An Approach for the Automated Detection of XSS Vulnerabilities in Web Templates

Sebastian Stigler sebastian.stigler@hs-aalen.de
Gulshat Karzhaubekova gulshat.karzhaubekova@hs-aalen.de
Christoph Karg christoph.karg@hs-aalen.de
Aalen University of Applied Sciences, Germany

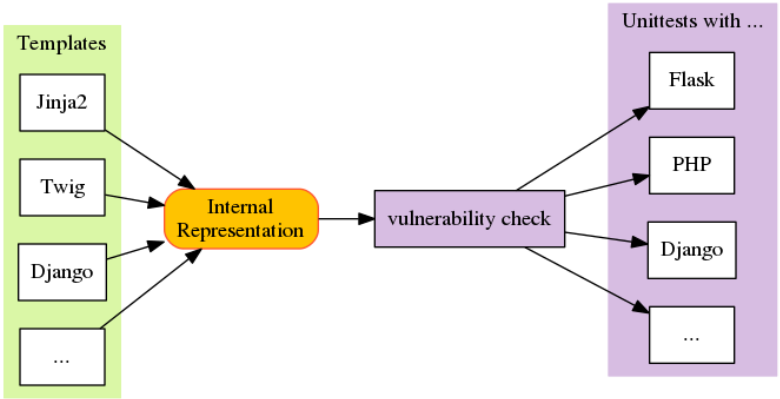
Motivation

- Web Frameworks are often predefined by the customer or already existing projects
- Templates from different Web Template Engines (WTE) look similar but don't behave completely equally
- Due to this facts, valid and secure code fragments from one WTE can lead to security risks in another WTE

Methodology

Architecture of our Approach

- Inspired by the LLVM Compiler



Source: own graphic

- Very easy to add a new Template Engine and unittest

Considered Web Template Engines

A WTE must comply with the following constraints for our approach

1. There is a restricted set of control structures (loop, condition, filter, setting and printing of variables)
2. There may be mechanisms to include other templates (inheritance, macros)
3. There is no way to write arbitrary code in the language of the backend within the template

Example of a Template

index.html

```
<!DOCTYPE html>
<html>
    <body>
        <h1>Hello world</h1>
      {% for name in usernames %}
        <p>Hello {{ name }}</p>
      {% endfor %}
    </body>
</html>
```

The Internal Representation (IR)

- The IR is represented by a two-staged intertwined abstract syntax tree (AST)
- The first stage AST is constructed by parsing the pure HTML part of the template and by marking the places where template tags are with special tokens
- The second stage AST is built by walking the first stage AST in depth-first-order and parsing the entries of the special tokens from above
- While constructing the second stage AST we collect the free variables and the output variables in a list and build a data flow graph between all used variables

First stage AST of the Example

First stage AST of `index.html`

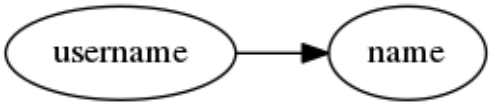
```
Declaration(raw="<!DOCTYPE html>")
StartTag(tag="html", attributes=[])
  StartTag(tag="body", attributes=[])
    StartTag(tag="h1", attributes=[])
      Data(raw="Hello world")
    EndTag(tag="h1")
    TemplateStatement(entry="for name in usernames")
    StartTag(tag="p", attributes=[])
      Data(raw="Hello ")
      TemplateExpression(entry="name")
    EndTag(tag="p")
    TemplateStatement(entry="endfor")
  EndTag(tag="body")
EndTag(tag="html")
```

Second stage AST of the Example

Second stage AST of `index.html`

```
1 Foreach(item=Variable(name="name", ref=None),
2         collection=Variable(name="usernames", ref=None),
3         ref=7)
4   Variable(name="name", ref=10)
5 EndForeach(ref=12)
```

Data flow graph of the Example



Source: own graphic

Finding vulnerabilities in templates

- Candidates are variables that reside in an expression template tag (the output variables)
- Using the data flow graph, we check if the value of this variable comes from a free variable
- Use 2nd stage AST to find all special tokens in 1st stage AST which influences the output variable
- Use the reference in 2nd stage AST to find the encapsulating html tag of the output variable in the 1st stage AST

Template for the unittest

- Generate from these nodes in the ASTs the template code and replace line 9 with it

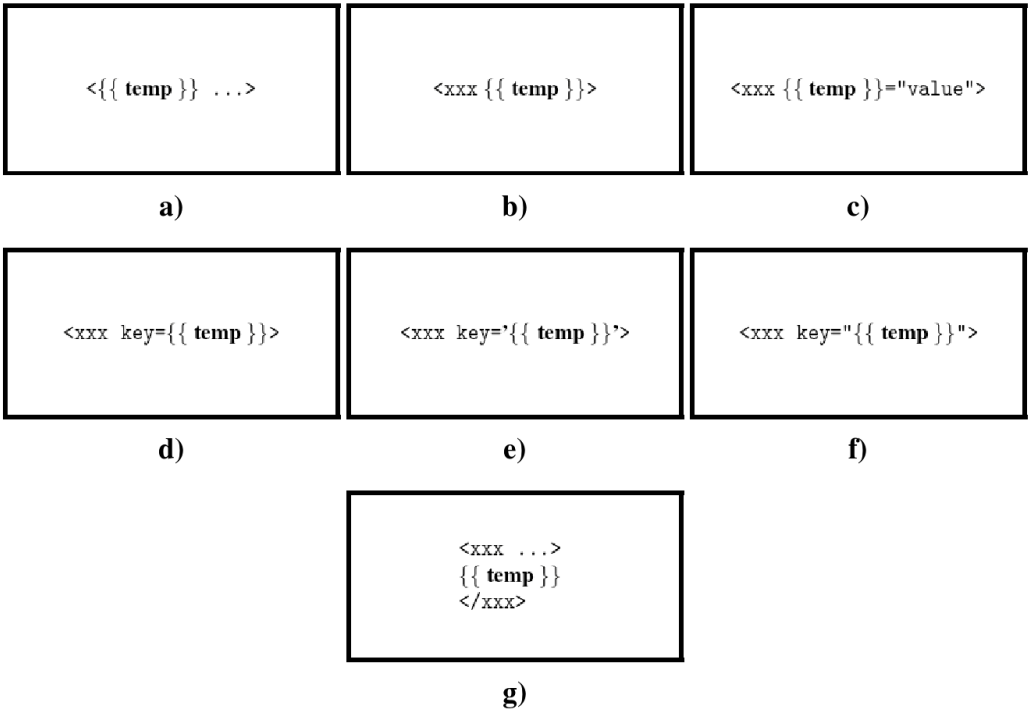
Skeleton for the template

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Unittest</title>
5          <script type="text/javascript">
6              function attack() { document.title="ATTACK"; };
7          </script>
8      <body>
9          <!-- <a onclick='attack()'>Attack</a> -->
10     </body>
11 </html>
```

- Use selenium webdriver to run the backend with this template and inject values for the free variables

Findings

Places for template tags in HTML



Source: own graphic

Attack on case e)

```
<xxx key='{{ temp }}'>
```

There are three cases for the value of `temp`:

`xxx` is clickable and `key` is **onclick**

```
attack();
```

`xxx` is clickable and `key` is not **onclick**

```
' onclick='attack();
```

`xxx` is not clickable

```
'><a onclick="attack();">click</a><xxx key='
```

Thank you for your attention!

This was

An Approach for the Automated Detection of XSS Vulnerabilities in Web Templates

Sebastian Stigler sebastian.stigler@hs-aalen.de
Gulshat Karzhaubekova gulshat.karzhaubekova@hs-aalen.de
Christoph Karg christoph.karg@hs-aalen.de

Questions?

References I

[1] Nils Adermann, Jordi Boggiano et al. *Composer*. URL: https://getcomposer.org (visited on 12/04/2018).

[2] Josip Bozic et al. "Attack Pattern-Based Combinatorial Testing with Constraints for Web Security Testing." In: *2015 IEEE International Conference on Software Quality, Reliability and Security* 6.4 (24 September 2015), pp. 207–212. DOI: 10.1109/QRS.2015.38.

[3] Eric Conrad, Seth Misenar and Joshua Feldman. "Chapter 7 - Domain 6: Security Assessment and Testing (Designing, Performing, and Analyzing Security Testing)". In: *CISSP Study Guide (Third Edition)*. Ed. by Eric Conrad, Seth Misenar and Joshua Feldman. Third Edition. Boston: Syngress, 2016, pp. 329–345. ISBN: 978-0-12-802437-9. DOI: B978-0-12-802437-9.00007-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128024379000072.

[4] Gabriel Diaz and Juan Ramon Bermejo. "Static analysis of source code security: Assessment of tools against SAMATE tests." In: *Information and Software Technology 55 (2013) 1462-1476* 55.2 (Aug. 2013), pp. 1462–1476. DOI: 10.1016/j.infsof.2013.02.005. URL: https://www.sciencedirect.com/science/article/pii/S0950584913000384.

References II

[5] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer Berlin Heidelberg, 2017. ISBN: 9783662536216. URL: https://books.google.de/books?id=FY5BvgAACAAJ.

[6] Katarina Goseva-Popstojanova and Andrei Perhinschi. "On the capability of static code analysis to detect vulnerabites." In: *Information and Software Technology 68 (2015) 18-33* 68.1 (Dezember 2015), pp. 18–33. DOI: 10.1016/j.infsof.2015.08.002. URL: https://www.sciencedirect.com/science/article/pii/S0950584915001366?via%3Dihub.

[7] Lars Hermerschmidt, Stephan Kugelmann and Bernhard Rumpe. "Towards More Security in Data Exchange: Defining Unparsers with Context-Sensitive Encoders for Context-Free Grammars". In: *2015 IEEE Security and Privacy Workshops* 8.6 (21-22 May 2015), pp. 134–141. DOI: 10.1109/SPW.2015.29.

[8] Chris Lattner. "Chapter 11 - LLVM". In: *The Architecture Of Open Source Applications*. Ed. by Amy Brown and Greg Wilson. lulu.com, 2012, pp. 151–166. ISBN: 978-1257638017. URL: http://aosabook.org/en/llvm.html.

[9] Mahmoud Mohammadi et al. "Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing." In: *UNC Charlotte, Charlotte, NC, USA. 2017. IEEE.* 10.5 (25-29 July 2017), pp. 364–372. DOI: 10.1109/QRS.2017.46.

References III

[10] Muhammad Parvez, Pavol Zavarsky and Nidal Khoury. "Analysis of Effectiveness of Black-Box Web Application Scanners in Detection of Stored SQL Injection and Stored XSS Vulnerabilities". In: *The 10-th International Conference for Internet Technology and Secured Transactions (ICITST-2015)*. 6.3 (14-16 December 2015), pp. 186–191. DOI: 10.1109/ICITST.2015.7412085.

[11] Fabien Potencier et al. *Twig*. URL: https://twig.symfony.com/doc/2.x/ (visited on 12/04/2018).

[12] Armin Ronacher et al. *Flask*. URL: http://flask.pocoo.org/docs/012 (visited on 12/04/2018).

[13] Armin Ronacher et al. *Jinja2*. URL: http://jinja.pocoo.org/docs/2.10/templates/ (visited on 12/04/2018).