

Geant-Validation

Gonzalo de la Cruz Fernández

September 1, 2017

Abstract

The aim of this document is to demonstrate the work performed in Geant-Validation application by Summer Student Gonzalo de la Cruz Fernández. In the following pages the architecture of Geant-Val and the main improvements and design decisions taken are described, as well as the main security challenges faced and what measures we have adopted against them.

1 Introduction

The EP-SFT group is leading the development of Geant4 [1] and GeantV [2] Monte Carlo simulation toolkits which are essential for the HEP experiments. Geant4 and GeantV simulate the passage of particles through matter using Monte Carlo methods. Simulations have applications in high energy, nuclear and accelerator physics, but also in other fields as medical and space sciences. Geant has new referece releases every month. As the number of toolkit versions increases also does the importance of performing regression and physics testing for comparing the simulation data with experimental data. Geant-Validation application allows the users to perform validation over the results obtained by different versions of Geant toolkit.

In the past validation of Geant results was performed using the GRID Validation system developed by technical student George Lestaris in 2013. The application was originally thought for keeping the results of *simplified calorimeter* test produced by running Geant4 tests on the GRID using DIRAC [3].

This application presented some serious shortcomings. The main problem was that the system was very difficult to scale. Inserting new tests in the system required the modification of database schema by manually creating new tables. It was based on DJANGO which caused some troubles to keep it updated and it used a very old version of DIRAC.

Last year summer student Ioana Ifrim created a prototype for Geant-Validation application based on modern technologies such as *Node.js* [4], *Express* [5], *AngularJS* [6] and *Bootstrap* [7]. The Geant-Validation development team has continued the improving the prototype with the objective of making Geant-Validation a powerful, intuitive, robust and secure tool for performing validation.

2 Service architecture

The architecture of Geant-Validation webpage follows a 3-tier architecture:

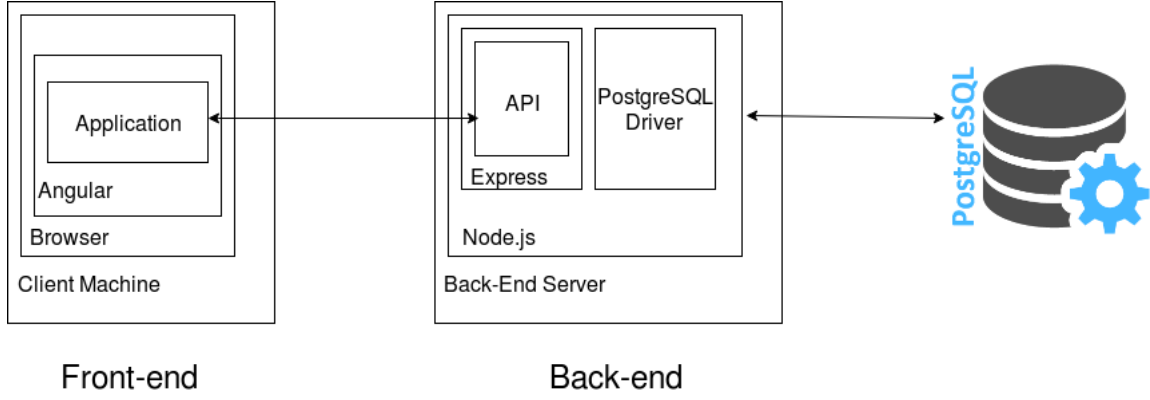


Figure 1: Geant-Validation application architecture.

2.1 Back-End

2.1.1 Node.js

Node.js [4] is an open source and multiplatform server framework using ECMAScript. It is asynchronous and has an event-driven architecture. The design of *Node.js* is focused on improving the throughput and scalability of the system. It is a great option for applications that have a lot of I/O operations and that don't need to perform CPU intensive operations. *Node.js* is aware of security and offers tools as the *Node Security Platform* for continuous security monitoring against known vulnerabilities.

Node.js was the chosen technology because of the following reasons:

- It is a fast and reliable framework.
- Geant-Val has a lot of I/O operations. Basically the user select some parameters as input and the application outputs the information stored in database that satisfy that conditions. *Node.js* is a perfect framework for this.
- Geant-Val does not perform any computation intensive tasks.
- Applications that use Node.js are easily scalable.
- *Node.js* framework is fast to learn and to develop in.
- *Node Package Manager* allows us to really easily administrate our packages and add new dependencies in a simple and secure way.

2.1.2 Express

Node.js is a low-level I/O mechanism. *Express* [5] is an open-source modular web application framework for *Node.js* that facilitates the creation of secure, modular and fast applications and the creation of RESTful API's.

2.1.3 PostgreSQL driver

For connecting the back-end with the database *Node-Postgres* [8] is used. This is a pure JavaScript client for *PostgreSQL*. It supports main *PostgreSQL* features such as connection pooling, prepared statements and name statements with query plan caching. The use of this client allow us to perform secure connections to database, to optimize query performance and avoid problems such as SQL injection by using prepared statements.

2.2 Front-End

2.2.1 AngularJS

AngularJS [6] is an open-source front-end framework for dynamic web applications. It's objective is to simplify the development of client-side Model-View-Controller (MVC) architecture applications. It allows to extend HTML syntax to express application components clearly.

HTML is a great language for static documents, but not for dynamic web applications. AngularJS offers additional HTML attributes which are interpreted as directives that bind I/O parts of the page to the model of the application.

2.3 Database

For the database we are using the *Database on Demand service* [9] that is supported by CERN IT Department. IT manages the database for us and they perform all the operations related with database administration (operating system and database engine updates, automatic backups, recovery service and user support)

We are using a relational database and *PostgreSQL* as Database management system (DBMS)

3 Statistical tests over histograms

At the beginning the validation of simulation results was performed by eye, looking the graphical representation of the histograms and comparing the differences between the results of distinct versions of Geant toolkits. Validation by eye can be performed when the amount of histograms to analyze is small, but it is impossible to perform when hundreds or thousands of histograms have to be analyzed.

In order to automatize and simplify the validation process statistical tests are introduced in the system. The goal is to quantify how different the results obtained by different versions of Geant are and how much the simulations results differ from experimental data.

Marilena Bandieramonte and her Summer Student Silvia are developing a C++ library to perform statistical tests over histograms. The idea is to integrate this new library in our application in order to perform statistical comparisons. We are collaborating with them in order to achieve this integration.

As a temporary solution and in order to set up all the necessary infrastructure to support statistical tests in Geant-Validation we are using our own code to calculate χ^2 test. The design decisions that we are going to face when we finally implement Marilena's solution can be anticipated by the usage of this temporary code.

Initial idea was to calculate the results of statistical tests on the fly when needed. Using this approach it was not necessary to store any result in database. After implementing this solution we realized that the process of calculating the results on the fly was very slow when hundreds of pairs of histograms have to be compared.

The most straightforward solution for avoiding this problem and increasing the efficiency of the system is to store the results of statistical tests between comparable histograms in database. This way, the statistical tests are only calculated once and in future executions the result is read from database. Database schema has been modified in order to store the results of statistical tests. The structure of the new introduced table is shown in next figure:

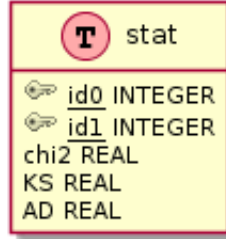


Figure 2: Stat table structure

Each row of the table stores the ids of the two histograms over which statistical tests have been performed. The other columns store the results of the statistical tests applied. We are interested in having χ^2 , *Kolmogorov-Smirnov* and *Anderson-Darling* tests, but in the future new statistical tests could be easily introduced adding new columns to *Stat* table. In order to store the results of the statistical tests two different approaches could be taken:

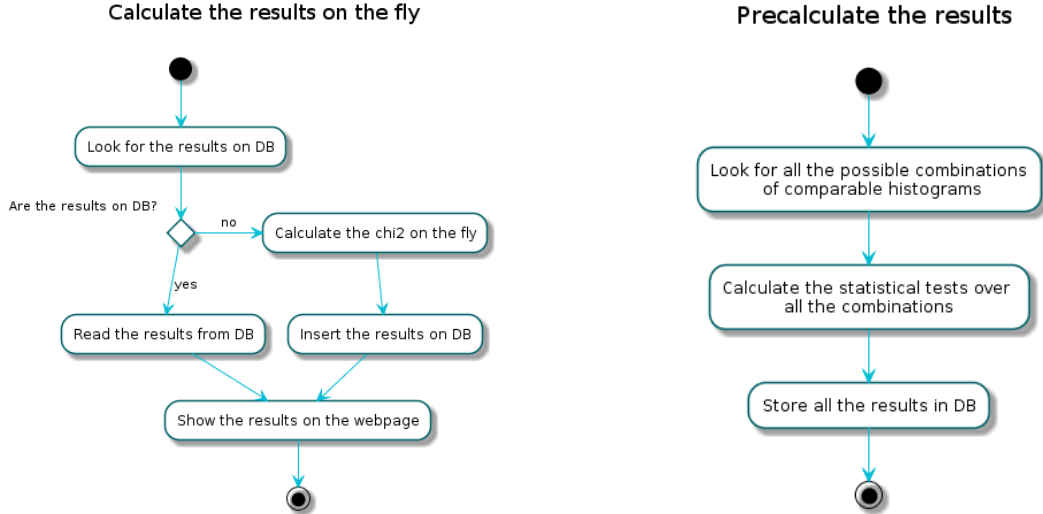


Figure 3: Activity diagram of both approaches

3.1 Calculate the results on the fly

This approach maintains the initial structure of calculating the results on the fly and once they are calculated they are stored in database. In the first execution, the application looks for the result in the database. As it is not in the database yet, the test is calculated on the fly and stored in database. For next executions the application only need to read the result from database. With this approach we are implementing a lazy evaluation of the test results: the results are calculated only when they are needed. This way we only have stored in database results that are being used. The main problem of this approach is that for the first executions it is even slower than our initial implementation. As we wanted the

performance of the application to be the same during its whole lifetime, the development team decided not to follow this approach.

3.2 Precalculate the results

The idea is to calculate and store in database the results of statistical tests over all the possible combinations of comparable histograms. In order to do this it is necessary to create a script that retrieves from database all pairs of compatible histograms, performs the statistical tests over them and store the results on database. The main advantage of this approach is that once all the results are calculated, reading from database is really fast and the performance is going to be the same. This solution has some shortcomings:

- **The amount of information that has to be stored in database is huge:** Database table is going to have as many rows as possible combinations of comparable histograms are in database. As the number of histograms stored increases, the number of combinations of compatible histograms increases exponentially.
- **When new histograms are inserted in database, new combinations have to be generated.**

After discussing between the different members of the team, we decided that this shortcomings were affordable for us because of the following reasons:

- The number of combinations of comparable histograms we have at the moment in database is affordable for precalculating the tests (about half a million of combinations)
- A database trigger can be implemented in order to automatically calculate all new combinations generated when inserting a new histogram.

The development team decided that precalculating the results was the optimal solution for increasing system performance.

4 Statistical Comparator

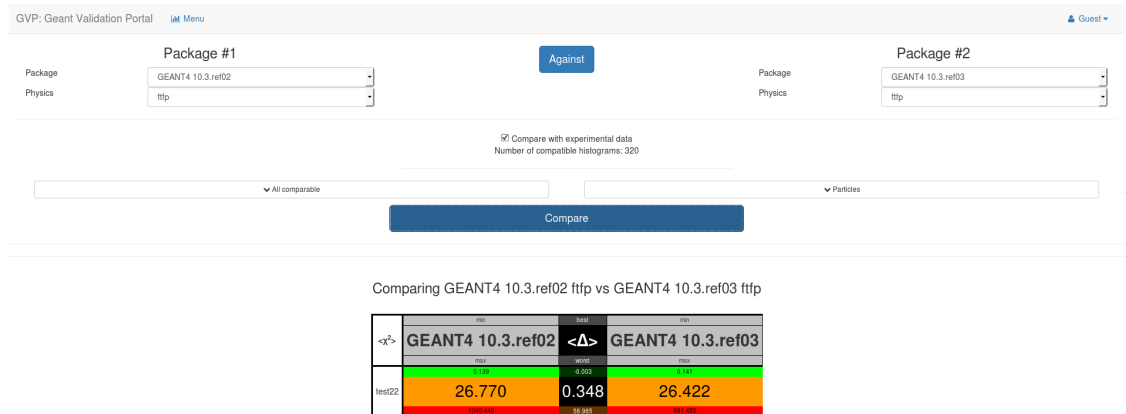


Figure 4: Statistical comparator comparing two Geant versions against experimental data

The introduction of statistical tests into the system open a new wide range of possibilities for implementing new features in the application. We have implemented a prototype of a

version comparator that allows to quantify how different two packages are between them taking into account the results of χ^2 test.

Statistical Comparator also allows to compare simulation data against experimental data. This way it is easy to know which versions of Geant generate results closer to real data.

At the moment the comparator is still an alpha version and its design may have changes in the future.

5 Performance Improvements

5.1 LaTeX formula rendering

Geant-Validation had a bottleneck when loading the page caused by slow formula rendering. *MathJax* [10] library was used in order to render formulas in some menus, and it rendered 5-6 seconds slower than the other elements of the page.

KaTeX [11] library was the appropriate solution for this problem. *KaTeX* provides a limited subset of the functionality provided by *MathJax*, but it works much faster. In our case, the formulas rendered about 25 times faster with *KaTeX* than with *MathJax*.

5.2 Speed improvements

Some code snippets inside the application needed some optimizations. *Node.js* is single threaded and data processing is not its strongest point. Taking this into account we decided to delegate great part of the filtering operations that were performed programatically to database. The SQL queries used to request database information were optimized in order to achieve a better performance.

6 User Interface

6.1 Dynamic menus

The application menus presented some problems that had to be solved. The menus only worked properly if the order of the dropdowns was respected. Otherwise, some problems and bugs could arise.

In order to avoid this the internal algorithm of the menus was totally rethought. The main objective is that the menus must show options that are consistent with the data stored in database. The new menus allow to start selecting from whatever parameter we want, and the other parameters are filtered taking into account previous selections.

6.2 Dynamic and interactive tables

Some parts of the application were using static HTML tables for showing data. *AngularJS* is such a powerful framework for front-end and makes very easy the creation of interactive applications. Using the *Angular-Tables* [12] module all the static table were replaced by dynamic and interactive tables.

χ^2 test

Observable	Beam	Model	Target	Beam energy	GEANT4: 10.4.beta01a with GEANT4: 10.4.beta01
Longitudinal Shower Profile	pi-	FTFP_BERT	AtlasFCAL	500	0.558791 (159194 and 158690)
Longitudinal Shower Profile	pi-	FTFP_BERT	TileCal	500	0.73088 (159135 and 158631)
energy resolution	pi-	FTFP_BERT	AtlasECAL	MULTIPLE	1.05167 (159380 and 158876)
energy resolution	pi-	FTFP_BERT	AtlasFCAL	MULTIPLE	1.14899 (159406 and 158902)
Longitudinal Shower Profile	pi-	FTFP_BERT	LhcbECAL	500	1.21012 (159328 and 158824)
Longitudinal Shower Profile	pi-	FTFP_BERT	AtlasECAL	500	1.37266 (159261 and 158757)
energy resolution	pi-	FTFP_BERT	LhcbECAL	MULTIPLE	1.42066 (159486 and 158982)
Longitudinal Shower Profile	pi-	FTFP_BERT	AtlasHEC	500	1.64387 (159128 and 158624)
energy resolution	pi-	FTFP_BERT	CmsECAL	MULTIPLE	1.99547 (159460 and 158956)
energy resolution	pi-	FTFP_BERT	TileCal	MULTIPLE	2.57128 (159513 and 159009)
energy resolution	pi-	FTFP_BERT	AtlasHEC	MULTIPLE	3.7931 (159433 and 158929)
Transverse Shower Profile	pi-	FTFP_BERT	LhcbECAL	500	11.5337 (159329 and 158825)
Transverse Shower Profile	pi-	FTFP_BERT	AtlasFCAL	500	12.6 (159195 and 158691)
Transverse Shower Profile	pi-	FTFP_BERT	AtlasECAL	500	13.853 (159262 and 158758)
Transverse Shower Profile	pi-	FTFP_BERT	AtlasHEC	500	19.4652 (159129 and 158625)
Transverse Shower Profile	pi-	FTFP_BERT	TileCal	500	21.0079 (159146 and 158642)
Transverse Shower Profile	pi-	FTFP_BERT	CmsECAL	500	22.4871 (159566 and 159062)
Longitudinal Shower Profile	pi-	FTFP_BERT	CmsECAL	500	39.5632 (159565 and 159061)

Figure 5: Interactive table showing χ^2 results

7 Security

7.1 Server Security

At the moment the web application is being hosted in a Linux virtual machine powered by *CERN Openstack* [13]. The firewall is configured to pass connections to ports 80 (HTTP) and 443 (HTTPS). This two ports are redirected to the ports used by Geant-Val application. We are considering the possibility of redirecting port 80 to port 443 in order to force SSL and then do port forwarding to the ports that the application is using.

The application has been enclosed in a *Docker* [14] container with limited capabilities. The use of a container adds an extra security layer. This way even if an attacker exploits a vulnerability in the system and manages to open a shell, he still couldn't get root access to the machine and the damage that he could make in the system is very limited. Containers also facilitate and automate system configuration.

CERN provides *Openshift* [15] in order to easily build, configure and deploy applications in containers. We are thinking about moving the container from our virtual machine in *OpenStack* to *Openshift*. As *Openshift* is a dedicated environment for deploying containers is more secure than deploying the container directly in our server. *Openshift* implements a multi-layered security solution using SELinux and other technologies. If someone is able to break out of SELinux, they are still in a container, and if someone manages to break out of the container and get root privileges SELinux keeps you stuck in a high security computing environment. As it is a solution supported by CERN, this is probably the optimal solution for deploying the application in a secure and robust environment.

The development team is in charge of maintaining updated all the technologies of the system in order to ensure that the latests security patches are applied. We have to take into account that even having the latests security updates we cannot defend ourselves against day-zero attacks.

There are other two important problems related with the implementation of the application that could compromise the security of the server: Command injection and dependency's vulnerabilities.

7.1.1 Command Injection

The Geant-Validation application depends on external tools in order to perform certain operations. The tools used are listed below:

- **Plotter:** Command-line tool developed in C++ that receives the ids of some histograms and generate a png file with the graphical representation of the histograms.
- **getChi2:** Command-line tool developed in C++ that receives the ids of two histograms and perform χ^2 test over them

This two tools are called in the REST API. Some API methods need to execute commands in the server and this is a possible critical point in the application. If the system is not properly coded an attacker could execute commands directly in the server and take control over the system. Several things were done in order to ensure the security of these methods:

1. **Perform input validation:** ensure that these two methods always receive valid ids. At the moment, the ids of the plots are always numerical. Validating that the parameters passed to the API methods are numeric should be enough to avoid command injection.
2. **Use *execFile()* and *spawn()* methods instead of using *exec()*:** *exec()* method is very problematic when user inputs are used as arguments in the command that is going to be executed. This method obliges the developer to concatenate strings for passing command arguments to the command. *execFile()* and *spawn()* take command arguments as an array, are not executed under a shell environment and do not manipulate the original command to run. This means that these two calls are not vulnerable to command injection. The calls to *exec()* have been substituted by calls to this two secure methods.

7.1.2 Secure dependencies

The use of *npm* [16] for managing dependencies is very convenient and comfortable. However, the packages installed can contain critical security vulnerabilities that can also affect the application. The security of the application is only as strong as the weakest link of its dependencies.

In order to guarantee the security of third-party packages we use the tool *nsp* [17], a command-line tool that checks in the *Node Security platform* [18] vulnerability database if the application uses packages with known vulnerabilities.

After analyzing the application the tool determined that it does not use any package with known vulnerabilities.

7.2 Data Integrity

As it was mentioned before, we are using a database on demand managed by IT. In principle it is an optimal solution because IT administrate it and maintain it for us. The application connects to database using a user with limited privileges. The application has read-only access to the database tables that it needs in order to work. Inserting, deleting and updating operations are not allowed.

Database is automatically backed up every day. Our database does not manage any sensible information that could be interesting for attackers. All the data stored is public and everyone can have access to it through our application.

Databases are the objective of numerous attacks, mainly SQL Injection attacks and XSS taking advantage of injection.

7.2.1 SQL Injection

In order to prevent this kind of attacks the REST API was analyzed in order to find possible SQL Injection vulnerabilities using *Sqlmap* [19]. It is an open-source penetration testing tool that allows to detect and exploit SQL injection flaws over database servers. The tool has also support to perform analysis over APIS.

In order to perform the analysis a list with the API methods that are going to be analyzed is provided to the tool. *Sqlmap* sends requests to the API methods testing different injection techniques until it finds one that works or determine that injection cannot be made over that method. Once it have found an injection point, *Sqlmap* provides features in order to easily take advantage of the vulnerability.

For the majority of the REST API methods the tool didn't find any vulnerability. Only one of the API methods analyzed was vulnerable against SQL Injection. This security problem was fixed and now the API is protected against injection.

7.3 Secure Information Transmission

An SSL certificate provided by CERN is used for creating secure and encrypted connections. There are also some security-related HTTP headers that any site developed with *Node.js* and Express should have. The following headers are being currently used in our application in order to prevent some security problems:

1. **Content-Security-Policy:** prevents cross-site scripting (XSS), clickjacking and other code injection attacks resulting from execution of malicious content in the webpage.
2. **X-Powered-By:** Specifies the technology supporting the web application. Can be used to exploit known vulnerabilities in Express or Node.js
3. **HPKP:** add Public Key Pinning headers to avoid Man in The Middle attacks (MITM) with fake certificates
4. **Strict-Transport-Security:** forces secure connections with the server (HTTP over SSL/TLS)
5. **X-Download-Options:** Specific for IE8. Avoids executing scripts in the context of the website
6. **X-Content-Type-Options:** Avoids that browsers can infer the MIME types and use it to execute scripts.
7. **X-XSS-Protection:** Establish XSS filter in modern browsers to detect and block reflected XSS.
8. **X-DNS-Prefetch-Control:** tells browsers whether they should do DNS prefetching.
9. **X-Frame-Options:** tells browsers to prevent your webpage from being put in an iframe. This mitigates clickjacking attacks.

This headers can be easily used in Express using *Helmet* [20] module. The installation is very simple and it is not intrusive with the code already implemented. This has been already implemented in our application.

7.4 User Security

In order to filter some features of the application that could not be useful for a common user of Geant-Val (e.g. view results of reference versions that are still being developed) an authentication system for CERN users has been introduced in the application. For performing the authentication process we are using the *OAuth2* [21] system provided by CERN. The application delegate the user authentication to CERN OAuth Authorization Service.

This solution guarantees that the authentication system that is being used is strong, robust and follow the guidelines required by CERN.

7.5 Code Security

All the application code is stored in one CERN's private *GitLab* [22] repository that can only be accessed by the development team. In this repository there are not stored any credentials or sensible information that could be used to access the system.

8 Acknowledgements

I would like to thank my supervisors Dmitry Konstantinov and Witold Pokorski for their trust and hospitality during my stay at CERN. Thank my coworkers Ivan Razumov and Grigory Latyshev with whom I have learned a lot. Acknowledgements to all the EP-SFT group for giving me this great opportunity for my personal and professional life.

References

- [1] CERN. Geant4 montecarlo simulation toolkit. <http://geant4.cern.ch/>. Accessed: 2017-08-31.
- [2] CERN. Geantv montecarlo simulation toolkit. <http://geant.cern.ch/>. Accessed: 2017-08-31.
- [3] DIRAC. Dimeson relativistic atom complex (dirac). <https://home.cern/about/experiments/dirac>. Accessed: 2017-08-31.
- [4] Node.js Developers. Node.js. <https://nodejs.org/en/>. Accessed: 2017-08-28.
- [5] StrongLoop TJ Holowaychuk. Express.js. <http://expressjs.com/>. Accessed: 2017-08-28.
- [6] Google. Angular.js. <https://angularjs.org/>. Accessed: 2017-08-28.
- [7] Twitter. Bootstrap. <http://getbootstrap.com/>. Accessed: 2017-08-31.
- [8] Brian C. Node-postgres. <https://github.com/brianc/node-postgres>. Accessed: 2017-08-28.
- [9] CERN IT Department. Database on demand. <http://information-technology.web.cern.ch/services/database-on-demand>. Accessed: 2017-08-28.
- [10] Khan Academy. Katex. <https://khan.github.io/KaTeX/>. Accessed: 2017-08-31.
- [11] Khan Academy. Katex. <https://khan.github.io/KaTeX/>. Accessed: 2017-08-31.
- [12] Mattias Holmlund. Angular-tablesort. <http://mattiash.github.io/angular-tablesort/>. Accessed: 2017-08-31.

- [13] CernVm. Cern openstack. <https://cernvm.cern.ch/portal/openstack>. Accessed: 2017-08-28.
- [14] Docker Inc. Docker. <https://www.docker.com/>. Accessed: 2017-08-28.
- [15] Red Hat. Openshift. <https://www.openshift.com/>. Accessed: 2017-08-28.
- [16] Isaac Z. Schlueter. Node package manager. <https://www.npmjs.com/>. Accessed: 2017-08-28.
- [17] The Node Security Platform. Nsp command-line tool. <https://github.com/nodesecurity/nsp>. Accessed: 2017-08-28.
- [18] The Node Security Platform. Node security platform. <https://nodesecurity.io/>. Accessed: 2017-08-28.
- [19] SQLMap Project. Sqlmap. <http://sqlmap.org/>. Accessed: 2017-08-28.
- [20] Helmetjs. Helmet. <https://github.com/helmetjs/helmet>. Accessed: 2017-08-28.
- [21] CERN OAuth Authorization Service. OAuth. <http://oauth.web.cern.ch/>. Accessed: 2017-08-28.
- [22] GitLab. Gitlab@cern. <https://gitlab.cern.ch/>. Accessed: 2017-08-28.