Instituto Superior Técnico - Alameda

Software Security Project Report

Static Code Analysis Tool

Discovering vulnerabilities in PHP Web Applications

Group 10

Rui Ventura 81045

Diogo Castilho 78233

November 20, 2017

Experimental part

Our analysis tool was conceived using the `Python` programming language, version `3.6.3`. It consists of a main component, the analyser (`analyser.py`), containing the logic for traversing some elements of an AST, and a pattern module (`pattern.py`) that houses the `Pattern` class, used to instantiate objects that represent vulnerable patterns which can be found in the `patterns` file, another component of the tool.

Components

Analyser

The main analyser component is run by invoking it and passing it a PHP program slice in JSON object format as a command line argument.

```
$ analyser.py /path/to/slice.json
```

Listing 1: Command line to run analyser

The slice being provided must already be in the form of an AST just like the ones produced by Glayzzle's PHP Parser [1]. Otherwise, the tool will not work as it's designed to only accept ASTs with that specific format.

Patterns

The `patterns` file contains a set of vulnerable patterns with the following format:

```
Vulnerability
Entry_1,Entry_2,...,Entry_i
Sanitizer_1,Sanitizer_2,...,Sanitizer_j
Sink_1,Sink_2,...,Sink_k
```

Listing 2: Vulnerable pattern template

where `Vulnerability` is the name of the vulnerability, `Entry` is an entry point, `Sanitizer` is a sanitization/validation function, and `Sink` a sensitive sink. Example:

```
SQL injection (PostgreSQL)
$_GET,$_POST,$_COOKIE,$_REQUEST
pg_escape_string,pg_escape_bytea
pg_query,pg_send_query
```

Listing 3: SQL Injection pattern, specific to PostgreSQL

Method

To start off, the patterns are fetched from the `patterns` file and parsed, generating a list of `Pattern` objects.

Then, the JSON formatted slice is loaded and the AST is converted into a Python dictionary, which is used throughout the analysis. Adopting a visitor-like pattern, the tool's able to analyse the nodes separately, which allows for some modifiability, yet not much, in the sense that, for another construct to be introduced, one or two function need(s) to be added to analyse the corresponding node.

During the traversal, a list of tainted symbols (variables) is carried along, as well as a dictionary of defined variables and their respective values (even if previously undefined). This allows us to perform, as we go, some basic taint analysis as we can detect when a tainted object is used in a sensitive sink.

Once such a case is detected, the program reports the vulnerability, suggests a set of possible sanitizations that can be used and, for simplicity's sake, exits, since we assumed not more than one vulnerability was present in the given slices of code.

Examples

Slice 6

```
echo $_POST['username'];
```

```
$ ./analyzer.py /path/to/slice6.json
Possible vulnerability detected: XSS
Please consider sanitizing tainted code with 1 of the following:
- htmlentities
- htmlspecialchars
...
```

Listing 4: Shortened example XSS vulnerable slice analysis output

Slice 8

```
$nis=$_POST['nis'];
if($indarg=="") {$query="SELECT * FROM siswa WHERE nis='$nis'";}
else {$query="SELECT * FROM siswa WHERE nis='$indarg'";}
$q=mysql_query($query,$koneksi);
```

```
$ ./analyzer.py /path/to/slice8.json
Possible vulnerability detected: SQL injection...
Please consider sanitizing tainted code with 1 of the following:
- mysql_escape_string
- mysql_real_escape_string
```

Listing 5: Shortened example SQLI vulnerable slice analysis output

Discussion

References

[1] Glayzzle and Various Contributors. PHP Parser. Available at `https://github.com/glayzzle/php-parser`. NodeJS PHP Parser - extract AST or tokens (PHP5 and PHP7).