Continuous deployment: applying security for web application development

May 9, 2022

Share

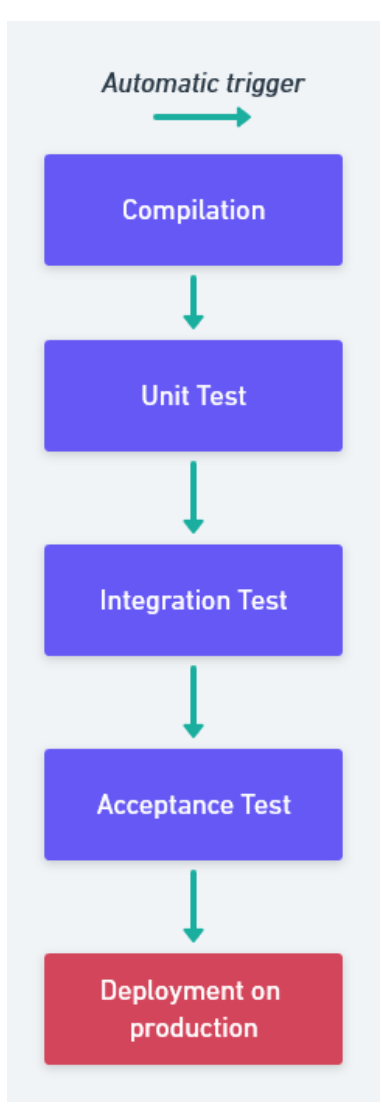
The rise of Continuous deployment

With the rise of the **Continuous Deployment**[1] activity, the frequency at which web applications (website, API, etc.) are deployed has significantly increased. Nowadays it is common to see companies deploying a new version of a web application several times a weeks/months[2].

Continuous deployment has a price to pay

With the increase in the frequency of deployments as well as the full automation of the deployment processes, the risk to introduce a problem allowing to attack a freshly deployed web application significantly increased. To be honest, the validation steps (unit test, integration test, etc.), in a continuous deployment pipeline, are critical as they represent the "watchdog" before the exposure of the application to end-users.

A common continuous deployment pipeline is like the following:



Often, these test steps (Unit, Integration, Acceptance, etc.) focus on the objective to ensure that the version deployed is functional from a business point of view (features do what they are expected to perform without bugs).

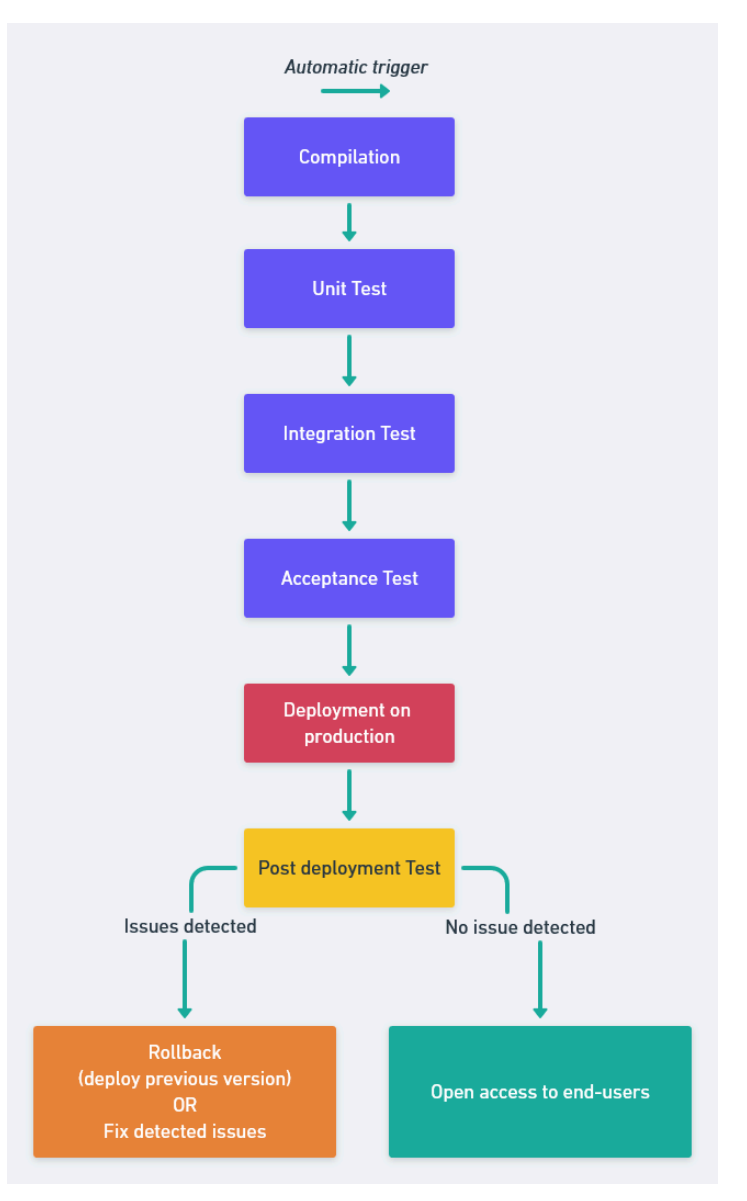
It is technically possible to add security-focused tests in these test steps to cover the security aspect. Even if it is not the objective of this post to present possible tests, an interesting talk about this topic, by the WE45 company (https://we45.com/), is provided here.

However, once the deployment on the production step is finished, doubts like the following remain:

- Does the version deployed only expose content that is expected to be accessible by end-users?
- Does the production configuration harden as expected?

Doubt removal

To try to remove the mentioned doubts, it is possible to add a final validation step to reach this continuous deployment pipeline:



This step, automatically triggered once the application is deployed, applies different security-focused validations. The objective is to ensure that the application is consistent with a production environment.

If issues are detected, then, two options are possible depending on the issues and level of automation achieved by the company in its continuous deployment activity:

- Option 1: Fix the detected issues leveraging automation on the components affected via web API provided by the components.
- Option 2: Trigger a continuous deployment pipeline to deploy the previous version.

If no issue is detected, then access to end-users (or no action) is performed depending on the deployment model of the application.

The table below provides a list of validations that can be performed in this final post-deployment step. In this table, every tool leveraged was chosen to perform processing without depending on an online service. The goal is to open the capability to either target an internal (Intranet) or an external (Internet) application. All chosen tools are free and open source.

Validation identifier	Validation objective	Tool used
VAL00	Ensure that all HTTP security headers applicable for the application topology are present and correctly defined.	– https://github.com/ovh/venom – Venom test plan following the *OWASP Secure Headers Project* recommendations: https://gist.github.com/righettod/f63548ebd96bed82269dcc3dfea27056
VAL01	Ensure that only a secure protocol is used (HTTPS).	– Curl combined with some bash commands: https://github.com/curl/curl
VAL02	Ensure that the TLS configuration is secure according to the current standard.	– https://github.com/drwetter/testssl.sh – JQ for results handling: https://github.com/stedolan/jq
VAL03	Ensure that no sensitive content, secrets, or unexpected content are exposed.	– https://github.com/ffuf/ffuf – Custom dictionary (text file specific to the application) of items (path/file) that must not be present after the deployment. – Curl commands to verify some potential information disclosure. – JQ for results handling: https://github.com/stedolan/jq
VAL04	Ensure that a security.txt file is present to allow security bug reporting in a secure way.	– Curl combined with some bash commands.
VAL05	Ensure that a Web Application Firewall is present in front of the application.	– https://github.com/stamparm/identYwaf
VAL06	Ensure that robots.txt file does not disclose any internal application path (absence of *disallow* clause).	– Curl combined with some bash commands.
VAL07	Ensure that directory listing is not enabled.	– Curl combined with some bash commands.

The validations above are a good foundation to start implementing a "post-deployment Test" step in a continuous deployment pipeline. They are straightforward and provide a rapid overview after deployment.

Proof of concept

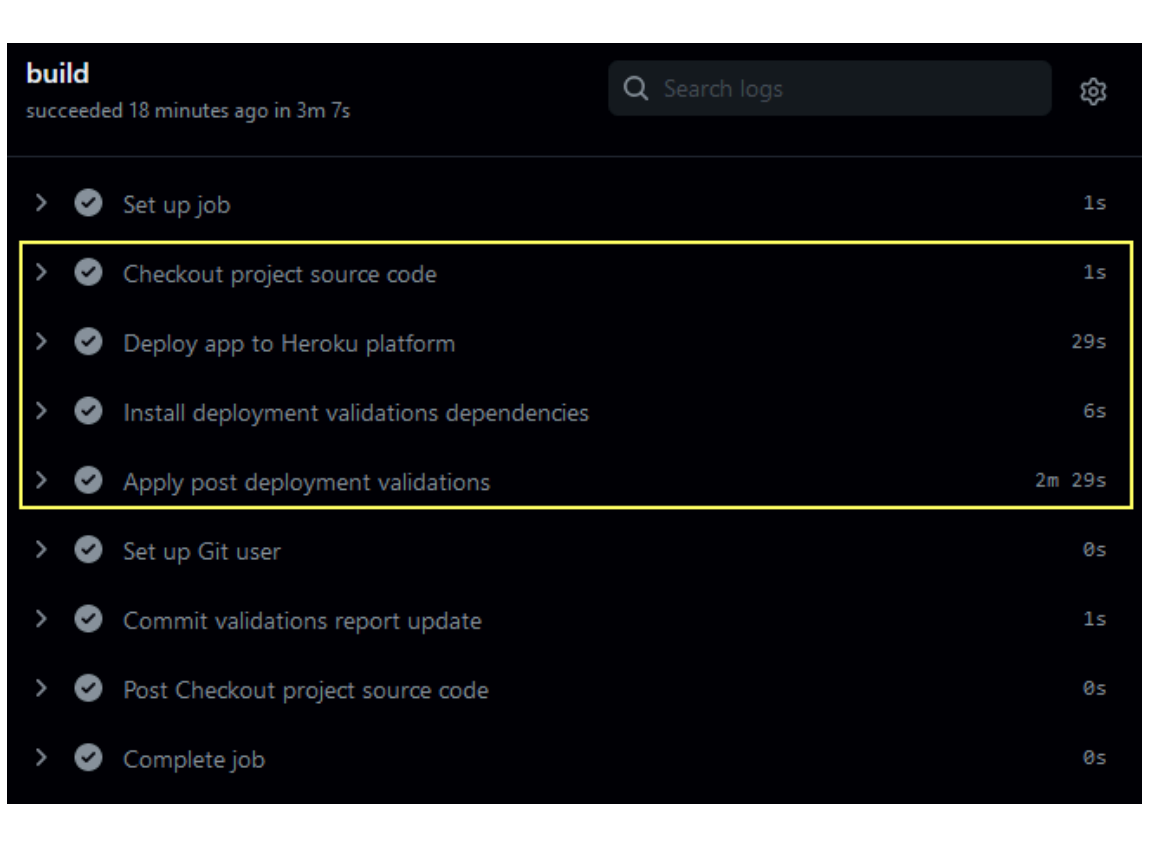
This script demonstrates an example of "low-level" implementation of the validations presented in the previous paragraph. Usage of a shell script allows to strongly customize validations according to the application and its deployment context.

Below is an example of the report generated, providing all the details about the different validations as well as a final state. The final state can be used to make the pipeline fail to trigger a rollback or other automated fixation operations:

```
[+] Execute 'validate_exposed_content'
/home/runner/work/PostDeploymentSecurityCheck-Study/PostDeploymentSe

deploy.key              [Status: 200, Size: 356, Words: 7, Lines: 7]
1 excluded item(s) found.
NodeJS Express framework usage disclosed (0 = no): 1
Error handling misconfiguration (0 = no): 0
[+] Execute 'validate_securitytxt_file_presence'
File is present (0 = no): 1
[+] Execute 'validate_waf_presence'
WAF is present (1 = no): 1
[+] Execute 'validate_robotstxt_file_content'
Disallow clause present 2 times (expected 0 time)
[+] Execute 'validate_directory_listing_enabling_status'
Directory listing is enabled.
[+] Cleanup
[+] Global status - RC: 15
[!] Issue found
```

Overview of the pipeline using GitHub action and the processing time of every step:



Validation operations stay short in terms of delay, less than 3 minutes. It is important to keep this delay the shortest as possible in order to[3]:

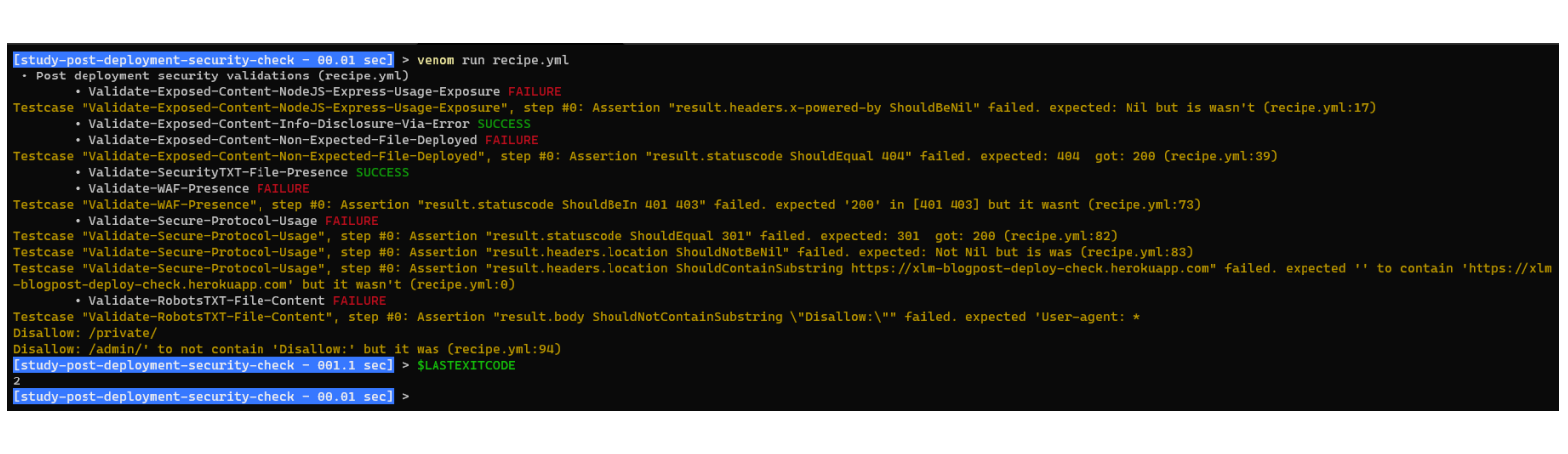
- Not impact parallel deployments of several applications by the continuous deployment platform.
- Provide quick feedback about a deployment, allowing running a deployment several times in case of need.
- Not monopolize resources for a long time frame.

Increase the maintainability

In the previous section, a shell script was used to perform the collection of security validations proposed. Even if it is a direct and effective way to achieve the validation steps, it can become difficult to maintain with the time and the increase of validation steps performed (in addition to being a platform-specific script). For the steps requiring only to perform HTTP requests (no execution of local tools like "testssl" for example), it is possible to move the collection of validations to a "recipe", which is easier to edit, maintain, test and be portable across different operating systems on which a continuous deployment platform can be installed.

The tool, named "venom"[4], can help to achieve the migration to a recipe via its "tests plan" approach and its cross-platform support.

This test plan demonstrates how it can be achieved (execution from a Windows machine):

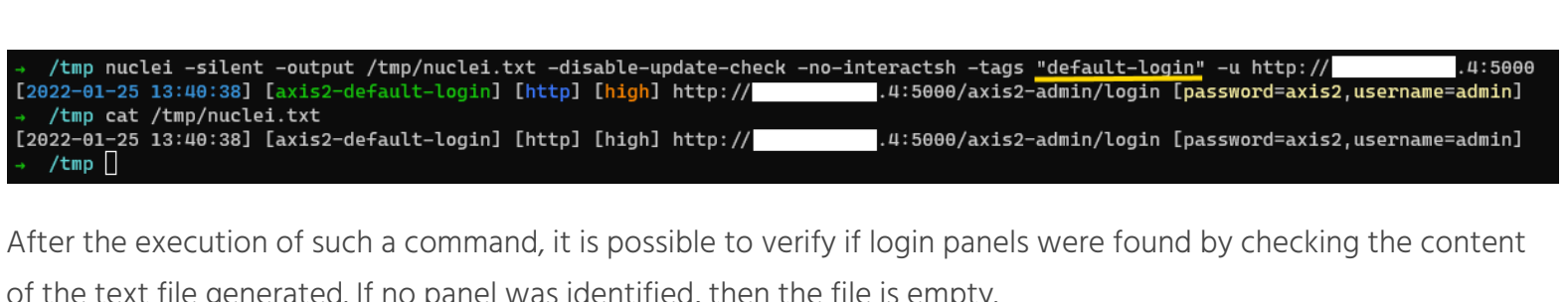


It is interesting to note that venom can execute local tools and deal with a generated report for the assertions part, therefore, it is possible to include operations requiring external tools to achieve a global test plan like the one implemented via a shell script. The only drawback of including the execution of external tools is that it broke the portability if the tools are not cross-platform. However, it is possible to keep the portability aspect via the creation of a dedicated ephemeral docker image containing the tools, the venom binary file, and the test plan.

Going further: additional suggestions for security validations

It is possible to add many more security tests, there is no limit. One suggestion can be to ensure that no administration interface, with default credentials, is left accessible, moreover, if the application is based on a product (for example the application is a custom module of a Content Management System). To achieve this, the tool, named "nuclei" (cross-platform), can be leveraged. In fact, via its approach based on templates, it provides a collection of "templates" to detect different kinds of administration interfaces. In case of need, custom templates can be created.

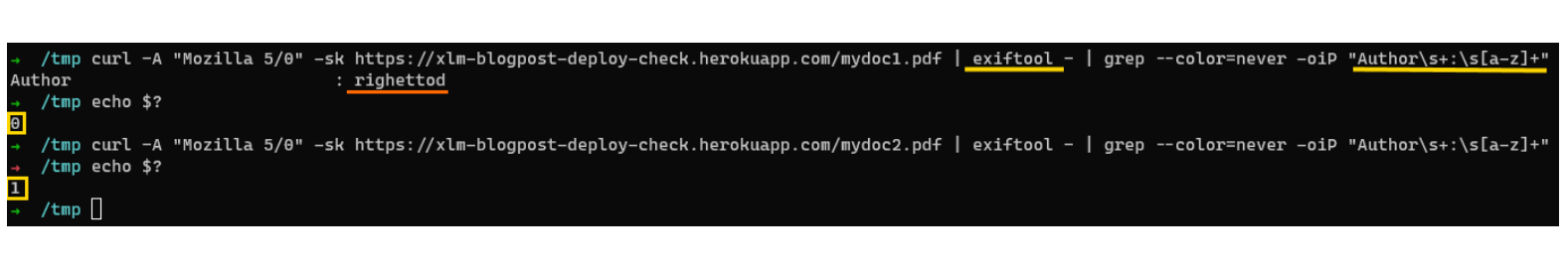
Below is an example of the usage of "nuclei" to identify every login panel with default credentials, the tag "default-login" instructs "nuclei" to apply all templates in charge of such detection:



After the execution of such a command, it is possible to verify if login panels were found by checking the content of the text file generated. If no panel was identified, then the file is empty.

If the application is delivering static Microsoft Office or PDF documents, then, another suggestion can be to ensure that these files do not disclose internal information like, for example, login or email via their metadata. Indeed, these kinds of information are interesting, from an attacker's perspective, in the phase of preparation of a phishing campaign or for a phase of the gathering of a collection of accounts in the context of an account takeover tentative.

The following command line leverages the tool, named "exiftool", to verify if published PDF documents contain login name using the format defined at the company level (not Excellium one here 😊) – the return code can be used, as an indicator, to identify if login names were found or not:



This validation is useful to ensure that common static documents like legal notices, privacy notices and so on are clean from a metadata perspective.

Going beyond the application itself

It is possible to add security validations not directly related to the deployed application itself. Every application relies on some configurations that are performed before the application was initially deployed. Even if these configurations do not change across several application deployments, it can be useful, from a security perspective, to ensure these parameters are not changed after a deployment operation. The objective is to detect any unexpected change as soon as possible to take remediation action.

One suggestion can be to ensure that a "CAA" DNS record is present on the application domain if the domain is a public one.

Extract from Gandi.nets documentation page :

"*The CAA record is a type of DNS record used to provide additional confirmation for the Certification Authority (CA) when validating an SSL certificate. This record allows you to specify which certification authorities are authorized to deliver SSL certificates for your domain.*"

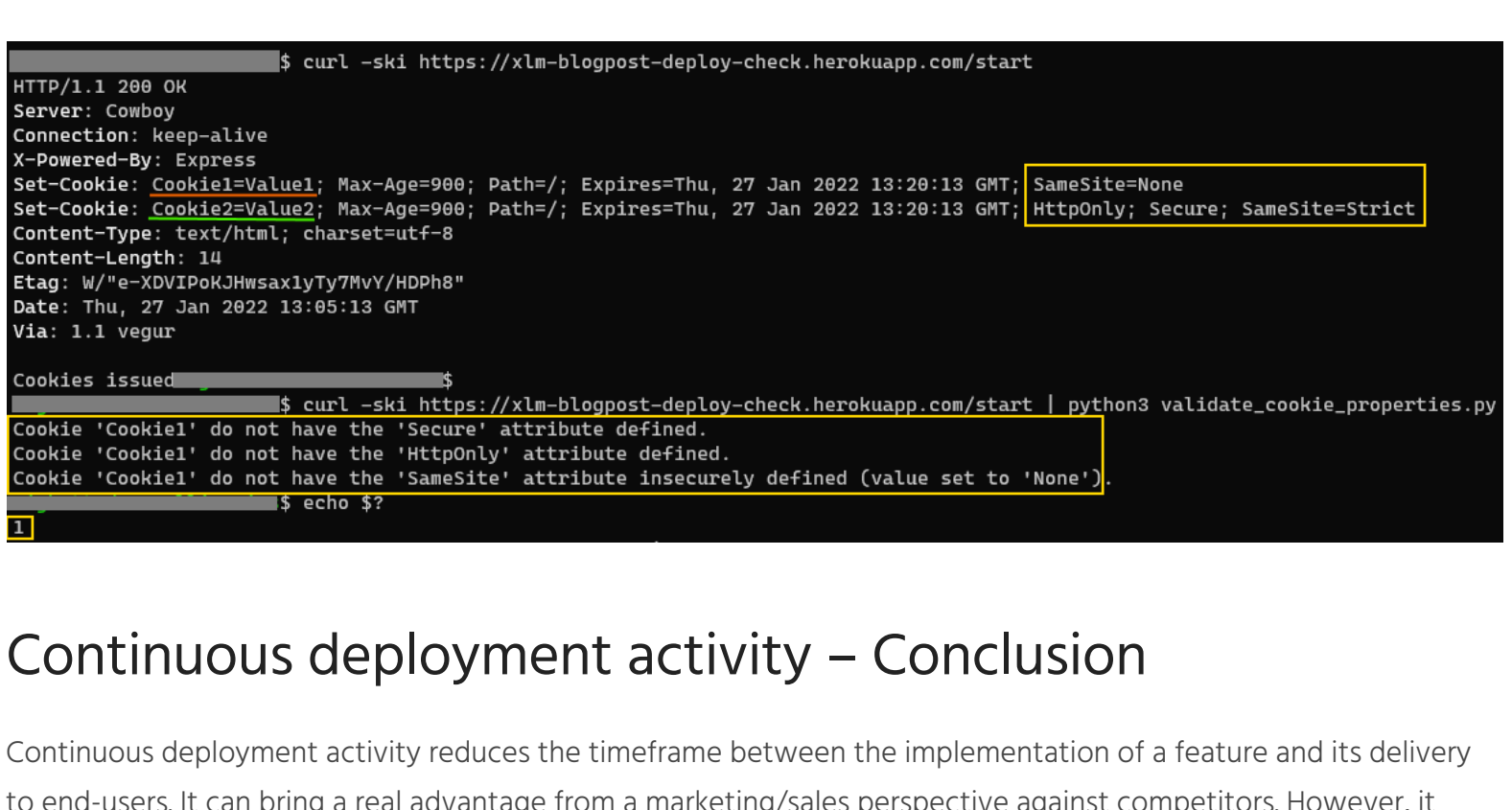
Extract from the Digicert documentation page:

> How the CAA RR process works
>
> Prior to issuing an SSL/TLS certificate for your domain, a CA (such as DigiCert) checks the CAA RRs to determine whether they can issue a certificate for your domain. A CA can issue a certificate for your domain if one of the following conditions is met:
>
> - They do not find a CAA RR for your domain.
> - They find a CAA RR for your domain that authorizes them to issue that type of certificate for it.
> - They find only CAA RRs without "issue" or "issuewild" property tags in them for your domain.

The validation can be performed using the following command line[5] and the return code can be used, as an indicator, to identify if a CAA record was found or not:

Another suggestion can be the following; if the application leverage cookies to carry information, ensure that they are correctly configured from a security perspective.

Unfortunately, "venom" does not have a convenient way to apply assertions on cookies, therefore, a python3 script can be used to apply the validations – the return code can be used, as an indicator, to identify if any non-well security configured cookie was found or not:



Continuous deployment activity – Conclusion

Continuous deployment activity reduces the timeframe between the implementation of a feature and its delivery to end-users. It can bring a real advantage from a marketing/sales perspective against competitors. However, it requires to be in full control posture regarding the product delivered to ensure that it did not represent a security risk for the provider. This blog post provided technical hints, to achieve this situation of control, and to fully benefit from a continuous deployment activity.

Feel free to use all provided hints/materials to build your own post-deployment security validations strategy 😉

Did you like the article? Find more blog articles right here.

Dominique Righetto

Alexis Pain

Valentin Giannini

References

↑1 https://www.atlassian.com/continuous-delivery/continuous-deployment

↑2 https://cloud.google.com/blog/products/devops-sre/another-way-to-gauge-your-devops-performance-according-to-dora

↑3 https://www.atlassian.com/continuous-delivery/continuous-integration

↑4 https://github.com/ovh/venom

↑5 https://github.com/projectdiscovery/nuclei/issues/1542

Uncategorized