TÉCNICO LISBOA

Segurança em Software

Discovering Vulnerabilities in PHP Web Applications

Grupo: 21

Rúben Martins, João Freitas, Rui Santos

79532, 81950, 88143

rubenjpmartins@gmail.com; joaotavaresfreitas@hotmail.com; rui.figueiredo.santos@tecnico.ulisboa.pt

Abstract
The aim of project consist to achieve an in-depth understanding of a security problem and carry out a hands-on approach to the problem, by implementing a tool for tackling it. Analyze its underlying security mechanism according to the guarantees that it offers, and to its intrinsic limitations and understand how the proposed solution relates to the state of the art of research on the security problem.

Key Words: *Software security, web applications, vulnerabilities analyses*

1. Introdução

O projeto consiste em resolver o problema de *discovery vulnerabilities in PHP web applications* [1], ou seja, consiste em estudar e identificar vulnerabilidades em aplicações da web que possam ser detetadas de uma forma estática por meio de análise de validação do registo de entrada. Isto porque, grande parte das vulnerabilidades dos programas permitem informações de entrada por parte do utilizador a valores de determinados parâmetros cujas funções são sensíveis à segurança, podendo comprometer a mesma.

Neste sentido, inicialmente o relatório, é centrado na arquitetura da ferramenta desenvolvida onde são identificadas as principais opções da arquitetura e alguns exemplos de resposta (*output*). Seguidamente, é realizada a discussão onde é efetuada uma explicação detalhada da capacidade da ferramenta em verificar imprecisões, como é que estes poderão ser minimizados e eventuais propostas de melhoramentos. Finalmente, culmina-se com as conclusões.

2. Arquitetura da solução

2.1. Características

Tendo em conta que as ferramentas de análise complexa são demasiado complexas, a ferramenta foi projetada para analisar *slices* (fatias) de programas PHP. Ao fazer isto, o problema torna-se muito semelhante à análise dinâmica.

Um *slice* é a sequência de todas as instruções que podem afetar um fluxo de dados num determinado ponto de entrada e de um coletor sensível (*sensitive sink*).

```
11: $u = $_GET['username'];
23: $q = "SELECT pass FROM users WHERE user='".$u."'";
24: $query = mysql_query($q);
```

Figura 1 – Exemplo de um *slice* [1]

Deste modo, a aplicação, que foi desenvolvida em *Python*, tem como objetivo a procura de pontos de entrada e coletores sensíveis (tais como os que estão presentes na figura 1), posteriormente tenta descobrir a função de sanitização correspondente e, caso exista, o *slice* será considerado como não vulnerável.

Tendo por base a definição de um *slice* elencado anteriormente, assume-se que o fluxo passará sempre pela função de sanitização/validação. Razão pela qual a estrutura da ferramenta, realiza sempre uma procura de padrões vulneráveis nos *slice*s, tais como [1]:

- Nome da vulnerabilidade;
- Um conjunto de pontos de entrada;
- Um conjunto de funções de sanitização/validação;
- E um conjunto de coletores sensíveis.

Os padrões (*patterns*) pode ser personalizados pelo utilizador[1] num ficheiro devendo respeitar uma estrutura do tipo do exemplo identificado na figura abaixo.

```
SQL_Injection
$_GET,$_POST,$_COOKIE,$_REQUEST,HTTP_GET_VARS,(...)
mysqli_escape_string,mysqli_real_escape_string
mysqli_query,mysqli_real_query, (...)

Cross_site_scripting_(XSS)
$_GET,$_POST,$_COOKIE,$_REQUEST,(...),$_FILES,$_SERVERS
htmlentities,htmlspecialchars,(...),urlencode,san_out
echo,print,printf,die,error,exit
```

Figura 2 – Exemplo do ficheiro de padrões

2.2. Alguns exemplos de resposta

Seguidamente serão apresentados alguns exemplos da extraídos da ferramenta desenvolvida de forma a ilustrar as respostas obtidas pelo mesmo (*output*) quanto testados os vários *slices*.

```
➜ ProjectoSSoft git:(master) ✗ python analyser.py slice1.json
Has vulnerability SQL_Injection, should use the function mysql_escape_string for sanitization
```

Figura 3 – Resposta do *slice* 1 – vulnerabilidade *SQL Injection*

Como se pode observar na figura anterior, regista-se uma vulnerabilidade do tipo SQL injection onde o nis é influenciado pelo ponto de entrada $_POST, a query influenciada pelo nis e o coletor sensível influenciado pela query. Logo a informação dada ao utilizador é que existe uma vulnerabilidade, indicando que devem ser usadas funções de sanitização mysql_escape_string, para correção.

Como a maioria dos *slices,* apresentam vulnerabilidades do tipo *SQL injection,* apenas iremos apresentar como exemplo os *slices* cujas vulnerabilidades sejam diferentes. Tal como, o *slice* 6.

```
➜ ProjectoSSoft git:(master) ✗ python analyser.py slice6.json
Has vulnerability Cross_site_scripting_(XSS), should use the function htmlentities for sanitization
```

Figura 4 – Resposta do *slice* 6 - vulnerabilidade XSS

Neste caso, a indicação apresentação informa o utilizador que está perante uma vulnerabilidade de cross site scripting e que deverá recorrer a funções de sanitização para *untainting* do tipo *htmlentities*.

Os slices fornecidos tinham todos algum tipo de vulnerabilidade. Assim, para efeitos de teste, foram criados novos slices para demonstrar o funcionamento onde estes não tinham vulnerabilidades. Nesse caso, o output esperado avisa que o slice não tem vulnerabilidades e qual foi a função de sanitização usada.

```
➜ ProjectoSSoft git:(master) ✗ python analyser.py slice1.1.json
There is no vulnerability, sanitization function mysql_escape_string was used
```

Figura 5 – Resposta do *slice* 1.1 – sem vulnerabilidade

[1] No caso foi configurado um ficheiro de padrões de acordo com a tabela apresentada em: http://awap.sourceforge.net/support.html [6]

3. Discussão

Dadas as limitações intrínsecas do problema de análise estática, o mecanismo utilizado é considerado impreciso. Pois pode ser: incorreto, produzir falsos negativos; incompleto, produzir falsos positivos; ou ambos.

Face ao exposto, seguidamente será efetuada uma análise do que a ferramenta é capaz de fazer face aos seguintes tópicos: Imprecisões da ferramenta em relação ao mecanismo utilizado; de que forma é que as imprecisões poderão ser minimizadas e por último, uma proposta de melhoramento.

3.1. Imprecisões da ferramenta

Tendo em conta que a ferramenta foi projetada para uma análise com uma profundidade limitada e para um número restrito de combinações, a mesma apresenta um elevado numero de imprecisões em comparação com uma ferramenta desenvolvida para uma análise com maior complexidade (maior profundidade).

É também importante referir que a ferramenta em causa foi estruturada para reconhecer apenas os padrões vulneráveis identificados na tabela referida anteriormente[2]. Sendo estes métodos intrínsecos à linguagem PHP foi assumido que qualquer entrada é corretamente validada e que cumpre o requisito de *soundness*.

Outra das imprecisões diz respeito ao fluxo do código PHP, pois a análise é feita sobre uma componente estática do programa, ou seja se existirem uma sequência de condições em que umas sanitizam uma dada variável e outras não, é necessário tomar uma decisão quanto há variável em questão originar um falso positivo ou um falso negativo. A ferramenta foi criada para originar falsos positivos, isto é mesmo que o fluxo do programa passe por uma função de sanitização da variável o programa continuará a ser vulnerável.

Os falsos positivos referidos acima podem ser contornados na nossa ferramenta se a variável vulnerável não influenciar diretamente as variáveis que irão influenciar a sensitive sink isto é por exemplo fazer condições em que se a variável vulnerável tiver um dado caracter, passa esse caracter a uma variável que será usada na sensitive sink.

No caso referido em cima teremos um falso negativo, pois a ferramenta não detetará a vulnerabilidade, quando a mesma existe.

Outra das imprecisões diz respeito ao mau uso das sanitizações no programa pois estas podem não ser as sanitizações necessárias para evitar uma dada vulnerabilidade, a ferramenta utiliza os padrões para ver se as sanitizações são devidamente usadas para uma slice com um dado par entry point/sensitive sink, garantindo assim que as sanitizações só têm relevância estiverem no mesmo padrão que o entry point e o sensitive sink.

Quanto à recuperação dos *slices* de programas potencialmente vulneráveis (através da análise *taint*), consideramos que o fluxo de informação não correcto foi omitido. Pois, esta hipótese baseia-se no facto de que todos os pontos de entrada (funções que permitem a entrada do utilizador) e todos os coletores sensíveis (entradas de utilizadores mal formatadas que podem comprometer o estado do sistema) formam comtemplados.

3.2. Minimização das imprecisões identificadas

Dos testes realizados, consideramos que o principal problema da aplicação consiste na identificação de falsos positivos (relato de não-vulnerabilidades). Neste sentido, como técnicas para minimização da deteção de falsos positivos propomos em aumentar o número do conjunto de funções de sanitização.

Não obstante, de acordo com os artigos disponibilizados, identificamos outras possíveis soluções, nomeadamente: habilitar o programa com alertas tipo (*type-awaress*) de forma a permitir reconhecer todas as variáveis que possam ser potenciais perigos, através de instruções eco como proposto por *Huang et al* [2] no subcapítulo 4.2. *Type-Aware Security Classes*; outra alternativa consiste, em recorrer a técnicas de *data mining* para prever os falsos positivos [3]; por último, é possível recorrer ao algoritmo de análise estática baseada em gramáticas [4], onde são modelados valores como contexto de gramáticas livres (CFG[3]) e operações de sequência de caracteres como transdutores de linguagem. Esta abordagem consegue rastrear os efeitos de operações de sequência de caracteres na estrutura de valores que fluem para os *hotspots* (construção onde a consulta ocorre) e apresenta resultados muitos positivos na deteção de falsos positivos.

Para a restrição nas funções de sanitização, como medida de minimização a considerar, prende-se com as funções de modificação de *string* (ex: *str_replace*) e funções de substituição recorrendo a expressões regulares (*ereg_replace* e *preg_replace*) como defendido por *Balzarotti et al* [5] na apresentação do *Saner* para validação de sanitização em aplicações *web*.

3.3. Proposta de melhoramento

Dadas as características do problema e as ferramentas disponíveis para exploração do problema, consideramos que a melhor maneira de aumentar a precisão passaria por expandir a biblioteca de funções de sanitização. Acreditamos que permitiria uma diminuição muito significativa do impacto associado à identificação de falsos positivos (como referido no ponto 3.2.). É, igualmente, considerado que esta medida não teria efeito significativo na eficiência da ferramenta.

Poderíamos ainda utilizar um método híbrido [3], isto é analisar a árvore do programa não apenas de maneira estática, mas também de maneira dinâmica. Este método iria eliminar bastantes casos de falsos positivos, pois analisaria o programa em execução e saberia as condições em que o código passaria, evitando assim que quando o código passasse em condições que teriam sanitizações das variáveis vulneráveis, que considerasse o programa vulnerável. A maior desvantagem desta abordagem vem do facto de a ferramenta ter que percorrer a árvore uma

[2] http://awap.sourceforge.net/support.html

[3] *Context Free Grammars*

vez em em modo estático e outra em modo dinâmico diminuindo assim a eficência da ferramenta.

A ferramenta poderia ainda ser melhorada se não fosse percorrida análise do programa uma vez por cada padrão, mas sim percorrida uma vez e analisando todos os padrões de uma só vez.

4. Conclusões

Tendo em conta os objetivos do projeto, considera-se que o mesmo foi cumprido com sucesso, uma vez que foi possível explorar os primeiros dez (10) *slices* ao ponto de conseguir identificar todas as vulnerabilidades apresentadas.

Assim como, o desenvolvimento da ferramenta contribuiu de uma forma muito positiva para o melhoramento das habilidades de pesquisa e identificação de vulnerabilidades do tipo *SQL Injection*, *Cross Site Scripting* e *PHP Code Injection*.

Referências

[1] A. Matos, "Segurança em Software," 2017. [Online]. Disponível: https://fenix.tecnico.ulisboa.pt/disciplinas/SSof251795/2017-2018/1-semestre/discovering-vulnerabilities-in-php-web-applications. [Acedido em 14 11 2017].

[2] Y. Huang, F. Yu, C. Tsai, D. Lee e S. Kuo, *Securing Web Application Code by Static Analysis and Runtime Protection,* New York, USA: ICWWW, 2004.

[3] I. Medeiros, N. Neves e M. Correia, "Automatic Detection and Correction of Web Application Vulnerabilities using Data Mining to Predict False Positives," em *Proceedings of the 23rd Internatinal Conference on World Web*, Seoul, Korea, 2014.

[4] G. Wassermann e Z. Su, *Sound and Precise Analysis of Web Applications for Injection Vulnerabilities,* California: PLDI, 2007.

[5] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel e G. Vigna, *Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications,* Austria: FWF, SBA and NSF, 2008.

[6] I. Medeiros, "Web Application Protection," FCT, 2017. [Online].
Disponivel:http://awap.sourceforge.net/support.html [Acedido em 19 11 2017].