

Integrating a Pentesting Tool for IdM Protocols in a Continuous Delivery Pipeline*

Andrea Bisegna^{1,2}, Roberto Carbone¹, and Silvio Ranise^{1,3}

¹ Security & Trust, Fondazione Bruno Kessler, Trento (Italy)

{a.bisegna, carbone, ranise}@fbk.eu

² DIBRIS, University of Genova, Genova (Italy)

³ Department of Mathematics, University of Trento, Trento (Italy)

Abstract

Identity Management (IdM) solutions are increasingly important for digital infrastructures of both enterprises and public administrations. Their security is a mandatory prerequisite for building trust in current and future digital ecosystems. IdM solutions are usually large-scale complex software systems maintained and developed by several groups of ICT professionals. Continuous Delivery (CD) pipeline is adopted to make maintenance, extension, and deployment of such solutions as efficient and repeatable as possible. For security, CD pipeline is also used as a continuous risk assessment to quickly evaluate the security impact of changes. Several tools have been developed and integrated in the CD pipeline to support this view in the so called DevSecOps approach with the notable exception of a tool for protocol pentesting and compliance against standards such as SAML 2.0, OAuth 2.0 and OpenID Connect. To fill this gap, we propose an approach to integrate Micro-Id-Gym—a tool for the automated pentesting of IdM deployments—in a CD pipeline. We report our experience in doing this and discuss the advantages of using the tool in the context of a joint effort with Poligrafico e Zecca dello Stato Italiano to build a digital identity infrastructure.

1 Introduction

Identity Management (IdM) implementations exchange authentication assertions and consist of a series of messages in a preset sequence designed to protect information as it travels through networks or between servers. By using third-party authentication, IdM protocols eliminate the necessity of storing authentication information within the services for which they are used, providing a solution that helps private and public organizations prevent the misuse or abuse of login credentials and reduce the risk of data breaches.

IdM protocol standards—including the Security Assertion Markup Language 2.0 (hereafter SAML) [1], OpenID Connect (OIDC) [2], and OAuth 2.0 (OAuth) [3]—handle user requests for access to services and deliver responses based on the information a user provides. If the authentication methods, such as a password or a biometric identifier, are correct, the protocol allows the level of access assigned to the user within the service. Existing IdM protocols support policies (such as allowing password authenticated users to only read financial data while permitting also to perform payments to those authenticated by using two authentication factors) by securing assertions and ensuring their integrity during transfer. They include standards for security to simplify access management, aid in compliance, and create a uniform user experience.

*This work was partially funded by the Horizon 2020 project “Strategic Programs for Advanced Research and Technology in Europe”(SPARTA), grant agreement No. 830892, and by the Italian National Mint and Printing House (Istituto Poligrafico e Zecca dello Stato).

For instance, both SAML and OIDC support the so-called Single Sign-On (SSO) experience whereby one set of credentials allows users to access multiple services.

To have a robust security in an IdM implementation, following the compliance with the standard is extremely important. In some case, like it happens in SPID [4], it is required to ensure compliance with both the technical and legal rules defined in the regulations.

Given the many advantages of IdM protocols standard, they have been widely adopted in many different scenarios encompassing corporate organizations (typically adopting SAML), cloud platforms (e.g., Google uses both OIDC and OAuth), and both national and international infrastructures for digital identity such as those in Europe (e.g., SPID in Italy is based on SAML and similarly the eIDAS framework for identity portability across Member States is also based on SAML). Despite being used on a large scale and for many years, the deployment of these IdM protocols has proved to be difficult and fraught with pitfalls. This is so mainly because such protocols inherit the difficulties of designing, implementing, and deploying the cryptographic mechanisms on top of which they are built. Even assuming that the design of IdM protocols is secure, implementations add complexity by specifying functional details (such as message formats and session state) while deployments include further aspects (such as programming interfaces) that are absent at the design level. These additions may bring low-level threats (such as missing checks of the content in certain message fields and vulnerabilities of functions imported from third party libraries) thereby significantly enlarging the attack surface of deployed IdM protocols. There is a long line of papers devoted to the identification of vulnerabilities and attacks in deployed IdM protocols at design, implementation, and deployment level see, e.g., [5, 6, 7, 8].

Indeed, preventing such varied attacks on large IdM solutions that use complex cryptographic mechanisms is a daunting task that requires automated assistance to meet also the strict temporal constraints of delivering software systems to production environments frequently and in a safe way by adopting the Continuous Delivery software engineering methodology. This methodology has steadily gained adoption although it is difficult to reconcile with traditional security testing and analysis techniques such as penetration testing or static and dynamic analysis that take substantial time and need expertise to interpret the results (e.g., to eliminate false positives). In this context, not only automation becomes even more important to identify vulnerabilities but also the capability to provide actionable security suggestions to software developers with little security awareness is crucial to reduce security risks to an acceptable level. The advantage of actionable security suggestions is twofold: (i) they speed up the process of fixing vulnerabilities while facilitating the creation of security awareness among developers and (ii) they allow security experts to focus on more complex security issues possibly contributing to further decrease security risks.

In this paper, we present our experience of integrating a pentesting tool for IdM protocols (called **Micro-Id-Gym** [9]) in the Continuous Delivery pipeline for deploying the Italian digital identity solution based on the electronic identity card (Carta d'Identità Elettronica 3.0) that we are collaborating to develop in the context of a joint effort with the Italian National Mint and Printing House (IPZS, Poligrafico e Zecca dello Stato Italiano). We describe how the capability of **Micro-Id-Gym** to automatically perform a battery of tests derived from the IdM protocols standards and include actionable security suggestions on how to patch them, not only allows for identifying and fixing known security problems but also helps developers better understand the negative impacts of ignoring or underestimating the security considerations included in such standards while coding or deploying IdM protocols. This, in turn, contributes to increasing the level of security awareness of developers. **Micro-Id-Gym** also have the capability to create a local faithful copy of the system and gives the possibility to perform attacks that would have

catastrophic effects when performed in the production environment (e.g., Denial of Service). Finally, we show how the tool can help identify false positives by classifying tests in two classes, namely passive and active. The former only performs checks without modifying messages while the latter also interfere with the flow of the protocols by injecting suitably crafted messages with the goal of mounting an attack. Our findings have been experimentally validated by integrating Micro-Id-Gym in the Continuous Delivery pipeline infrastructure offered by GitLab. The experiments were conducted in the context of the deployment of a new version of the Italian digital identity solution based on the electronic identity card.

Structure of the paper. In Section 2, we give an overview of the DevSecOps philosophy and tools used in DevSecOps together with a pentesting tool for IdM protocols. In Section 3, we describe the scenario and its requirements, while in Section 4 we detail the design and the implementation of the proposed solution. To evaluate the effectiveness of the solution, Section 5 reports the use of the integration in a real scenario. We conclude and give an overview of future work in Section 6.

2 Background on DevOps, DevSecOps and Pentesting

Traditionally, operations and development teams have worked independently. This separation has created an environment filled with communication and alignment problems resulting in productions delays. In response to these issues DevOps was born. DevOps’s goal is to bridge the gap between the two teams to improve communication and collaboration, create smoother processes and align strategies and objectives for faster and more efficient delivery [10]. The term DevOps originates from the union of development and operations and describes the approaches to be adopted to accelerate the processes that allow an idea to move from development to release in a production environment, where it can provide value to the user. Such approaches require frequent communication between the two teams. In DevOps, developers who typically create code in a standard development environment work closely with operations staff to speed up software creation, testing, and release [11].

Security has become an important and challenging goal in the DevOps philosophy and in the Software Development Life Cycle (SDLC). This was done with DevSecOps, an extension of DevOps whose purpose is to integrate security controls and processes into the DevOps to promote the collaboration among security, development and operations teams. The DevSecOps process requires the integration of planning, design, and release, which is typically obtained by a collaborative tool chain of technologies to facilitate cooperation, provide more secure development processes and, according to [12], allows companies to save money in case of data breaches due to the effectiveness of an organization’s incident response and containment processes. Moreover, adopting DevSecOps practices for automatic building deployment eliminates a large number of manual steps that could introduce many opportunities to make mistakes.

2.1 Security practices in DevSecOps

As reported in Table 1 several state-of-the-art security practices can be used in DevSecOps to ensure that automation and security are handled continuously throughout the SDLC [13]. In this section we provide more details about the Dynamic Application Security Testing (DAST) practices since the tool we propose belongs to this cluster of practice even if in the industrial

context,¹ there are different classifications of the security practices in DevSecOps and according to them the tool we integrated is classified as a Pentesting Tool.

In general DAST practice uses a black-box security testing method that examines an application while it is running, by detecting common vulnerabilities [14]. The performed tests can detect flaws during the authentication or authorization process performing client-side, inappropriate command execution, SQL injection, erroneous information disclosure, interfaces, and API endpoints attacks but also perform penetration testing and vulnerability scanning [15]. A list of DAST tools is provided by OWASP.² Two of the most well-known tools in this context are OWASP Zed Attack Proxy³(ZAP) and Burp Suite Professional Commercial Edition⁴(Burp PRO). They observe the HTTP traffic of the application in use and create alerts on any vulnerabilities they can detect through regular usage and attempt a variety of attacks by replaying modified HTTP traffic. These tools have been designed mainly to support manual testing, but they also provide API's that could allow integration into a DevSecOps pipeline. A good example on how to integrate DAST into DevSecOps is reported in GitHub⁵ where both ZAP and Burp Pro are ready to be added in the pipeline to conduct an active real time vulnerability scan and return a security scan report as result.

The limit of the available DAST tools when deploying IdM solutions is that they are covering only partially the possible security tests needed to produce a security assessment. In fact, there is no state-of-the-art DAST tool which verifies the security aspects in IdM implementation in

Table 1: State-of-the-art security practices in DevSecOps.

Security Practice	Description
Security Practice Availability Threat Modeling	Process that defines, classifies, and analyses potential threats, assessing their risk and the appropriate countermeasure during the plan phase.
Pre-commit Hooks	Check whether a code contains strings that match specified patterns to help not to leak credentials.
IDE Plugins	Automatically perform code analysis as the developers open, edit, and save file in the IDE to get early warning of vulnerabilities.
Dependency Analysis	Technique used to detect vulnerabilities contained within a project's dependencies.
Unit Testing	Process of software testing where individual units/components of a software are tested.
Static Application Security Testing	Testing methodology that analyses source code to find security vulnerabilities that make an application susceptible to attacks.
Dynamic Application Security Testing	Type of black-box security test that scans web applications for vulnerabilities. It works by simulating external attacks on an application while it is running.
Infrastructure As Code	Instead of configuring the hardware physically, it is managed through the definition of files.
Secrets Management Configuration Management	Tools and methods for managing sensitive parts of an IT ecosystem. Control of the configuration for an information system with the goal of enabling security and managing risk.
Version Control	Practice for tracking and managing changes to software code.
Container Security Scanning	Practice for securing containers.

¹<https://dzone.com/articles/shifting-left-devsecops>

²https://owasp.org/www-community/Vulnerability_Scanning_Tools

³https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

⁴<https://portswigger.net/burp/pro>

⁵<https://github.com/jacksingleton/dast-pipeline>

terms of compliance with the standards and detect vulnerabilities coming from scientific papers.

2.2 Overview of Micro-Id-Gym

Micro-Id-Gym [9] is a tool which assists system administrators and testers in the pentesting of IdM protocol implementations. More precisely, Micro-Id-Gym considers web protocols where a Service Provider (SP) relies on a trusted third-party, called Identity Provider (IdP), for user authentication. SAML, OIDC and OAuth are three of the most known standardized protocols providing this authentication pattern (modulo different names used to refer to the aforementioned entities).

Micro-Id-Gym supports two main activities: pentesting of IdM protocol implementations and creating sandboxes with an IdM protocol deployment. The former consists of tools with a GUI to support pentesting activities on the System Under Test (SUT), namely a Proxy, a set of Pentesting Tools, and two tools called MSC Drawer and MSC STIX Visualizer. The latter can be carried out by recreating locally a sandbox of an IdM protocol implementation and it can be done by uploading the proprietary implementation or by composing a new one choosing the instances provided by the tool. The capability of creating a local copy of the SUT allows for performing pentesting activities that may cause severe disruptions such as Denial of Server (DoS) attacks.

All the exchanged HTTP messages intercepted by the Proxy, during the authentication process performed on the SUT, allows the Pentesting Tools to execute the available automated tests. The tool verifies whether the SUT suffers from the vulnerabilities tested automatically by the tool and provides details of the discovered vulnerabilities. In addition, the MSC Drawer automatically creates a Message Sequence Chart of the authentication flow by using the information collected by the Proxy. Thus the pentester can recognize at a glance whether the SUT follows the expected flow or not.

The results for each executed test are reported in a box inside the tool where the user finds a recap with *(i)* the check performed together with a brief description, *(ii)* the status of the executed test (Successful or Failed) and, in case of failure, *(iii)* the portion of HTTP message that is not compliant with the standard together with *(iv)* different suggestions to mitigate the identified flaws.

Micro-Id-Gym performs tests on both the IdP and SP in automatic manner and it supports the following test categories in the IdM protocol deployment: *(i)* performing general security web checks on any collected HTTP message, not strictly related to the protocol implementations but more in general related to web security, *(ii)* verifying the compliance with a given standard in terms of format of the messages, and mandatory fields, and *(iii)* mounting specific attacks to spot any false positives among the vulnerabilities reported by previous tests.

The tests executed by the tool can be passive or active. Passive tests analyze the traffic generated during the authentication flow to discover compliance issues such as missing parameters required by one of the supported standards, namely SAML, OAuth and OIDC. Instead active tests modify the exchanged messages to verify how the SUT reacts to the malicious messages injected by an attacker [9]. Since passive tests perform a static analysis, they can be executed all at once on the traffic collected when the authentication process is completed. This is not the case of active tests as each of them runs independently because it performs some modifications to the messages related for that test. Performing multiple modifications simultaneously would make it more complex to identify the root-cause of the failed test. For instance, an active test consists of checking if the `relaystate` parameter used to prevent CSRF attacks can be tampered with during the authentication flow. In case the authentication process is completed

despite the modification, it means the SUT does not manage correctly the `relaystate` parameter and it might expose the user to CSRF attacks [16]. In case the SUT notices the change, the test fails meaning that the SUT performs adequate verification on the `relaystate` parameter.

Currently Micro-Id-Gym supports three standards, SAML, OIDC and OAuth. As regards SAML a comprehensive list of all the automated tests are reported in Table 2, where, for each test, we specify: (i) a name to identify the security test (e.g., Session Replay), (ii) the target of the test (SP or IdP), (iii) the type of test Passive (P) or Active (A), (iv) the description of the security test, and (v) the description of the mitigation.

In order to use the Pentesting Tools provided by Micro-Id-Gym, the pentester needs to install in his device the tool which leverages the Proxy, Burp Community Edition,⁶ because the interactions with the GUI of the tool is required. The pentester is also required to provide an authentication trace which contains a list of user actions to interact with the browser used to complete the authentication process in the SUT. The authentication trace (e.g., visit www.example.com, click on *Login* button, etc.) can be easily recorded by the pentester and the Pentesting Tools verify the correctness (i.e., the correct execution) before running the automated passive and active tests.

3 Scenario and Requirements

IdM solutions are increasingly important for digital infrastructures of both enterprises and public administrations and they are a pre-requisite for building trust in current and future digital ecosystems. Unfortunately, their secure deployment is a non-trivial activity that requires a good level of security awareness [5].

For the sake of concreteness we now describe the Italian digital identity infrastructure based on the national electronic identity card, which we have deeply studied as part of a collaboration with IPZS. The scenario given by IPZS consists of an IdP based on SAML SSO protocol and (a stub) SP required to complete the authentication process. IPZS is adopting DevOps in the development stage: after each commit made by developers in the repository, an automatic build is created and the deployment in the SUT is performed. In this scenario, so far, the pentesting activities are performed manually and outside of the DevOps pipeline by running Micro-Id-Gym under the responsibility of the Security Team (ST). From the IPZS perspective this flow looked too cumbersome and error prone due to the steps required by the pentesting activities and so, in order to make the process shorter and faster, we introduced DevSecOps. We decided then to integrate Micro-Id-Gym in the SDLC.

Therefore, to reduce the redundant tasks, to make builds less error-prone and deployments more secure, we designed and implemented a solution joining the main advantages of Continuous Integration and Continuous Delivery (CI/CD) and integrate Micro-Id-Gym as a known and valid tool for the pentesting activities required by IPZS. During the process of integration we encountered some issues due to the constraints given by the tool itself and the solution adopted is described in Section 4.

From the aforementioned scenario, we identify two challenges (C1 and C2) and from them we derive some functional and security requirements.

[C1] Automated assistance. To enable the deployment automation that leads to repeatable and reliable deployments across the SDLC. Indeed, the process to perform a deployment of a SAML SSO implementation contains many critical steps like the federation process. Moreover the automatic pentesting provides a set of pentesting tools for the automatic security analysis

⁶<https://portswigger.net/burp>

of the IdM protocols by identifying security issues and helps to maintain compliance with the standard in order to eliminate basic security issues and allow the security experts to focus more on complex security issues. From C1, we derive the following two requirements:

- R1 integration in the SDLC: to automatically perform pentesting on a solution based on IdM protocols which involves several entities that interact with each other based on complex cryptographic mechanisms.
- R2 false positives elimination: thanks to the automation and the capabilities of the pentesting tool, it is possible to perform passive tests that identify vulnerabilities and through the active tests to mount attacks which help eliminating any false positives detected by the passive tests.

[C2] *Increasing security awareness.* To enable the secure deployment of IdM protocols which is a complex and error prone activity that requires a high level of security awareness in several and heterogeneous aspects. Indeed, developers get bogged down in the myriads of security practices that they are required to tame when trying to deploy or understand IdM solutions. There are plenty security indications for SAML, OAuth and OIDC spread in different sources and most of them are not easy to understand for a developer with limited security skills. The Pentesting Tools identify vulnerabilities in the implementation but also provide security mitigations with the aim to increase the security awareness of the developers. From C2, we obtain the following requirements:

- R3 compliance with the standards: faithfully adopting the best practices in order to achieve security by checking the compliance with a given standard in terms of format of the messages, and mandatory fields. Following the suggestions indicated in the standard increases the security of the protocol implementation.
- R4 collaboration support: the cooperation among developers is critical especially when senior developers are helping less skilled developers to fix vulnerabilities reported in the security issue message provided by the ST and containing actionable suggestions.

We also identify two requirements that are common to both challenges:

- R5 mitigation: provide to the developers a comprehensive and actionable analysis of the options to mitigate the discovered vulnerabilities. The details of the mitigations provided by the tool will improve the security awareness among the developers.
- R6 notification: ST and developers should have access to the test report with different levels of details in order to simplify the problems of mitigations. The test report contains different information according to the audience; for developer more aspects related to the implementation and mitigations while for security specialist only aspect related to security.

4 Continuous Delivery Solution for Pentesting of IdM Protocols

4.1 Design

As depicted in Fig. 1, the proposed solution is composed of two main components. The former, located in the bottom of the figure, is in charge to handle the repository with the source code

of the entities (IdP and SP) and the CI/CD System—indeed to satisfy the R1—and which was the starting point of a classic CI.

The latter—our contribution identified on the top part of the figure—aims to perform the pentesting on the IdM deployment while increasing the security awareness among the developers. Moreover the red arrows in the figure indicate the operations which are automatically executed, while the black dashed arrows indicate that the operations require a human intervention.

When the developer pushes new code in the repository, a notification will be sent to the ST and the build and deployment phases will start.

The source code of the SP and IdP will be built, federated and both deployed in the SUT. The sent notification notifies the ST that the automatic penetration testing on the SUT has been executed and test reports are available. The penetration testing will check the compliance with the standard (it complies with R3) by executing passive tests, and mount specific attacks by executing active tests to spot false positives vulnerabilities identified by the passive tests (fulfill R2).

The notification also contains some information needed to execute the Pentesting Tools and to allow the communication between the repository, the CI/CD system, the SUT and the Pentesting Tools. Whether the ST decides to perform pentesting on the deployed solution, the Pentesting Tools will automatically create security feedback and a test report with all the discovered vulnerabilities in a tool for team collaboration (it complies with R4) and accessible by both developers and ST (fulfill R6). The tool for the team collaboration will be used by the developers to retrieve information about mitigations and by the ST to help the developers on the fixes. This stage is thus helpful to increase security awareness on the developers. The test report contains different levels of content accordingly to the role played in the SDLC. The developers will have access to only a brief recap of the status of the vulnerabilities and mitigations and indeed satisfies R5 while ST all the details about mitigations.

4.2 Implementation

4.2.1 CI/CD System

We adopted GitLab CI/CD⁷ which is a tool built into GitLab for software development to support CI and CD. At the repository’s root in the GitLab CI/CD there is a `configuration` file used to create a CD pipeline which executes jobs contained in stages. We implemented the three operations marked with red font in Fig. 1 by interpreting three stages in the pipeline: (i) Send Notifications, (ii) Build, and (iii) Deploy. The first stage is in charge to notify the ST about the changes in the repository. We decided to send an email which contains the authentication trace and parameter required by GitLab to create Issues namely `Project Id` and `Host URL`. The second stage is responsible to build the source code, the third one will setup the webserver and deploy the solution in the SUT. These jobs get executed by the GitLab Runner agent which is an open-source application written in Go that works with GitLab CI/CD to run jobs in a pipeline. GitLab Runner is triggered by every push to the central repository if a `configuration` file is available (unless explicitly configured not to).

4.2.2 Integration of Micro-Id-Gym

Given the interoperability of Micro-Id-Gym with any operative system and its capability to perform compliance checks, we decided to integrate it in the solution and use to perform the

⁷<https://www.gitlab.com>

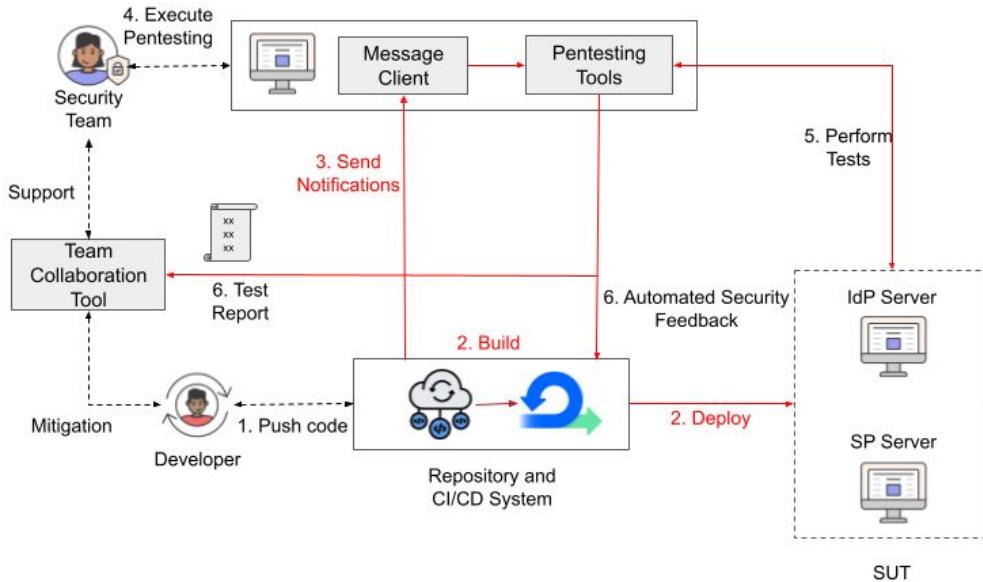


Figure 1: High Level Architecture of the proposed solution.

pentesting activities in the SUT. The integration of the tool itself is immediate also considering its flexibility to adapt in any scenario and it needs no software development nor particular skills.

The tool creates automatically GitLab issues in the repository of those tests where vulnerabilities have been discovered. In order to grant Micro-Id-Gym to read and write access to the GitLab Repository, a [GitLab token](#), [Project Id](#) and [Host URL](#) are required. For the former, which is a personal access token used to authenticate with the GitLab API, is required the user to retrieve it from his GitLab profile.

Micro-Id-Gym will also generate a report of the test results which will be automatically added a Slack⁸ channel, a collaboration team tool, accessible by the developers and ST.

5 Use Case: SAML SSO implementation

As anticipated in Section 3, we had to assess an IdP based on SAML and its compliance with SPID⁹. The implementation is given by IPZS in a context of a joint-lab and it is the Italian digital identity infrastructure based on the national electronic identity card. In this scenario we have to assess whether the SAML authentication process follows the rules defined by the standard protocol [1] to avoid security issues and lack of compliance with the technical rules provided by SPID. The scenario consists of the IdP provided by IPZS and a stub SP required to complete the authentication process.

To assess the IPZS provided scenario, we used the solution reported in Section 4 excluding the part related to the team collaboration tool because it will be a future work.

The content of the email sent to the ST is depicted in Fig. 3 and contains all the details to use Micro-Id-Gym. In detail, it provides: (i) name of the developer who changes the repository

⁸<https://slack.com/>

⁹<https://www.spid.gov.it/>

Table 2: Collection of security tests targeting SAML implementation.

Security Test	Target	Type	Description	Mitigation
Session replay	SP	A	Check whether the SP does not manage properly the session during the authentication process allowing CSRF attacks	Implement mechanisms to handle properly the user sessions
Session hijacking	SP	A	Check whether the SP does not manage properly the session during the authentication process allowing CSRF attacks	Implement mechanisms to handle properly the user sessions
<code>Relaystate</code> parameter tampering	SP, IdP	A	Check if the value of <code>Relaystate</code> parameter can be tampered	Sanitize the value of <code>Relaystate</code> parameter
Presence of ID attribute	SP, IdP	P	Check whether the ID element is present in the SAML request	Configure the SP to include ID attribute in the SAML request and the IdP to accept only SAMLRequest with ID attribute
Presence of <code>Issuer</code> attribute	SP, IdP	P	Check whether the <code>Issuer</code> element is present in the SAML request	Configure the SP to include <code>Issuer</code> attribute in the SAML request and the IdP to accept only SAMLRequest with <code>Issuer</code> attribute
SAML request compliant with SPID	SP, IdP	P	Check whether the SAML request is compliant with SPID standard	Configure the SP to issue SPID compliant SAML request
Presence of <code>Audience</code> element	SP, IdP	P	Check whether the <code>Audience</code> element is present in the SAML response	Configure the IdP to include <code>Audience</code> element in the SAML response and the SP to accept only SAML response with <code>Audience</code> element
Presence of <code>OneTimeUse</code> attribute	SP, IdP	P	Check whether the <code>OneTimeUse</code> attribute is present in the SAML response	Configure the IdP to include <code>OneTimeUse</code> attribute in the SAML response and the SP to accept only SAML response with <code>OneTimeUse</code> attribute
Presence of <code>NotOnOrAfter</code> attribute	SP, IdP	P	Check whether the <code>NotOnOrAfter</code> attribute is present in the SAML response	Configure the IdP to include <code>NotOnOrAfter</code> attribute in the SAML response and the SP to accept only SAML response with <code>NotOnOrAfter</code> attribute
Presence of <code>InResponseTo</code> attribute	SP, IdP	P	Check whether the <code>InResponseTo</code> attribute is present in the SAML response	Configure the IdP to include <code>InResponseTo</code> attribute in the SAML response and the SP to accept only SAML response with <code>InResponseTo</code> attribute
Presence of <code>Recipient</code> attribute in <code>SubjectConfirmationData</code> element	SP, IdP	P	Check whether the <code>Recipient</code> is present in the SAML response	Configure the IdP to include <code>Recipient</code> attribute in <code>SubjectConfirmationData</code> element of the SAML response and the SP to accept only SAML response with <code>Recipient</code> attribute
Canonicalization algorithm	SP, IdP	P, A	Check if the Canonicalization algorithm used by the XML parser encodes also comments	Configure IdP and SP to use XML parser Canonicalization algorithm that includes comments
SAML response compliant with SPID	SP, IdP	P	Check whether the SAML response is compliant with SPID standard	Configure the IdP to issue SPID compliant SAML response

 $P = \text{Passive Test}; A = \text{Active Test}.$

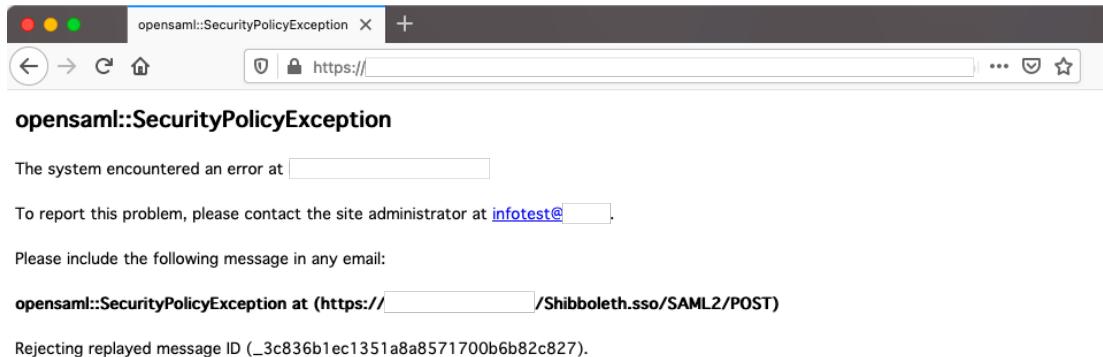


Figure 2: IdP response after a replay attack.

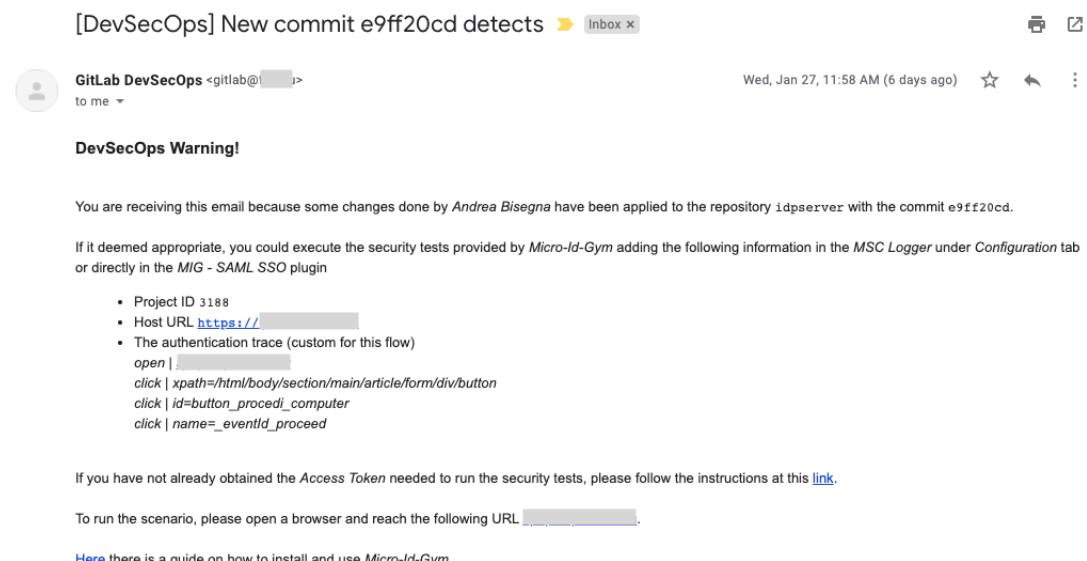


Figure 3: Email sent to the ST.

and commit number, (ii) instructions to use and install Micro-Id-Gym (iii) GitLab Project Id, (iv) GitLab Host URL, (v) authentication trace composed by a list of user actions to perform the authentication process on the SUT by a test user, (vi) instructions to retrieve a GitLab Token and (vii) URL of the SUT.

5.1 Results

The results after running the passive tests of Micro-Id-Gym show the detection of vulnerabilities both on the IdP and a stub SP. The report provided by Micro-Id-Gym is depicted in Fig. 4 showing the GUI of Micro-Id-Gym after executing the tests and where the sensitive information

like URL of the SUT and the value of the `GitLab token` are hidden for privacy and security reasons. The top part of the figure contains the information required by the tool and the authentication trace, the browser used and the information needed by GitLab (`GitLab token`, `Project Id` and `Host URL`). In the middle of Fig. 4 it is depicted the set of tests available in the tool while on the bottom part the details of the results are reported.

Even if the SP is out of scope for this assessment because the goal of IPZS was to assess only the IdP, we highlight that Micro-Id-Gym detects a missing attribute in the `SAMLRequest` namely `ProviderName`. The tool reported it as a warning issue (orange colour) because the identified attribute is optional as reported in the SAML Core [1] even if it is a good practice to include it. The attribute might be used for some other reasons by the SAML authorities.

Concerning the IdP, Micro-Id-Gym identified two issues by executing the passive tests:

- The detected canonicalization algorithm used in the `SAMLResponse` is unsecure [8] as reported in Fig. 5.
- The `OneTimeUse` element in the `SAMLResponse` is missing even if it is reported as Optional element [1].

The first vulnerability identified is due to not using a safe canonicalization algorithm which may lead to an impersonation attack and can allow an attacker with authenticated access to authenticate himself as a different user without knowing the victim user's credentials. A possible remediation, suggested by Micro-Id-Gym, could be the use of a canonicalization algorithm¹⁰ which does not omit comments during canonicalization. This algorithm would cause comments added by an attacker to invalidate the signature. For the second vulnerability, the element `OneTimeUse` specifies that the assertion should be used immediately and must not be retained for future use [17] and the mitigation provided by Micro-Id-Gym is to add the attribute in the assertion. If the element of `OneTimeUse` was added in the `SAMLResponse`, the replay attacks of `SAMLResponse` could be avoided.

To eliminate false positives and to check whether with the discovered vulnerabilities it is possible to mount attacks, Micro-Id-Gym provides an advanced analysis by running the active tests. One of the active tests, which in this scenario failed, verifies the correct interpretation of the SAML messages by the XML parser. This test adds arbitrary string as comments inside the `SAMLRequest` and check whether the IdP XML parser interprets the SAML messages without any issues and possibility to hack the messages and exploiting by an impersonation attack.

Another provided active test performs a replay attack by sending in a new session a `SAMLResponse` previously obtained. We ran these active tests provided by Micro-Id-Gym. According to the results of Micro-Id-Gym, reported in Fig. 2, all the active tests failed, meaning that the issues detected by the passive tests were false positives. We were curious to understand how the system is protected against these issues, and we thus performed a manual in-depth analysis. We found out that the technology used by the IdP includes countermeasures like cookies and time-based authenticators to prevent replay attacks. Also we noticed that the `SAMLRequest` issued are stored in the IdP to guarantee one-time use and are not valid after they have been verified.

Finally, according to the results (both active and passive), we conclude that the potential vulnerabilities identified by the passive tests are considered false positives due to the results given by the active tests.

The results pointed out the importance to include Micro-Id-Gym in the DevSecOps pipeline. Since it is well-known that most of the time during the pentesting activities is spent to eliminate

¹⁰<http://www.w3.org/2001/10/xml-exc-c14n#WithComments>

The screenshot shows the Burp Suite interface with the following details:

- Passive checks on Request:**
 - ID attribute (checked)
 - ProviderName (checked)
 - Issuer (checked)
 - Signature (checked)
 - Service Provider reference (checked)
 - InResponseTo (checked)
 - Canonicalization form (checked)
 - Audience (checked)
 - Recipient (checked)
 - OneTimeUse (checked)
 - NotOnOrAfter (checked)
- SPID Request:**
 - SPID AuthnRequest compliance (checked)
 - ID
 - Version
 - IssueInstant
 - Destination
 - Issuer
 - NameQualifier
 - Format (Issuer)
 - NameIDPolicy
 - Format (NameIDPolicy)
 - RequestedAuthnContext
 - AuthnContextClassRef
 - Signature
 - AssertionConsumerServiceIndex
- SPID Response:**
 - SPID AuthnResponse compliance (checked)
 - ID
 - Destination
 - Status
 - StatusCode
 - Format
 - InResponseTo
 - IssueInstant
 - Issuer
- Active tests:**
 - Session protection (checked)
 - RelayState protection (checked)
 - Test successful (green)
 - Warning (yellow)
 - Test failed (red)
- Table of Test Results:**

Result	Request / Response	Check	Comment
Successful	Request	ID attribute found	ID attribute found
Warning	Request	Probably missing ProviderName	Could not find ProviderName attribute
Successful	Request	Issuer found	Issuer attribute found
Successful	Request	Signature found	Signature element found
Successful	Request	SPID AuthnRequest compliance	Request follows the SPID standard
Successful	Request	SPID AuthnRequest compliance - ID	ID attribute found
Successful	Request	SPID AuthnRequest compliance - Ver...	Version attribute found and set to the...
Successful	Request	SPID AuthnRequest compliance - Des...	Destination attribute found
Successful	Request	SPID AuthnRequest compliance - Iss...	IssueInstant attribute found
- Messages to display:**
 - All tests (radio button selected)
 - Failed tests and warnings
 - Only failed tests
- Buttons:**
 - Run selected passive checks
 - Select all passive checks
 - Print request
 - Print response
 - Save report as JSON
 - Send Issues on GitLab

Figure 4: Test results in Micro-Id-Gym.

```
<saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">https://...</saml2:Issuer>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha1z2"/>
  <ds:Reference URI="#_6ae8eee8c831360128495ce330fc1f7a">
    ...
  </ds:Reference>
</ds:Signature>
```

Figure 5: Usage of an unsecure Canonicalization algorithm.

false positives, the integration of Micro-Id-Gym with its features allows to reduce the time of the pentesting and thus to satisfy R2.

Thanks to the DevSecOps philosophy we were able to automatically deploy the solution and thus eliminate redundant tasks, mitigate the risk of human error during the deploy process and last but not least to automatically and continuously security assess the deployment.

6 Conclusion and Future Work

We have proposed a flexible solution for DevSecOps satisfying all the defined requirements by integrating a pentesting tool for the automatic security analysis in a DevSecOps pipeline for IdM protocols.

We have implemented the DevSecOps pipeline by integrating Micro-Id-Gym with the aim

to build, deploy and perform penetration testing on a scenario based on IdM protocols. We have provided Micro-Id-Gym for the automatic security analysis of IdM protocols which assists to easily identify security issues by performing passive and active tests and helps to maintain compliance with the standard. Finally, we have assessed an IdM protocol scenario applying the DevSecOps philosophy and performed a security analysis of IPZS scenario by running Micro-Id-Gym.

As future work, we plan to extend the proposed solution by (i) integrating a tool for teams to return the actionable security hints provided by Micro-Id-Gym to improve security awareness among developers and (ii) further elaborate the role of actionable security suggestions of cyber threat intelligence using the STIX interface to focus more on real and current vulnerabilities which help to increase security awareness instead of possible threats whose seem unlikely.

References

- [1] O. Consortium, SAML V2.0 Technical Overview, <http://wiki.oasis-open.org/security/Saml2TechOverview> (Mar. 2008).
- [2] N. Sakimura, J. Bradley, M. Jones, B. De Medeiros, C. Mortimore, OpenID Connect core 1.0, The OpenID Foundation (2014) S3.
- [3] D. Hardt, The OAuth 2.0 Authorization Framework (RFC6749), Internet Engineering Task Force (IETF).
- [4] A. P. L. Digitale, SPID - Regole Tecniche, <https://docs.italia.it/italia/spid/spid-regole-tecniche/it/stabile/index.html>.
- [5] N. Engelbertz, N. Erinola, D. Herring, J. Somorovsky, V. Mladenov, J. Schwenk, Security Analysis of eIDAS—the Cross-Country Authentication Scheme in Europe, in: 12th USENIX Workshop on Offensive Technologies (WOOT 18), 2018.
- [6] A. Armando, R. Carbone, L. Compagna, J. Cuellar, L. Tobarra, Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-Based Single Sign-On for Google Apps, in: Proceedings of the 6th ACM workshop on Formal methods in security engineering, 2008, pp. 1–10.
- [7] J. Jurreit, P. Fehrenbach, F. Kaspar, Analysis of Security Vulnerabilities in Microsoft Office 365 in Regard to SAML, *informatikJournal* (2017) 127.
- [8] T. D. Blog, Duo Finds SAML Vulnerabilities Affecting Multiple Implementations, <https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations>.
- [9] A. Bisegna, R. Carbone, G. Pellizzari, S. Ranise, Micro-Id-Gym: A Flexible Tool for Pentesting Identity Management Protocols in the Wild and in the Laboratory, in: International Workshop on Emerging Technologies for Authorization and Authentication, Springer, 2020, pp. 71–89.
- [10] C. Ebert, G. Gallardo, J. Hernantes, N. Serrano, DevOps, IEEE Software 33 (3) (2016) 94–100.
- [11] DevOps, DevOps Definition, <https://www.redhat.com/it/topics/devops>.
- [12] I. Security, P. Institute, Cost of a Data Breach Report, <https://www.ibm.com/security/data-breach>.
- [13] R. Mao, H. Zhang, Q. Dai, H. Huang, G. Rong, H. Shen, L. Chen, K. Lu, Preliminary Findings About DevSecOps from Grey Literature, in: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2020, pp. 450–457.
- [14] J. Peterson, Dynamic Application Security Testing, <https://resources.whitesourcesoftware.com/blog-whitesource/dast-dynamic-application-security-testing>.
- [15] H. Myrbakken, R. Colomo-Palacios, DevSecOps: a Multivocal Literature Review, in: International Conference on Software Process Improvement and Capability Determination, Springer, 2017, pp. 17–29.

- [16] A. Bisegna, R. Carbone, I. Martini, V. Odorizzi, G. Pellizzari, S. Ranise, Micro-Id-Gym: Identity Management Workouts with Container-Based Microservices, *Int. J. Inf. Secur. Cybercrime* 8 (1) (2019) 45–50.
- [17] F. Hirsch, R. Philpott, E. Maler, Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2. 0, Committee Draft 1.