nodeList.Count + " it took: " + ↩
                v.Elapsed);
225       });
226     }
227
228
229     HashSet<CFGBlock> visited = new HashSet<CFGBlock> ();
230     HashSet<File> inFile = new HashSet<File>();
231     int BFSRUNS = 1;
232     int activebfs = 1;
233     private void BFS (CFGBlock root, BidirectionalGraph<CFGBlock,↩
          TaggedEdge<CFGBlock, EdgeTag>> _graph)
234     {
235       Console.WriteLine("Total BFS recursions: " + BFSRUNS + " ↩
            Active BFS: " + activebfs + " nodes currently in graph:↩
             " + nodeList.Count);
236       BFSRUNS++;
237       activebfs++;
238       Queue<CFGBlock> queue = new Queue<CFGBlock> ();
239       queue.Enqueue (root);
240       while (queue.Any ()) {
241         var node = queue.Dequeue ();
242         if (visited.Contains(node))
243           continue;
244         visited.Add (node);
245         var cNode = MakeCNode (node);
```

```
246         if (cNode != null)
247           cNode.graph = _graph;
248 
249         if (node.AstEntryNode != null && node.AstEntryNode.↩
                LocalName == AstConstants.Nodes.Expr_Include) {
250           File output = null;
251           resolver.TryResolveInclude (node.AstEntryNode, out ↩
                output);
252           if (output != null && !inFile.Contains(output)) {
253             var _root = output.CFG.Roots ().Single (v => v.↩
                  IsSpecialBlock);
254             inFile.Add(output);
255             //Console.WriteLine("Recursive call: " + output.Name);
256             BFS (_root, (BidirectionalGraph<CFGBlock, TaggedEdge<↩
                  CFGBlock, EdgeTag>>)output.CFG);
257             //Console.WriteLine("Finished call: " + output.Name);
258             //Console.WriteLine("Still " + inFile.Count() + " ↩
                  files left");
259             inFile.Remove(output);
260           }
261         }
262 
263         foreach (var edge in _graph.OutEdges(node))
264           if (!visited.Contains (edge.Target)) //No loops, please
265             queue.Enqueue (edge.Target);
266       }
267       activebfs--;
268     }
269 
270     public void makeModel (BidirectionalGraph<CFGBlock, ↩
          TaggedEdge<CFGBlock, EdgeTag>> graph, IncludeResolver ↩
          resolver, string path)
271     {
272       this.graph = graph;
273       this.resolver = resolver;
274       var root = graph.Roots ().Single (v => v.IsSpecialBlock);
275 
276       BFS (root, this.graph);
277 
278       Console.WriteLine("Finished BFS Traversal, generating if ↩
            sentences...");
279       GenerateIf ();
280 
281       Console.WriteLine("Writing to file...");
282       WriteToFile (path);
283     }
284   }
285 }
```

C CTLLTLNode

```
using System;
using PHPAnalysis.Data.CFG;
using QuickGraph;

namespace PHPAnalysis
{
  public class CTLLTLNode
  {
    public CFGBlock block { get; set; }

    public string nodeName { get; set; }

    public string nodeText { get; set; }

    public string methodName = null;

    public string filterName = null;

    public BidirectionalGraph<CFGBlock, TaggedEdge<CFGBlock, ↩
        EdgeTag>> graph;
  }
}
```

D OpenFilter

D.1 OpenFilter code

```
<?php
/*
  Plugin Name: Open Filter
  Description: Opens a filter for uploading PHP files - very ↩
      vulnerable!
  Version: 1.0
  Author: Thomas Vejlby Nielsen
  Author URI: http://jtvn.dk
*/

function openfilter_upload_mimes ( $existing_mimes=array() ) {
  $existing_mimes['php'] = 'file/php';
  return $existing_mimes;
}

add_filter('upload_mimes', 'openfilter_upload_mimes');

//Do stuff that requires uploading PHP files using the WordPress↩
    upload functions..
```

D.2 OpenFilter graph

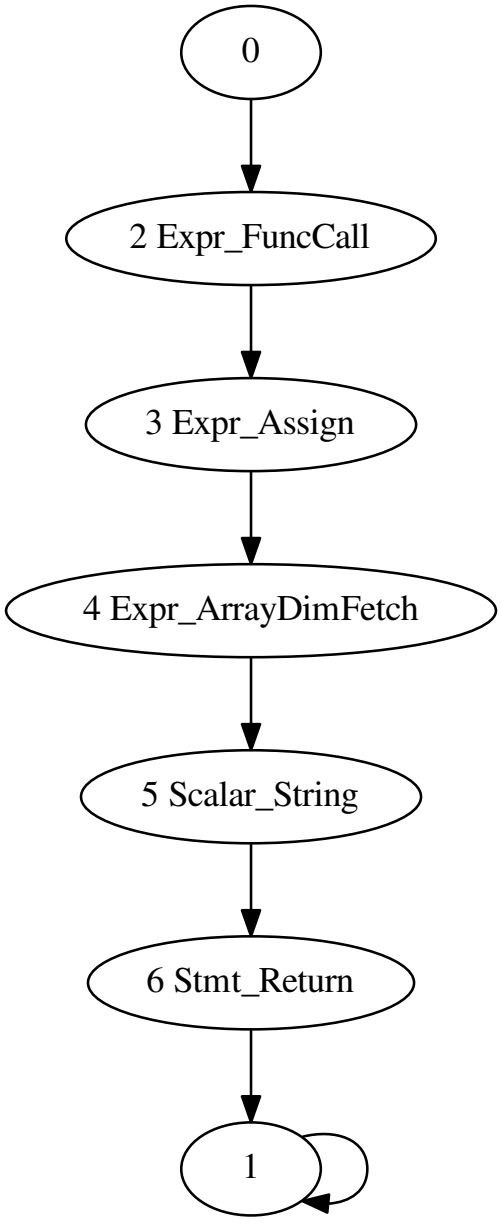

Figure D.1: OpenFilter Super-CFG

D.3 OpenFilter model

```
MODULE main

VAR
    state: {1,2,3,4,5,6,7};
    move_uploaded_file : boolean;
    htmlentities : boolean;
    htmlspecialchars : boolean;
    preg_match : boolean;
    mysqli_real_escape_string : boolean;
    mysql_real_escape_string : boolean;
    af_upload_mimes: boolean;
    rf_upload_mimes: boolean;

ASSIGN
    init(state) := 1;
    init(move_uploaded_file) := FALSE;
    init(htmlentities) := FALSE;
    init(htmlspecialchars) := FALSE;
    init(preg_match) := FALSE;
    init(mysqli_real_escape_string) := FALSE;
    init(mysql_real_escape_string) := FALSE;
    init(af_upload_mimes) := FALSE;
    init(rf_upload_mimes) := FALSE;

next(state) := case
    (state=1) : {2};
    (state=2) : {3};
    (state=3) : {4};
    (state=4) : {5};
    (state=5) : {6};
    (state=6) : {7};
    (state=7) : {7};
    esac;

next(move_uploaded_file) := case
TRUE : move_uploaded_file;
esac;

next(htmlentities) := case
TRUE : htmlentities;
esac;

next(htmlspecialchars) := case
TRUE : htmlspecialchars;
esac;
```

```
46
47 next(preg_match) := case
48 TRUE : preg_match;
49 esac;
50
51 next(mysqli_real_escape_string) := case
52 TRUE : mysqli_real_escape_string;
53 esac;
54
55 next(mysql_real_escape_string) := case
56 TRUE : mysql_real_escape_string;
57 esac;
58 next(af_upload_mimes) := case
59 state =2 : TRUE;
60 TRUE : af_upload_mimes;
61 esac;
62 next(rf_upload_mimes) := case
63 TRUE : rf_upload_mimes;
64 esac;
65
66 SPEC AG(move_uploaded_file = FALSE);
67 SPEC AG(htmlentities = FALSE);
68 SPEC AG(htmlspecialchars = FALSE);
69 SPEC AG(preg_match = FALSE);
70 SPEC AG(mysqli_real_escape_string = FALSE);
71 SPEC AG(mysql_real_escape_string = FALSE);
72 SPEC AG(EF(af_upload_mimes = TRUE -> rf_upload_mimes = TRUE));
```

D.4 OpenFilter nuXmv result

```
1  *** This is nuXmv 1.0.1 (compiled on Mon Nov 17 16:49:54 2014)
2  *** Copyright (c) 2014, Fondazione Bruno Kessler
3
4  *** For more information on nuXmv see https://nuxmv.fbk.eu
5  *** or email to <nuxmv@list.fbk.eu>.
6  *** Please report bugs at https://nuxmv.fbk.eu/bugs
7  *** (click on "Login Anonymously" to access)
8  *** Alternatively write to <nuxmv@list.fbk.eu>.
9
10 *** This version of nuXmv is linked to NuSMV 2.5.trunk.
11 *** For more information on NuSMV see <http://nusmv.fbk.eu>
12 *** or email to <nusmv-users@list.fbk.eu>.
13 *** Copyright (C) 2010-2014, Fondazione Bruno Kessler
14
15 *** This version of nuXmv is linked to the CUDD library version ↩
       2.4.1
16 *** Copyright (c) 1995-2004, Regents of the University of Colorado
```

```
17
18 *** This version of nuXmv is linked to the MiniSat SAT solver.
19 *** See http://minisat.se/MiniSat.html
20 *** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
21 *** Copyright (c) 2007-2010, Niklas Sorensson
22
23 *** This version of nuXmv is linked to MathSAT
24 *** Copyright (C) 2014 by Fondazione Bruno Kessler
25 *** Copyright (C) 2014 by University of Trento
26 *** See http://mathsat.fbk.eu
27
28 -- specification AG move_uploaded_file = FALSE is true
29 -- specification AG htmlentities = FALSE is true
30 -- specification AG htmlspecialchars = FALSE is true
31 -- specification AG preg_match = FALSE is true
32 -- specification AG mysqli_real_escape_string = FALSE is true
33 -- specification AG mysql_real_escape_string = FALSE is true
34 -- specification AG (EF (af_upload_mimes = TRUE -> ↩
    rf_upload_mimes = TRUE)) is false
35 -- as demonstrated by the following execution sequence
36 Trace Description: CTL Counterexample
37 Trace Type: Counterexample
38   -> State: 1.1 <-
39     state = 1
40     move_uploaded_file = FALSE
41     htmlentities = FALSE
42     htmlspecialchars = FALSE
43     preg_match = FALSE
44     mysqli_real_escape_string = FALSE
45     mysql_real_escape_string = FALSE
46     af_upload_mimes = FALSE
47     rf_upload_mimes = FALSE
48   -> State: 1.2 <-
49     state = 2
50   -> State: 1.3 <-
51     state = 3
52     af_upload_mimes = TRUE
```

E
CD

This page is intentionally left blank.