

Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi

**Automated Security Testing for Identity Management
of Large-scale Digital Infrastructures**

by

Andrea Bisegna

Theses Series

DIBRIS-TH-2023-XX

DIBRIS, Università di Genova

Via Opera Pia, 13 16145 Genova, Italy

<https://www.dibris.unige.it/>

Università degli Studi di Genova

Dipartimento di Informatica, Bioingegneria,

Robotica ed Ingegneria dei Sistemi

Ph.D. Thesis in Computer Science and Systems Engineering

Computer Science Curriculum

**Automated Security Testing for Identity
Management
of Large-scale Digital Infrastructures**

by

Andrea Bisegna

May, 2023

**Dottorato di Ricerca in Informatica ed Ingegneria dei Sistemi
Indirizzo Informatica
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università degli Studi di Genova**

DIBRIS, Univ. di Genova
Via Opera Pia, 13
I-16145 Genova, Italy
<https://www.dibris.unige.it/>

**Ph.D. Thesis in Computer Science and Systems Engineering
Computer Science Curriculum
(S.S.D. INF/01)**

Submitted by Andrea Bisegna
DIBRIS, University of Genoa, Genoa (Italy)
Center for Cybersecurity, FBK, Trento (Italy)

Date of submission: May 2023

Title: Automated Security Testing for Identity Management of Large-scale Digital Infrastructures

Advisor: Silvio Ranise
Director, Center for Cybersecurity, FBK, Trento (Italy)
Full professor, Department of Mathematics, University of Trento, Trento (Italy)

Co-Advisor: Roberto Carbone
Head of Unit Security and Trust, Center for Cybersecurity, FBK, Trento (Italy)

Ext. Reviewers: Luca Compagna*, Jorge Cuéllar[◇]
* Dr., Senior Scientist / Research Architect
SAP Security Research, Mougins (France)
[◇] Professor, Department of Informatics and Mathematics
Universität Passau, Bavaria (Germany)

Abstract

Ensuring the security of an organization's digital assets against cyber threats is critical in today's technology-driven world. Regular security testing is one of the measures that can help assess the effectiveness of security controls, identify vulnerabilities, and strengthen the overall cybersecurity posture. Identity Management (IdM) protocols such as Security Assertion Markup Language 2.0, OpenID Connect, and OAuth 2.0 play a crucial role in protecting against identity theft, fraud, and security breaches. Also, following the Best Current Practices introduced by the standards to enhance the security of IdM protocols is essential to minimize the risk of unauthorized access, data breaches, and other security threats and to maintain compliance with regulatory requirements, and build trust with users and stakeholders. However, deploying these protocols can be challenging due to the complexity in designing, developing and implementing cryptographic mechanisms. The implementation of IdM protocols encounters three significant obstacles: fragmented security information, rapidly evolving threat environment, and the need for a controlled testing environment. Security testers must stay up-to-date with emerging threats and establish an appropriate testing infrastructure to guarantee the security and robustness of IdM implementations, while also minimizing the possibility of security incidents that could adversely affect operations. Automated security testing plays a crucial role in addressing security concerns, particularly as the intricate functional aspects of IdM solutions contribute to their complexity. It is essential to prioritize automation to bridge the cybersecurity skills gap among IT professionals.

In this thesis, we propose Micro-Id-Gym (MIG), a framework that offers (i) an easy way to configure and reproduce the IdM production environment in a sandbox, allowing hands-on experiences with potentially impactful security tests that may winder availability of services and (ii) automatic security testing of IdM implementations together with suggestions for mitigations to avoid identified vulnerabilities. MIG provides a set of security testing tools for creating, executing, and analyzing security test cases through MIG-L, a declarative test specification language. We have evalu-

ated the effectiveness of MIG by conducting experiments to assess the accuracy in supporting detection of relevant vulnerabilities in the implementation of IdM protocols. We utilized MIG to conduct security analyses across various corporate scenarios and projects, identifying vulnerabilities and responsibly disclosing them through bug bounty programs. Our findings were recognized by the providers, who awarded us both monetary compensation and public recognition. Overall, MIG can help organizations establish a robust and agile security testing strategy, supported by suitable infrastructure and testing procedures, that can ensure the security and resilience of their IdM implementations.

Contents

List of Figures	7
List of Listings	9
List of Tables	11
Glossary	12
Chapter 1 Introduction	13
1.1 Context and Motivations	13
1.2 Goals and Contributions	16
1.3 Thesis Structure	18
Chapter 2 Background	21
2.1 Security and Penetration Testing	21
2.2 Threat Model	22
2.3 Identity Management Protocols	23
2.3.1 SAML v2.0 Web Browser SSO Profile	23
2.3.2 OAuth 2.0	24
2.3.3 OpenID Connect	26
Chapter 3 A Declarative Approach to Security Testing of IdM Protocols	27

3.1	Suggesting mitigations of automated security testing of IdM protocols based on standards	27
3.2	Overview of the proposed approach	29
3.3	MIG-L: a declarative language for Security Testing	34
3.3.1	Session Example	35
3.3.2	Session Details	38
3.3.3	MIG-L Syntax	40
3.3.4	MIG-L Semantics	49
3.3.5	Machine Readable Specification of Testsuite	63
Chapter 4	Micro-Id-Gym Design	67
4.1	Overview	67
4.2	MIG Backend	68
4.3	Dashboard	69
4.4	MIG Frontend	70
4.4.1	Proxy	71
4.4.2	MIG Tool	71
4.4.3	MIG Drawer	72
4.4.4	MIG STIX Visualizer	73
4.5	MIG in DevSecOps	74
Chapter 5	Micro-Id-Gym Implementation	76
5.1	Overview	76
5.2	MIG Backend	77
5.2.1	Client and Identity Provider repositories	77
5.2.2	STIX Repositories	78
5.3	Dashboard	80
5.4	MIG Frontend	82
5.4.1	Proxy	82

5.4.2	MIG MSC Drawer	86
5.4.3	MIG STIX Visualizer	88
5.4.4	MIG Tool	90
5.5	Usage of MIG	92
5.6	MIG in DevSecOps	92
Chapter 6 Experiments		94
6.1	User Validation of MIG Drawer	94
6.1.1	Evaluation of the Effectiveness	94
6.1.2	Experimental definition and Context	95
6.1.3	Results	99
6.2	Security Testing in the Lab using MIG	103
6.2.1	OIDC for CIE deployment	104
6.2.2	OIDC for Developers Italia deployment	106
6.3	Security Testing in the Wild using MIG	108
6.3.1	OAuth for PSD2 deployment	108
6.3.2	SAML deployment	110
6.3.3	SSO-based account linking process	112
Chapter 7 Related Work		123
7.1	State of the art	123
7.2	Methodology	124
7.3	Automated tools	126
7.3.1	General purpose tools	126
7.3.2	Tools for OAuth and OpenID implementations	128
7.3.3	Tools for SAML implementations	131
7.3.4	Tools for OAuth/OIDC and SAML	132
7.3.5	Sandboxing and education	133

7.3.6 Considerations	134
Chapter 8 Conclusions and Future Work	137
Appendix A Survey Questionnaires' Content	139
Appendix B Mapping between MIG-L and Machine Readable Specification	140
Bibliography	148

List of Figures

2.1	MSC of the SAML SSO protocol [Ale13].	24
2.2	OAuth Authorization Code Flow (simplified view).	25
2.3	OIDC flow (simplified view).	26
3.1	Generic threat model.	29
3.2	The extended threat model used in our approach.	31
3.3	High level view of the architecture of our approach.	35
4.1	Overview of MIG.	68
4.2	MIG Backend.	70
4.3	MIG Frontend.	71
4.4	MIG-D components.	73
4.5	High Level Architecture of MIG in DevSecOps.	74
5.1	Details of technology used in MIG.	77
5.2	Dashboard user interface.	81
5.3	Interface to add a new filter.	82
5.4	MIG-D login page.	88
5.5	Button to look up CTI information.	88
5.6	MIG-S search result.	89
5.7	Example of an MSC draw by MIG-D.	89
5.8	Test results example.	90

5.9	MIG-T user interface.	91
6.1	OWASP ZAP visualization.	98
6.2	MIG-D visualization.	99
6.3	Number of correct and wrong answers by vulnerability and tool.	100
6.4	Number of correct and wrong answers by question.	101
6.5	Active Test results of CIE OIDC deployment.	104
6.6	Passive Test results of CIE OIDC deployment.	105
6.7	Active Test results of Developers Italia OIDC deployment.	107
6.8	Passive Test results of Developers Italia OIDC deployment.	107
6.9	Acknowledgment received by Developers Italia.	108
6.10	Entry (excerpt) from [Moh18].	116

List of Listings

3.1	Login at Dailymail - example.	36
3.2	Link accounts - Dailymail with Facebook.	37
3.3	Assert - Dailymail with Facebook.	37
3.4	Unlink accounts - Dailymail with Facebook.	38
3.5	Examples of using the <code>assert</code> action.	40
3.6	Example of a Session with <code>assert</code> action.	40
3.7	Example of a Test.	41
3.8	Example of <code>regex</code> operation.	43
3.9	Example of <code>check</code> operation on Passive Test.	43
3.10	Example of <code>validate</code> operation.	45
3.11	Example of <code>validate</code> operation with <code>regex</code>	46
3.12	An example illustrating the usage of the <code>precondition</code>	46
3.13	Example of Testsuite.	50
3.14	Passive Test creation example.	51
3.15	Example of adding <code>operations</code> in a Passive Test.	52
3.16	Example of the Proxy API used.	52
3.17	Structure of an Active Test.	54
3.18	Example of <code>session</code> operation.	56
3.19	Example of <code>operations</code> on Test.	58
3.20	Example of using the <code>decode</code> param.	61

3.21	Example of a SAML Test in JSON Format.	64
3.22	Example of an OIDC Test in JSON Format.	65
3.23	Example of an OAuth Test in JSON Format.	66
5.1	Example of the Dockerfile of a Client implementing SAML.	78
5.2	Example of STIX JSON Object.	80
5.3	Example of MSC Logger configuration.	85
5.4	Example of an API call for adding a new message to a session.	87
5.5	Deletion of all messages in the session test.	87
6.1	Test procedure for SSOLinking Account Hijack with inference procedure.	113
6.2	IdP metadata for Facebook.	118

List of Tables

5.1	Collection of Clients and IdP instances.	79
6.1	Experimental design.	96
6.2	Summary of correct and wrong answers.	99
6.3	Analysis of co-factors.	100
6.4	Answers to feedback questionnaire.	102
6.5	IdPs candidates.	117
6.6	Clients selection per IdP.	118
6.7	Overview of the Client test results per IdP.	121
6.8	Overview of the Client disclosure procedure.	122
7.1	Comparison between MIG and the other state-of-the-art tools.	135
A.1	Post-experiment survey questionnaire.	139
B.1	Mapping between MIG-L and JSON.	140

Glossary

Digital Identity A digital identity is as a set of attributes that uniquely describe a user in a specific context (e.g., a payment service).

HTTP Message An HTTP message is a message that follows the HTTP protocol. There are two types of HTTP messages: requests and responses. An HTTP request is a message sent by a Client, such as a web browser, to a server in order to request specific information. The request message typically includes a request method, such as GET or POST, a request URI (Uniform Resource Identifier), and other headers and data (such as request body). An HTTP response is a message sent by a server to a Client in response to an HTTP request. The response message typically includes a status code, such as 200 OK or 404 Not Found, a response body, and other headers.

Operation An object that represents a specific instruction definable in a Test.

Proxy An intermediate interface that intercepts and manages the flow of HTTP messages exchanged between a browser and a server. It is used to improve performance, filter content.

Proxy API A collection of interfaces and functions provided by the Proxy that are necessary for MIG-T to operate properly. It allows to interact with the Proxy to control the interception and modification of HTTP messages.

Session A list of user actions which can be seen as a UI integration test that testers use to create for web applications and which inherits the Selenium engine and its primitives.

Test An object representing a single test executable in MIG-T composed by a list of operations.

Test Suite An object containing a collection of Tests that are executable in MIG-T.

Chapter 1

Introduction

1.1 Context and Motivations

In today's rapidly evolving digital landscape, cybersecurity has become a crucial issue for organizations of all sizes and industries. With the increasing sophistication and frequency of web cyber threats, companies must ensure that their information systems and digital assets are secure from potential vulnerabilities and attacks. Effective cybersecurity measures are essential to protect against data breaches, loss of confidential information, financial damage, and reputational harm. Protecting sensitive data and preventing unauthorized access are crucial aspects that require the implementation of strong security controls. One crucial aspect of ensuring that security controls are effective is conducting regular security testing. Security testing involves assessing the effectiveness of security controls, identifying weaknesses or vulnerabilities, and providing recommendations for improvement. Maintaining a robust cybersecurity posture is critical for organizations with large-scale digital infrastructure, as it ensures operational continuity and safeguards against potential cyber threats. Regularly security testing enables organizations to proactively address potential threats and improve their overall cybersecurity posture.

Digital identity and security testing are crucial aspects of any effective management strategy in today's technology-driven world. It is essential to ensure that individuals and organizations are identified correctly, and sensitive information is kept secure. Digital identity management involves creating and managing unique identities for users, granting appropriate access levels, and monitoring their activities. Security testing, on the other hand, involves evaluating the strength and effectiveness of an organization's security measures through various assessments and analyses. These fundamental components of management help to safeguard against potential cyber threats and mitigate risks associated with digital transactions. By implementing strong digital identity and security testing protocols, individuals and organizations can protect themselves against identity theft, fraud, and other security breaches.

Multi-Party Web Applications (MPWA)[Avi16] are a common component of many digital identity systems for many online applications and plays a crucial role in building trust in current and future digital ecosystems by adopting Identity Management (IdM) protocols to secure their systems. These protocols involve three entities: the User (through a web browser), the web application (playing the role of Client), and a trusted third party known as an Identity Provider (IdP). IdM protocols refer to web applications that employ authentication assertions from IdPs to authenticate users via Single Sign-On (SSO) protocols. IdM protocol standards—including the Security Assertion Markup Language 2.0 (hereafter SAML) [Con08], OpenID Connect (OIDC) [Nat14], and OAuth 2.0 (OAuth) [Har]—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. IdM protocols 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. 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. For instance, both SAML and OIDC support SSO experience whereby one set of credentials allows users to access multiple services.

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., [Car23, Nil18, Dim19, Ale08, Jen17, Blo, Dan16, Dim19, MSP18]. Indeed, preventing such varied attacks on large IdM solutions that use complex cryptographic mechanisms is a daunting task that requires deep and automated security testing activities to mitigate security issues.

Following the compliance with the standard is extremely important for having a robust security in an IdM implementation, as it is related not only to the security policies defined in stan-

dards and certified in Best Current Practices (BCPs), but also to legal aspects when it comes to national-level infrastructure. The former refers to the implementation of BCPs [Rob10, Nil18] as a mechanism for reducing the impact of attacks on applications and verified by Security Testers (STs), whereas non-compliance with the standards defining the IdM implementations can lead to significant security problems [Yue19, Jia19, Rob20]. The latter requires to ensure compliance with both the technical and legal rules defined in the regulations for both national and international infrastructures for digital identity such as in “Sistema Pubblico di Identità Digitale” (SPID) [Dig] which is the Public Digital Identity System in Italy based on SAML [oED13], in “Entra con CIE” (CIE) [Dev23] which is the authentication scheme that allows access to digital services provided on the public administration based on the Italian electronic identity card (CIE 3.0 - Carta d’Identita’ Elettronica) [AGI] and similarly the “Electronic Identification, Authentication and Trust Services” (eIDAS) framework for identity portability across Member States (also based on SAML) [Uni14].

From the aforementioned scenario of security testing for IdM implementation based on standards, we identify three challenges (C1, C2 and C3).

[C1] Scattered security information. It is non-trivial for a Security Tester to have the skills and technical knowledge to perform all the security testing activities required to ensure proper security posture for IdM protocols. Several tools for automatic security testing exist, but they usually target specific vulnerabilities, few of them are able to spot a set of relevant vulnerabilities for IdM protocols [PS18, Pie, MMSG15] without the possibility of being easily customized or extended even if they are open source. In addition, in case a vulnerability has been detected, the burden of finding adequate mitigation measures is completely left on the Security Tester who must also collect information about the identified problem and related fixes [Dan17]. Typically, such information is distributed in several sources ranging from the official standards and related security considerations to scientific papers addressing specific novel vulnerabilities.

[C2] Fast evolving threat landscape. The fast-evolving threat landscape requires continuous updating of security testing techniques and the ability to repeat similar test suites with each configuration change in IdM implementations. This means that Security Testers must keep pace with the latest threats and be able to adapt their testing approach accordingly. It is crucial to have a robust and agile security testing strategy that can quickly identify vulnerabilities and weaknesses to ensuring the security and resilience of IdM implementations.

[C3] Controlled testing environment. Ensuring traceability, reproducibility, and business continuity requires the establishment of a suitable infrastructure for security testing that guarantees these properties. In many cases it is not possible to perform a security testing of a IdM protocols in the wild in the production environment due to several attacks with high impact like DoS or identity theft with serious legal implications. Therefore it is desirable to reproduce the production solution in a controlled environment with well-defined testing procedures and tools that enable Security Testers to easily reproduce tests and track the results. Unfortunately, the steps to reproduce in the laboratory the production environment are complicated and it is not

always possible to create the right conditions to be able to spot the same vulnerabilities as in the production environment. Moreover, it should support the integration of testing activities into the development lifecycle, allowing for early detection of vulnerabilities and defects in the IdM implementations. Such an infrastructure can help organizations ensure that their security posture is always up-to-date and in line with their business goals, while reducing the risk of security incidents that could negatively impact their operations.

1.2 Goals and Contributions

This thesis focuses on the security of IdM protocols, both in controlled environments such as in the lab, and in production deployment in the wild. Our objective is to bridge the gap between advanced research and system administrators who may benefit from it but face challenges accessing the information directly due to either lack of time or expertise. This is particularly true for Security Testers who require extensive knowledge of protocols and their implementation in digital identity scenarios. The work starts by noting that information related to vulnerabilities in IdM implementations is fragmented, lacking a complete and consolidated catalog. This makes it difficult for Security Testers to conduct thorough analyses. We aimed to tackle this inadequacy by providing a framework which returns actionable hints that would assist system administrators in deploying secure IdM instances. The framework adopts a declarative specification language to assist Security Testers in creating test cases and provide them with a fully expandable security testing tool. Finally, to showcase the efficiency and practicality of the framework, we assess its impact in few real-world scenarios and both in the wild and in the lab.

To summarize, this work provides the following main contributions:

- a comprehensive study was conducted on the state-of-the-art security tools that provide sandboxing, learning capabilities to increase security awareness such as training simulations environment and cyber threat intelligence, perform compliance analysis, and automate vulnerability detection for IdM solutions. To fulfill C1, the study includes the collection of BCPs, vulnerabilities, and attacks associated with IdM protocols with the aim of developing a test plan with detailed specifications for comprehensive testing using a declarative language, particularly for SAML and OAuth/OIDC. The test plan will be executed automatically in different use cases using a developed tool as part of a contribution to this thesis.
- the design and development of an approach for security testing of IdM protocols which takes into account attacks and BCPs introduced by IdM protocol standards. It is based on a threat model that *(i)* subsumes the web attacker capability and provides actionable hints for *(ii)* mitigating the discovered issues and *(iii)* making informed decisions for risk management. We do this by defining a declarative language namely MIG Test Specification Language (MIG-L)—indeed to satisfy the C2—for the unambiguous specification of two types of test cases: *(i)* test cases checking whether the security controls associated with the

BCPs described in standards (such as OIDC or OAuth) are correctly implemented (automated compliance testing) and *(ii)* test cases implementing attacks to exploit vulnerabilities (automated pentesting). The first type of test cases identifies recurring vulnerabilities with high risk while the second type mounts specific (known) attacks.

- the design and development of Micro-Id-Gym (MIG), a flexible and extendable framework that can assist system administrators and Security Testers in the deployment and security testing of IdM protocols instances. In the lab, MIG provides various IdM protocols instances which can federate them in a local network and thus to satisfy C3. Additionally, the tool comprises security testing tools that can automatically analyze the security of IdM deployment. The tool is designed to be easily expandable by allowing third-party and internal contributors the integrations of new IdM deployments and security testing tools. The tool's advantages lie in its capacity to report actionable suggestions that offer precise and brief details. These suggestions can be put into action to address identified vulnerabilities and guarantee compliance with the given standard. In the wild, MIG offers a collection of security testing tools aimed at carrying out automated security analysis of IdM protocols. The tools comprise of three components: *(i)* MIG-Tool (MIG-T), which runs automated tests specified in MIG-L, *(ii)* MIG-MSD Drawer (MIG-D), which provides users with an intuitive way to view and evaluate security testing results through graphical displays, and *(iii)* MIG-STIX Visualizer (MIG-S), which simplifies the process of generating and exchanging cyber threat intelligence data.
- the effectiveness of MIG by conducting an experiment to demonstrate its accuracy in assisting with the detection of relevant vulnerabilities in the implementation of IdM protocols. Moreover the application of MIG and MIG-T across various corporate scenarios and projects describing its impact and the lessons learned from its vulnerability assessment. In particular, we identified security issues *(i)* in the implementations of the CIE OIDC and of the SPID/CIE OIDC Federation deployments in the context of the Italian digital identity, *(ii)* in a PSD2 (Second Payment Services Directive) service provided by an important Italian IdP based on OAuth and, *(iii)* in several SSO-based account linking (SSOLinking in short) procedures of online services. With SSOLinking we refers to the process of connecting or linking a user's account on a Client website to their existing account on an IdP website, so that the user can use their IdP credentials to log in to the Client website. In this process we found an attack called SSOLinking Account Hijack which exploits two CSRF vulnerabilities *(i)* Authentication CSRF (known as Login CSRF) and *(ii)* CSRF on the button triggering SSOLinking. We designed the test cases in MIG-L and conducted an experimental analysis on 40 popular websites with major IdPs, finding that 19 of them (47.5%) had SSOLinking vulnerabilities (e.g., Medium, Asos, Workable). By submitting our findings to various bug bounty programs for responsible disclosure, we were recognized by the affected vendors who awarded us both monetary rewards and public recognition for our identified vulnerabilities.

1.3 Thesis Structure

The thesis is structured as follows.

Chapter 2 - Background

This chapter introduces the foundational concepts and definitions of this work. It focuses on the security and penetration testing, including threat model and IdM protocols.

Chapter 3 - A Declarative Approach to Security Testing of IdM Protocols

This chapter introduces a declarative approach for automated security testing of IdM solutions. The approach uses a domain-specific language called MIG-L to define test cases that replicate attack scenarios and check for the adoption of Best Current Practices (BCPs) from standards like OIDC or OAuth. MIG-L is interpreted by an automated security testing tool called MIG-T, which identifies vulnerabilities in IdM solutions and suggests mitigations. The chapter provides a high-level overview of the approach and then delves into the syntax and semantics of MIG-L.

Chapter 4 - Micro-Id-Gym Design

This chapter discusses the design of MIG, a tool that helps system administrators and testers in the deployment and security testing of instances of the IdM protocol. The tool automates many of the tasks involved in the setup and configuration of IdM protocol instances, making it easier for administrators to deploy and manage copies of these systems on which it is possible to perform extensive and aggressive security testing activities. It also provides a range of tools for security testing to identify and exploit vulnerabilities in IdM protocol instances, along with automated remediation of vulnerabilities. Some of its features include automated installation of IdM protocols, easy configuration of security settings, automated penetration testing, and automated vulnerability remediation. With MIG, administrators and testers can confidently deploy and manage instances of IdM protocols, knowing that their systems are secure.

Chapter 5 - Micro-Id-Gym Implementation

This chapter focuses on the implementation of MIG and its integration into a DevSecOps scenario as part of the CI/CD pipeline. It begins by introducing the tool and its features, followed by a detailed explanation of the implementation process, including configurations and settings. The chapter also discusses the integration of MIG into a DevSecOps scenario, highlighting how it can improve the security of software applications. By the end of this chapter, readers will have a clear understanding of how to integrate MIG into their own development process and the benefits it can provide in terms of security.

Chapter 6 - Experiments

This chapter assesses the effectiveness of MIG by performing an experiment to showcase its precision in aiding the identification of vulnerabilities in IdM protocol implementation. Furthermore, the chapter delves into the specific security assessments that were conducted across various corporate scenarios. Specifically, the chapter covers the security assessments across different corporate scenarios. The chapter describes security issues in the implementations of the CIE OIDC [Dev23] and SPID/CIE OIDC Federation deployments [Dev] in the context of the Italian digital identity, a PSD2 service provided by an important Italian IdP based on OAuth, the deployments based on SAML and several SSO-based account linking procedures of online services.

Chapter 7 - Related Work

In this chapter, we present the related work in different aspects of security testing tools.

Chapter 8 - Conclusions and Future Work

In this chapter, we conclude the thesis work and offer some insight into potential future directions.

The content of this thesis is based on the following peer-reviewed international conference and journal articles:

1. Andrea Bisegna, Roberto Carbone, Ivan Martini, Valentina Odorizzi, Giulio Pellizzari and Silvio Ranise. 2019. *Micro-Id-Gym: Identity Management Workouts with Container-Based Microservices*. In International Journal of Information Security and Cybercrime (IJISP), Volume 8, Issue 1. [And19]
2. Andrea Bisegna, Roberto Carbone, Mariano Ceccato, Salvatore Manfredi, Silvio Ranise, Giada Sciarretta, Alessandro Tomasi, and Emanuele Viglianisi. 2020. *6. Automated Assistance to the Security Assessment of API for Financial Services*. In Cyber-Physical Threat Intelligence for Critical Infrastructures Security: A Guide to Integrated Cyber-Physical Protection of Modern Critical Infrastructures. Now Publishers. [BCC⁺20]
3. Sergio Manuel Nóbrega Gonçalves, Alessandro Tomasi, Andrea Bisegna, Giulio Pellizzari and Silvio Ranise. 2020. *Verifiable Contracting: A Use Case for Onboarding and Contract Offering in Financial Services with eIDAS and Verifiable Credentials*. In 25th European Symposium on Research in Computer Security (DETIPS2020). [GTB⁺20]
4. Andrea Bisegna, Roberto Carbone, Giulio Pellizzari and Silvio Ranise. 2020. *Micro-Id-Gym: a Flexible Tool for Pentesting Identity Management Protocols in the Wild and in the Laboratory*. In 3rd International Workshop on Emerging Technologies for Authorization and Authentication (ETAA2020). [And20]
5. Andrea Bisegna, Roberto Carbone and Silvio Ranise. 2021. *Integrating a Pentesting Tool for IdM Protocols in a Continuous Delivery Pipeline*. In 4th International Workshop on Emerging Technologies for Authorization and Authentication (ETAA2021). [And22]

This thesis draws upon the work of several students whom I co-supervised, and incorporates material from their Bachelor's and Master's theses. In writing Chapter 3, the Bachelor's Thesis of Matteo Bitussi from the University of Trento, submitted in 2022 with the title *Declarative Specification of Pentesting Strategies for Browser-based Security Protocols: the Case Studies of SAML and OAuth/OIDC* has been considered, along with a preliminary work for Bachelor Thesis conducted by Alessandro Biasi. In Chapters 4, 5, and 7, it has been considered the Master's Thesis of Giulio Pellizzari from the University of Trento, submitted in 2020 with the title *Micro-Id-Gym: A Tool to Support Sandboxing and Automated Pentesting of Identity Management Protocols*. In Chapter 6, it has been considered the Bachelor's Thesis of Giulio Pellizzari from the University of Trento, submitted in 2018 with the title *Design and Implementation of a Tool to Detect Login Cross-Site Request Forgery in SAML SSO: G Suite Case Study*, as well as the Bachelor's Thesis of Sofia Zanrosso from the University of Trento, submitted in 2022 with the title *Enlarging the Pen-Test Coverage of SAML Single Sign-On Solutions with Cyber Threat Intelligence*.

Chapter 2

Background

This chapter presents the fundamental concepts and definitions upon which this work is founded. Specifically, it covers the differences between Security Testing and Pentesting, Threat Modeling, Digital Identity, and IdM Protocols, including SAML and OAuth/OIDC.

2.1 Security and Penetration Testing

Security Testing and Penetration Testing are two critical processes used to evaluate the security of software systems and applications. Penetration Testing aims to identify and mitigate vulnerabilities before they can be exploited, while Security Testing aims to understand the potential risks posed by a system and evaluate the effectiveness of existing security controls.

In [Mic16] Security Testing is defined as a process that aims to identify vulnerabilities and security weaknesses in a software system or application. This process is carried out by evaluating the system's security features, such as access controls, encryption, and input validation, against security requirements and industry standards. Security Testing is often performed as part of the software development life cycle (SDLC) to catch security issues before the system is deployed. The goal of security testing is to find and fix vulnerabilities before they can be exploited by attackers. Security Testing is usually performed by security experts or software testers who use automated tools, manual testing techniques, and/or a combination of both. The results of security testing are used to improve the system's security posture and to ensure that it meets the organization's security requirements.

In [Bra05] Penetration Testing, also known as “pentesting”, is defined as a process that aims to identify and exploit vulnerabilities in a software system or application. This process simulates an attack on the system by an external or internal attacker to uncover weaknesses and security risks. Penetration Testing is often performed by professional pentesters who have the skills and

knowledge necessary to identify and exploit vulnerabilities. The goal of pentesting is to provide organizations with an understanding of the potential risks posed by their systems and to help them prioritize security investments. Penetration Testing is usually performed after the system has been deployed, although it can also be performed during the SDLC to catch security issues early. The results of pentesting are used to identify and prioritize areas for improvement, to evaluate the effectiveness of existing security controls, and to develop an action plan to reduce security risks.

2.2 Threat Model

After completing the security testing, it is important to introduce threat modeling, which is a formalized process of identifying, analyzing, and mitigating security risks that might have been missed during the testing phase. It is an essential step to ensure the security of the system and protect critical assets from potential attacks. The Open Web Application Security Project (OWASP) Threat Modeling¹ documentation defines threat modeling as “the process of identifying and analyzing potential threats to the security of a system, application, or organization in order to inform decisions about security controls and mitigations”. This definition is similar to the one provided in [Sho14, Bur05]. Finally, the goal of threat modeling is to identify potential attackers, their motivations, and the methods they might use to exploit vulnerabilities, in order to prioritize the implementation of security controls.

It is also important to consider the OWASP threat modeling for several reasons:

- **Identify and prioritize risks:** OWASP threat modeling helps to identify and prioritize potential vulnerabilities and threats in a system, which allows organizations to focus their security efforts on the most critical areas.
- **Improve communication:** OWASP threat modeling provides a common framework for identifying and communicating security risks, which can improve communication between security teams, development teams, and management.
- **Early detection:** By identifying potential vulnerabilities and threats early in the development process, OWASP threat modeling can help organizations to detect and prevent security issues before they become major problems.
- **Cost-effective:** OWASP threat modeling is a cost-effective way to improve security, as it helps organizations to focus their resources on the most critical areas and avoid spending money on unnecessary security controls.
- **Industry standard:** OWASP is a well-known and respected organization in the field of information security, and their threat modeling methodology is widely accepted as an industry standard. Adopting OWASP threat modeling can help organizations to ensure that their

¹https://owasp.org/www-community/Threat_Modeling

security practices are in line with industry best practices.

- **Compliance:** in addition, it is important for organizations to ensure their systems are compliant with security protocols such as SAML, OAuth, OIDC. Many regulatory frameworks like PCI-DSS, HIPAA, and ISO 27001 require compliance with these protocols to prevent security breaches. OWASP threat modeling provides a valuable approach for organizations to identify and mitigate potential vulnerabilities and risks in their systems, which can help them meet these compliance requirements.

2.3 Identity Management Protocols

An IdM protocol allows a Client to authenticate users through a trusted third-party server called IdP. The purpose of IdM protocol is to provide a secure and efficient way for a Client to verify the identity of its users. The IdP acts as a central repository of user information and authentication credentials, and enables users to access multiple applications or services using a single set of login credentials, thereby providing Single Sign-On (SSO) functionality. This streamlines the user experience, increases security by reducing the number of passwords required, and simplifies the management of user identities. Security Assertion Markup Language v2.0 Web Browser SSO Profile (hereafter SAML) [HM05] and OAuth 2.0 (OAuth) [Har] / OpenID Connect (OIDC) [Nat14] are two of the most known protocols providing this authentication pattern despite the fact that different names may be used to refer to the aforementioned entities.

2.3.1 SAML v2.0 Web Browser SSO Profile

SAML [OAS08] involves three entities: a user agent (UA), an IdP, and a Client. UA is a web browser with which a user interacts; the user's goal is to have access to a service or a resource provided by the Client. IdP authenticates UA and issues authentication assertions that are trusted by Client - the SSO trust relationship is depicted with a handshake icon in Figure 2.1. Client uses the assertions generated by the IdP to decide on UA's entitlement to the requested service or resource.

Figure 2.1 shows a Message Sequence Chart (MSC) where each vertical line in an MSC represents an entity, and horizontal arrows represent messages from one component to another. IdM protocols are often expressed as MSC to identify any flaws. The main steps of the SAML protocol are briefly described as follows:

S1 UA asks Client to provide the resource at URI.

A1-2 Client sends UA an HTTP redirect response (status code 302) for IdP, containing an authentication request AuthReq(ID,Client), where ID is a randomly generated string uniquely

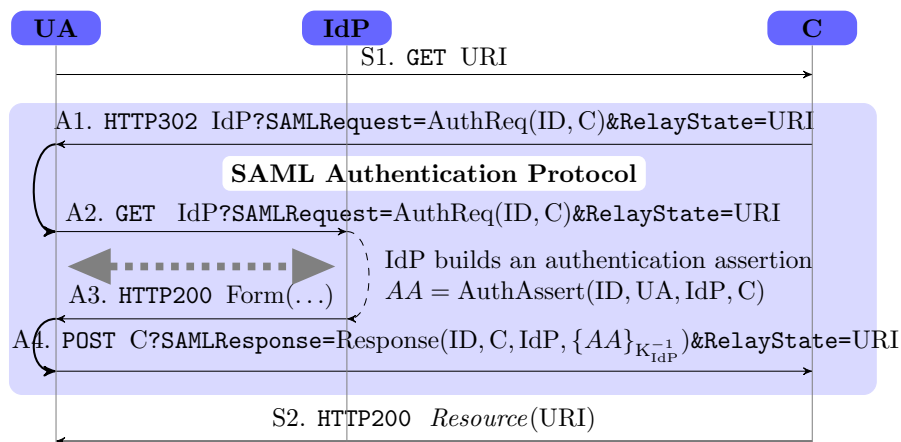


Figure 2.1: MSC of the SAML SSO protocol [Ale13].

identifying the request (steps A1 and A2). A frequent implementation choice is to use the `RelayState` field to carry the original URI that UA has requested (see [OAS08]).

↔ IdP challenges UA to provide valid credentials (dotted double arrows in the figure): this is not specified in the standard of the SAML in order to accommodate any authentication process offered by IdP.

A3-4 If the authentication succeeds, IdP builds an authentication assertion as the tuple $AA = \text{AuthAssert}(\text{ID}, \text{UA}, \text{IdP}, \text{Client})$ and embeds it in a response message $\text{Resp} = \text{Response}(\text{ID}, \text{Client}, \text{IdP}, \{AA\}_{K_{\text{IdP}}^{-1}})$ where $\{AA\}_{K_{\text{IdP}}^{-1}}$ is the assertion signed with IdP’s private key (the key icon in Figure 2.1). IdP places Resp and the value of `RelayState` received from Client into an HTML form and sends the result back to UA in an HTTP response (step A3) together with some script that automatically posts the form to Client (step A4).

S2 Finally, the Client sends UA an accepted HTTP response (status code 200) containing the requested resource.

2.3.2 OAuth 2.0

OAuth [Har] is an authorization standard allowing a user (Resource Owner, RO)—which interact with a user agent (UA), typically a web browser—is to delegate to a Client the access to his resources stored on another web server controlled by an Authorization Server (AS or IdP). The aim of the user which interacts with UA is to get access to a service provided by Client, leveraging AS as IdP. Figure 2.2 shows a MSC providing a simplified view of the OAuth Authorization Code flow. A brief description of the protocol is as follows:

- 1 Client initiates the flow by directing RO’s UA (typically a web browser) to the authorization endpoint. Client includes its identifier (`client_id`), requested scope (`scope`), local state

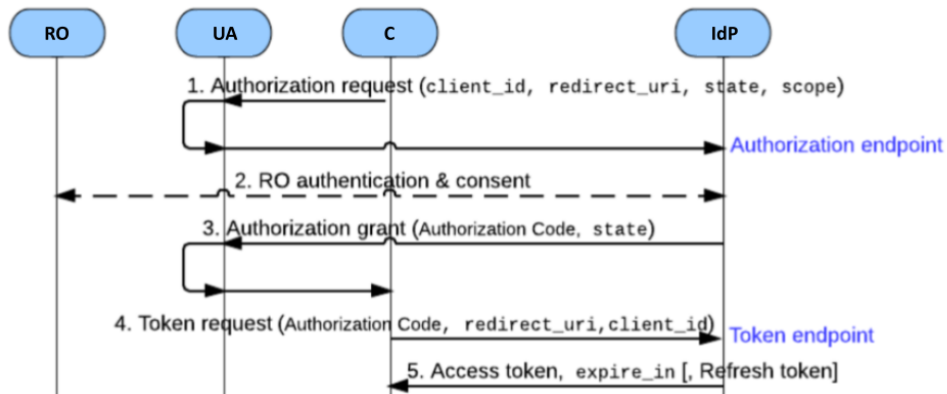


Figure 2.2: OAuth Authorization Code Flow (simplified view).

(state), and a redirection URI (`redirect_uri`) to which the AS will send UA back once access is granted (or denied).

- 2 The AS authenticates the RO (via the UA) and establishes whether the RO grants or denies the Client's access request.
- 3 Assuming RO grants access, AS redirects UA back to Client using the redirection URI provided earlier (in the request or during Client registration). The redirection URI includes an authorization code and any local state provided by Client earlier.
- 4 Client requests an access token from AS's token endpoint by including the authorization code received in the previous step. When making the request, Client authenticates with AS. Client includes the redirection URI used to obtain the authorization code for verification.
- 5 AS authenticates Client, validates the authorization code, and ensures that the redirection URI received matches the URI used to redirect Client in step 3. If valid, AS responds back with an access token and, optionally, a refresh token.

The description of OAuth abstracts away several details that are crucial for security. A trust relationship between Client and AS must be established before running the protocol by distributing appropriate meta-data between the two entities. The user must possess credentials to access AS. The configurations of auxiliary modules must be properly performed, and the format of the messages must be properly set and checked.

2.3.3 OpenID Connect

OIDC [Ope14a] is an authentication layer developed on top of the OAuth standard. The problem that has led to introduce OIDC is the fact that the OAuth protocol is often abused to implement authentication by major web/mobile native applications, while its main purpose is access delegation.

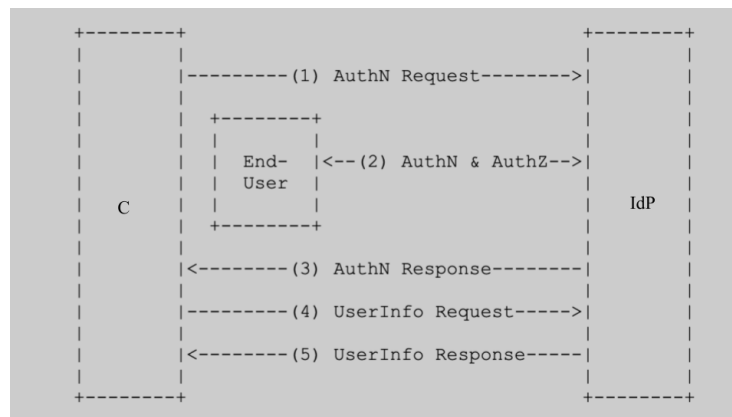


Figure 2.3: OIDC flow (simplified view).

In [Eri14], the authors studied several vulnerabilities that arise when using OAuth for authentication. Furthermore, the aforementioned study highlights the main demands that an authentication protocol should satisfy. OIDC adds two main features into the OAuth standard: the `id` token and the `userInfo` endpoint. The `id` token is a structured JSON token [Mic15] that contains information about the token issuer (the OIDC provider), the subject (user identifier) and the audience (the intended Client web/mobile native application), all signed by the OIDC provider (called also OP or IdP). This token enables Client to safely verify that the received token is issued as a result of its previous token request. While the `userInfo` endpoint is used to obtain identity-related attributes (e.g., the email and address).

OIDC protocol, in abstract, follows the following steps and illustrated in the Figure 2.3.

1. The Relying Party (RP or in general Client) sends a request to the OpenID Provider (OP or in general IdP).
2. The IdP authenticates the End-User and obtains authorization.
3. The IdP responds with an `id` token and usually an `access` token.
4. The Client can send a request with the `access` token to the `UserInfo` Endpoint.
5. The `UserInfo` Endpoint returns Claims about the End-User.

Chapter 3

A Declarative Approach to Security Testing of IdM Protocols

The purpose of this chapter is to introduce our declarative approach based on MIG-L, a (domain) language used for unambiguously defining test cases that replicate not only attack scenarios but also check the adoption of BCPs described in standards (such as OIDC or OAuth) to avoid recurring vulnerabilities with high risk for automated security testing of IdM solutions. To achieve this, we follow to identify and suggest mitigations of vulnerabilities in IdM protocol solutions. Then we provide a comprehensive explanation of the syntax and semantics of MIG-L. The semantics of MIG-L is defined by translation to a JSON formatted machine-readable language, which serves as the input language for MIG-T.

3.1 Suggesting mitigations of automated security testing of IdM protocols based on standards

Identity management is an essential aspect of online services, but it can be a complex and challenging task for developers of a Client web application. IdM protocols provide a solution to this challenge by delegating the task of managing digital identities to the IdP. Some of the benefits of delegating IdM include increased security, improved user experience, cost savings, and scalability. IdM protocols provide a secure way to manage digital identities, reducing the risk of identity theft and other cyber threats. IdPs often implement advanced security measures such as multi-factor authentication and encryption to protect user identities and prevent unauthorized access. Improved user experience is another advantage of delegating IdM. By using IdPs to manage digital identities, Clients can offer a seamless and user-friendly experience to their customers. Users can use a single set of credentials to access multiple services, eliminating the need to remember

multiple usernames and passwords. Delegating identity management can also result in cost savings. Implementing and maintaining a robust IdM system can be expensive and time-consuming for Clients. By delegating this task to IdPs, Clients can save on the costs of developing and maintaining their own IdM solution. Finally, delegating IdM offers scalability. IdM protocols offer a scalable solution to managing digital identities. As more users and services are added, IdPs can easily handle the increased demand without affecting performance or security.

Despite all these advantages, the design and implementation of IdM protocols can be prone to errors and vulnerabilities, as shown by a number of reported vulnerabilities in recent years [Car23, Nil18, Dim19]. This problem is further compounded by the high number of deployments, such as over 20% of the top 20,000 Alexa websites in the US having a vulnerable implementation of Facebook SSO [ZE14]. The complexity of these protocols can make it difficult for developers to implement them correctly and securely, and the reliance on third parties can introduce additional points of failure and potential security risks. Overall, it is important to consider these challenges when implementing IdM protocols. Trust is an important consideration when discussing IdM protocols. The use of IdM protocols can also raise concerns about privacy and data protection. The collection and sharing of personal data by trusted third parties, as well as the potential for data breaches and leaks, are all significant issues that must be considered when assessing IdM protocols. To ensure that users' personal data is protected and the risk of privacy violations is kept to a minimum, it is crucial to consider these issues and implement IdM protocols in a way that respects user privacy and protects personal data.

The information related to an IdM protocol is fragmented and spread across various sources, making it challenging for IT professionals tasked to deploy such solutions to identify a comprehensive and coherent source of information. For instance, when examining OAuth/OIDC, it is important to note that the BCPs and considerations are dispersed across various documents (e.g., [Har, Ope, IET12, Har, Ope14a, Ope14b, IET14]). Despite the acknowledged weaknesses and vulnerabilities present in IdM systems, administrators may not possess the necessary knowledge to patch these vulnerabilities, leading to a situation where the implementation remains unsecured. Furthermore, there is a lack of advancements in technology that provides support for remediation methods or evaluation plans for IdM systems. This exacerbates the issue and leaves IdM systems vulnerable to potential attacks.

Moreover, while there are numerous penetration testing tools available [Chr12, MMGS15, PS18, ZE14, BLM⁺13, PPJ22], they frequently concentrate solely on specific vulnerabilities such as a single IdM protocol or on a limited number of weaknesses. Additionally, the existing tools may not be suitable for IdM systems. This results in a lack of a centralized source of security analysis results about the IdM protocols. Furthermore, despite the existence of recognized vulnerabilities in IdM systems, administrators may not possess the knowledge to patch them, leading to an unsecure implementation. Suppose an IdM solution uses a JSON Web Token (JWT) as its authentication mechanism. A JWT consists of three parts: a header, a payload, and a signature. The signature is used to ensure the integrity of the token and to prevent tampering. We assume

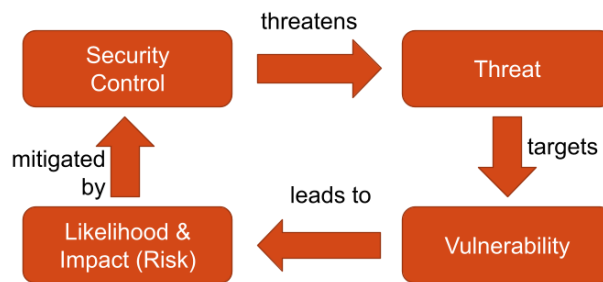


Figure 3.1: Generic threat model.

that an attacker discovers a vulnerability in the signature verification process that allows them to bypass the authentication check and gain access to a user’s account [Pau17, Jos21]. The vulnerability could be due to a weak cryptographic algorithm used to generate the signature or a flawed implementation of the signature verification process. Fixing this vulnerability would require advanced knowledge in cryptography because the solution would likely involve changing the cryptographic algorithm used to generate the signature or improving the implementation of the signature verification process. For example, the solution may require using a stronger cryptographic algorithm, such as the SHA-256 hashing algorithm, to generate the signature. This would require knowledge of cryptographic hash functions and their properties. Additionally, the solution may require implementing a more robust signature verification process, such as using digital signatures or certificate-based authentication, which would require understanding of digital signature algorithms and public key infrastructure. Doing any of this for an average IT expert is prone to be challenging.

3.2 Overview of the proposed approach

To automate the process of providing a coherent and actionable account of the heterogeneous information available on the security issues of IdM protocols, and enable the easy incorporation of future results, we propose the use of a declarative security testing tool, i.e. a tool able to execute test cases specified in a high level language capable of expressing the capabilities of standard (e.g., web) attackers or the checks for verifying the proper implementation of BCPs. The tool is based on a refinement of a well-known threat model (see e.g., [KM06]) allows for automated analysis and reporting of security issues. By using automation, we can save time and increase the accuracy of the returned results, ensuring that we are able to provide the most up-to-date and comprehensive information possible.

In [oST12], a threat model is defined as “*a systematic approach to identifying and evaluating the threats to an information system or application. The process includes identifying potential threats, identifying the vulnerabilities that could be exploited by those threats, and assessing the*

likelihood and potential impact of an exploitation". The model is reported in Figure 3.1 which includes (i) identifying the security controls to apply (e.g., such as best current practices), (ii) identifying potential attackers and their motivations, and (iii) spotting potential vulnerabilities that could be exploited. By identifying and analyzing these potential threats, appropriate (iv) security controls can be put in place to mitigate the impact. In the realm of security, a security control refers to a protective measure implemented to reduce the likelihood or impact of security risks, as defined in [oST12]. These controls can come in various forms, such as physical, administrative, or technical safeguards, and are designed to safeguard information and information systems from a range of threats. These threats can either be intentional or unintentional, and can originate from both internal and external sources. Examples of threats include hackers, viruses, natural disasters, and human errors. A vulnerability, on the other hand, is a weakness or flaw within a system or its security controls that could be exploited by a threat. Such vulnerabilities can arise in hardware, software, or the design and implementation of security controls. Examples of vulnerabilities include weak passwords, unpatched software, and misconfigured systems. Understanding the concepts of likelihood and impact is crucial for assessing and managing risk, both qualitatively and quantitatively. Various methodologies, such as OWASP, can be used to measure risk in terms of the likelihood of an event occurring and the potential impact it would have. However, cyber threats can be particularly complex and dynamic, making it challenging to stay ahead of potential risks. This is where Cyber Threat Intelligence (CTI) comes in. One of the main benefits of CTI is its ability to help organizations identify and prioritize potential risks. By analyzing data from a variety of sources, including dark web forums, social media, and other online forums, organizations can gain a better understanding of the types of threats they are likely to face and the potential impact of those threats on their business operations. Another important benefit of CTI is its ability to help organizations develop more effective risk management strategies. By using CTI to identify potential threats, organizations can develop targeted risk management plans that are tailored to their specific needs and vulnerabilities. This can help to minimize the potential impact of cyber threats and improve overall security posture. While CTI offers unprocessed data and information related to threats, MIG-S provides a consistent structure for explaining and distributing that data including their characteristics, indicators, and relationships to other threats and entities [JPD]. With this, cybersecurity professionals can organize and communicate threat intelligence in a way that is consistent and machine-readable. This allows risk evaluation to more easily consume and process the information, and make informed decisions about how to allocate resources and respond to potential threats. Risk is also a critical component of the threat modeling process which is typically expressed as a combination of the likelihood of a threat occurring, and the impact or consequences if it does.

In this study, we recognize that the threat model of Figure 3.1 needs to be adapted and extended to better address the specific needs of our approach. Therefore, we introduce a new threat model, as reported in Figure 3.2, which expands on the standard approach and in particular takes into account attacks and BCPs introduced by IdM protocol standards.

The basis of our approach is a threat model that incorporates the capabilities of web attackers

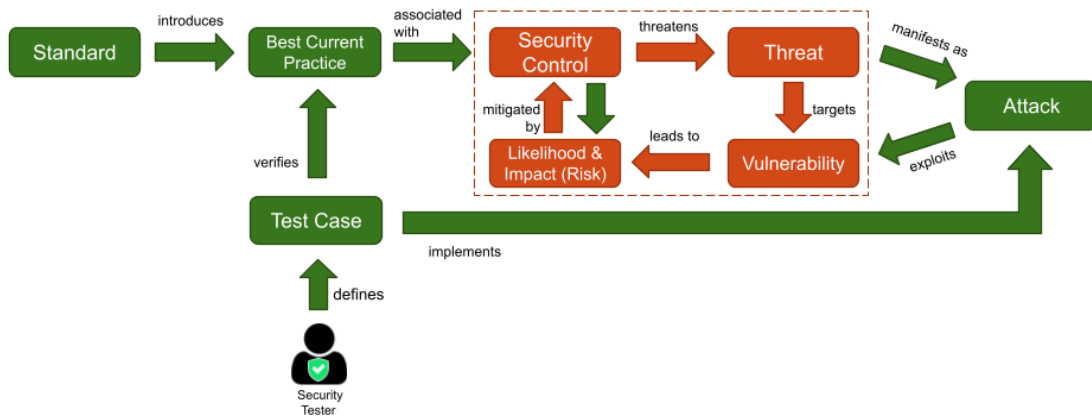


Figure 3.2: The extended threat model used in our approach.

and offers actionable guidance for mitigating discovered issues and making informed decisions regarding risk management. We accomplish this by introducing a declarative language, called MIG-L, for the unambiguous specification of two types of test cases. The first type of test cases checks the correct implementation of security controls associated with BCPs described in standards such as OIDC or OAuth, i.e. automated compliance testing. The second type implements attacks to exploit vulnerabilities, i.e. automated pentesting. By using these test cases, we can identify recurring vulnerabilities with high risk and mount specific, known attacks to assess the security posture of the system.

A BCP aims to identify an important vulnerability (with high risk) together with a structured recipe for its mitigation, i.e. Security Controls. A Security Tester is responsible for specifying test cases to assess the proper implementation of the Security Controls associated with BCPs. These test cases can include verifying the effectiveness of the controls in mitigating the identified vulnerabilities. One of the BCPs provided by the OAuth standard is the use of Proof Key for Code Exchange (PKCE) [Bra19]. The impact of not implementing PKCE can be high, as it can lead to the exposure of the authorization code and potentially result in unauthorized access to protected resources. PKCE is designed to mitigate this risk by adding an additional layer of security to the authorization flow. PKCE involves the use of a code verifier and a code challenge, which are used to generate and validate the authorization code. The code verifier is a randomly generated value that is used to create a code challenge, which is then sent to the authorization server along with the authorization request. The authorization server then compares the code challenge to the code verifier to validate the authorization code. By implementing PKCE, web applications can reduce the risk of unauthorized access to protected resources.

By playing the role of an Attacker, Security Tester can implement attacks that exploit the vulnerabilities to assess their potential impact and provide valuable insights through the Common Vulnerabilities and Exposures (CVE) into the level of risk associated with a given vulnerability, which can help inform decision-making around further mitigation strategies. An example of at-

tack is the “token replay” attack [Sah16], which involves an attacker intercepting a valid *access token* and replaying it to gain unauthorized access to protected resources. For example, suppose a user has authorized a client application to access their protected resources, and the authorization server issues an *access token* to the client application. If an attacker is able to intercept this token, they could use it to gain access to the user’s protected resources without going through the authorization flow. To mitigate this attack [Ric15], OAuth implementers can use measures such as token expiration and revocation, which ensure that *access tokens* are only valid for a limited period of time and can be revoked if they are compromised. Implementing additional security measures such as rate limiting and IP address filtering can also help prevent token replay attacks.

The proposed approach, which incorporates and extends the one in [oST20], recognizes the need for BCPs and the attacks to evolve over time and the importance of having an easy way to express and include new ones given the rapidly evolving threat landscape. To address this, our approach employs a declarative language based on MIG-L. Our security framework is based on a rigorous and formal representation of security concepts, which enables the expression of BCPs and attacks in a concise and clear manner. By leveraging this approach, our proposed security framework ensures that BCPs and attacks can easily be expressed, understood, and adapted, as the security landscape continues to evolve over time.

We can distinguish test cases specified in MIG-L w.r.t. two main different classifications namely syntactic or semantic. In the former, we can distinguish test cases according to the types of language constructs and, in the latter, to the semantics of the constructs occurring in them. The syntactic class contains two subclasses, namely Passive and Active whereas the semantics class comprises three, i.e. Web [Ste18a], Network [Ste17] and Powerful attacker [PPJ22]. We discuss each class in detail below.

One of the BCP in OAuth [Har] that requires a web attacker capability is the implementation of OAuth 2.0’s *state* parameter for preventing cross-site request forgery (CSRF) attacks [Har18]. The *state* parameter is a random value that is generated by the client and included in the *authorization request* sent to the authorization server. The authorization server then includes this same value in the response to Client, allowing it to verify that the response came from the same server that it sent the request to. This prevents an attacker from intercepting the *authorization request* and sending a fake response that could trick the client into revealing its access token. However, if an attacker can perform a successful Cross-Site Request Forgery (CSRF) attack on the client’s web application, they can still trick the client into sending an *authorization request* with a fake *state* value. To prevent this, the client should include the *state* value in a hidden form field or a cookie, in addition to including it in the *authorization request*. This way, the value cannot be easily obtained by an attacker attempting a CSRF attack.

Another OAuth BCP [Har] reports the use of the token binding protocol extension to prevent token interception attacks [Eli19] which requires a network attacker capability. Token binding is a security protocol extension that ensures that an *access token* is bound to a specific client device or user agent. This means that if an attacker intercepts the token in transit, they will not be able to

use it on another device or user agent. The protocol extension achieves this by binding the token to a cryptographic key that is unique to the client device or user agent. However, if an attacker is able to intercept network traffic between the client device or user agent and the authorization server, they could potentially intercept the token binding information and use it to generate a fake token. To prevent this, the client device or user agent should use a secure connection protocol such as Transport Layer Security (TLS) to protect the token binding information in transit.

In the OAuth standard [IET14] we also find an attack which requires a powerful attacker—with powerful attacker we assume that the eavesdropper with limitless resources can intercept all communication between parties involved in the OAuth protocol, and in addition, two of the three parties have the potential to collude and launch an attack against the third party—capability like in the “token leakage” attack. In this case, the attacker is able to compromise either the Client or the authorization server, or both, and gain access to the client’s OAuth *access token*. With this *access token*, the attacker can impersonate the Client and access protected resources on the resource server. To carry out this attack, the attacker may collude with either the Client or the authorization server. For example, the attacker could trick the Client into installing a malicious app that steals the *access token*, or they could compromise the authorization server and modify its behavior to leak an *access token* to the attacker. Once the attacker has the *access token*, they can use it to access resources on the resource server without the knowledge or consent of the Client. This can lead to the exposure of sensitive information or the unauthorized modification of data on the resource server. To prevent this type of attack, it is important to implement strong security measures, such as using HTTPS for all communications, enforcing token expiration policies, and regularly auditing the behavior of all parties involved in the OAuth protocol. Additionally, it is important to implement robust authentication and access control mechanisms on the resource server to limit the damage that can be done by an attacker with a stolen access token.

Mastering the technical aspects of BCPs and security issues in general requires a significant amount of expertise and can be a daunting task especially for IT professionals not trained in cybersecurity. To help with this, we can rely on automated support to automate the checks for BCPs and attacks, including both Passive Test (detailed and expressed in MIG-L) and Active Test that involve implementing attacks. By utilizing MIG-T, we can even receive actionable hints for mitigating any issues that are discovered during these tests. By relying on automated support, we can ensure that our security practices are being handled correctly, while also freeing up time for experts to focus on more complex issues that require their attention. Ultimately, the use of automated support is an essential tool for ensuring that our security practices are up-to-date and effective.

In the semantic class and based on the test case, we have identified two distinct capabilities related to the abilities of an attacker or Security Tester in a security testing scenario. The first capability is relevant in cases where we need to verify the BCPs. In such scenarios, we can infer that the attacker can be a network attacker [CMP17] or web attacker [CMP18], or a powerful attacker [PPJ22]. There may be BCPs that require the capability of a web attacker, while others

may require the capability of a powerful or network attacker. The specific capabilities required will depend on the particular BCPs being tested. For example, if the BCP being tested is related to redirect URI attacks in OAuth [Yan16], it may be more appropriate to assume the capabilities of a web attacker while others may require the capability of a powerful or network attacker like in the security of the OAuth token exchange process which involves the exchange of access and refresh tokens between the OAuth client and the authorization server [Rya14].

The second capability is relevant in cases where we need to implement attacks. In such situations, we can infer that the attacker has the capability of a web attacker as it happens in the authorization code interception attack [Ryu16]. In this attack, the attacker intercepts the *authorization code* returned by the authorization server to the OAuth client during the authorization flow. The attacker may do this by means of phishing attacks or exploiting vulnerabilities in the OAuth client application. Once the attacker has intercepted the *authorization code*, the attacker can use it to obtain access and refresh tokens, which can be used to access the user's protected resources. This attack can be particularly effective if the OAuth client is not properly securing the *authorization code*, or if the *authorization code* is transmitted over an insecure channel.

3.3 MIG-L: a declarative language for Security Testing

In our approach, a Security Tester tasked to develop two sets of test cases. The former to check whether the BCPs are present, while the latter to implement certain attacks. To define these test cases, we use MIG-L. This language provides a formal and concise way of specifying the test cases. We then use an automated security testing tool called MIG-T to execute these test cases. By using an automated tool, we can quickly and accurately assess the security posture of the System Under Test (SUT). Overall, our approach allows for efficient and effective security testing of software systems.

The architecture of our approach, which incorporates the extended threat model in Figure 3.2, is illustrated in Figure 3.3. In our approach, a Security Tester inputs a Session—a list of user actions which can be seen as a UI integration test that testers use to create for web applications and which inherits the Selenium¹ engine and its primitives—and Test, and receives an Output from the MIG-T. MIG-T replicates the user actions within the Session using a Browser that communicates to multiple servers, and all HTTP messages are stored through a Proxy. The main components of MIG-T are:

- The Test Handler is responsible for interpreting the Test provided as input and specified in MIG-L. The Test Handler performs the operations listed in the Test and executes it. The Test Handler also communicates with the Session Handler via the Session API and with reporting and Proxy Interaction without API.

¹<https://www.selenium.dev/>

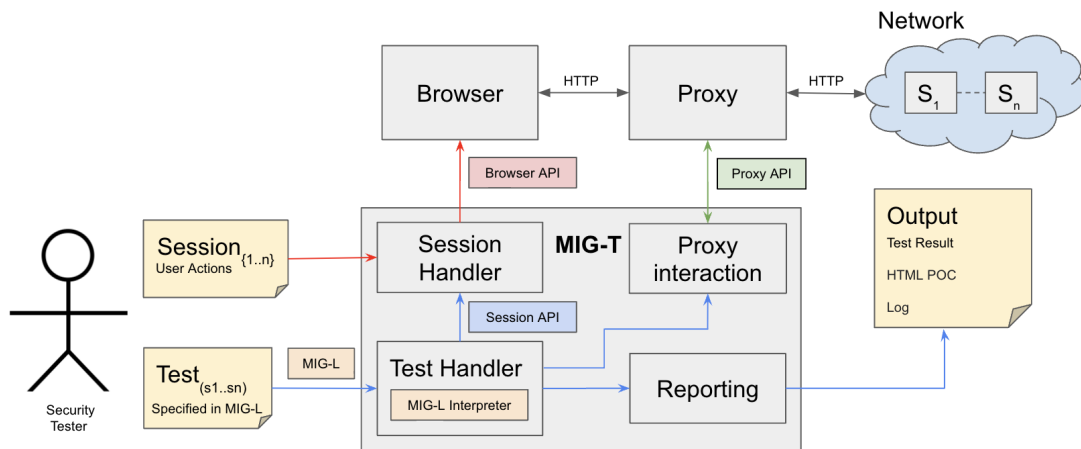


Figure 3.3: High level view of the architecture of our approach.

- The Session Handler requires the Session as input and some commands from the Test Handler. The Session Handler communicates with the browser using some browser API. This component ensures that the Test is executed in the proper environment.
- The Proxy Interaction component is responsible for communicating with the Proxy using some Proxy API. The Proxy Interaction ensures that the Test is executed under the specified conditions.
- Finally, the Reporting component provides the output of the executed test. This includes any vulnerabilities that were identified and any security controls that were properly implemented. This component is essential in helping the Security Tester and other stakeholders understand the results of the Test and take the necessary steps to improve the security of the system.

We identify three distinct APIs defined as *primitives*, which can be accessed by individual components, regardless of their underlying technology. The three APIs available in MIG-T are the Browser API, Proxy API, and Session API. The Browser API allows for the automated execution of user actions reported in a Session. For example, it provides the `driver.get(URL)` method to open a specific URL in the browser. The Proxy API is used to intercept and manipulate an HTTP Message. The `HttpRequestResponse.getRequest()` method can retrieve intercepted messages. Finally, the Session API manages Session executions. The `start(Session s1)` method creates a new thread object to run a specified Session `s1` and returns it as a result.

3.3.1 Session Example

In the following example we report a Session (`s1`) which shows the SSO-based account linking process (SSOLinking, in short) and allows users to link their Client accounts on a website to a

SSO account they own at an IdP. Using this process, users can log in on Client by leveraging popular IdPs for the authentication, while keeping their existing profiles on the Client. This ensures a SSO experience, besides the traditional form-based login. Indeed, (s1) just comprises the user actions to functionally execute with the Browser the entire SSOLinking process in an automated way at anytime so to be certain the process is working as expected. We can decompose (s1) in 4 main parts: Login at Client, Link accounts, Assert, and Unlink accounts. Let us discuss each one of them, also presenting some real examples where commands are presented in red, macros in purple, and comments in blue.

Login at Client. This comprises the user actions to login a testing user at the Client. We illustrate an example for the Dailymail² website in Listings 3.1. The Dailymail login page is opened and the cookie policy accepted. Then email and password are filled-in and the login button clicked.

```
1 open | https://www.dailymail.co.uk/registration/login.html |  
  //opens login page  
2 click | xpath=/html/.../div/button[2] | //accepts the cookie  
  policy  
3 click | xpath=/html/.../div[2]/input | //clicks on email text  
  field  
4 type | xpath=/html/.../div[2]/input | john@example.com //  
  types the email  
5 click | xpath=/html/.../div[3]/input | //clicks on password  
  text field  
6 type | xpath=/html/.../div[3]/input | 12345678 //types the  
  password  
7 click | xpath=/html/.../div[5]/button | //clicks the login  
  button
```

Listing 3.1: Login at Dailymail - example.

Link accounts. This includes the user actions to link the testing user account as authenticated at the Client with the one at the IdP. We continue the example for the Dailymail website (Client) when linking accounts with Facebook (IdP) in Listings 3.2. The user actions are identical to the ones in the previous listing and the comments describe each user action.

²<https://www.dailymail.co.uk/>

```

1 open | https://www.dailymail.co.uk/registration/profile/edit.
   html | //opens account page
2 click | xpath=/html/.../li[1]/a | //clicks on Facebook link
   account button
3 click | xpath=/html/.../button[2] | //accepts Facebook
   cookies
4 click | id=email | //clicks on email text field
5 type | id=email | john@abcd.com //types the email of user on
   Facebook) <email>
6 click | id=pass | //clicks on password text field
7 type | id=pass | abcdefgh //types the password on Facebook <
   password>
8 click | id=loginbutton | //clicks the login button
9 click | xpath=/html/.../div | //clicks to give consent to
   link accounts

```

Listing 3.2: Link accounts - Dailymail with Facebook.

Assert. Now that the linking between accounts should have been done, the Tester wants to be sure this was completed successfully. Assertions provide an easy way to set the expectations for the UI test. Listing 3.3 presents the assertions for our example on linking accounts between Dailymail (Client) and Facebook (IdP). The idea is very simple: the account profile is accessed and the fact that the linking button with Facebook is not there allows to derive that the linking was already done.

```

1 open | https://www.dailymail.co.uk/registration/profile/edit.
   html | //re-opens account page for @assertion purposes
2 assert not clickable | xpath=/html/.../div/a | //verifies the
   not availability of the linking button

```

Listing 3.3: Assert - Dailymail with Facebook.

Unlink accounts. If the test is not completed with a reset process to unlink the accounts, then the test would not be repeatable. Listing 3.4 completes our example and illustrates the user actions to unlink the user account at Dailymail from the account on Facebook.


```

1 open | https://www.dailymail.co.uk/registration/profile/edit.
   html | //re-opens account page
2 click | xpath=/html/.../div/a | //clicks unlink account
   button for Facebook
3 click | xpath=/html/.../span/a[2] | //confirm to unlink the
   accounts

```

Listing 3.4: Unlink accounts - Dailymail with Facebook.

3.3.2 Session Details

In this section, we will explain the syntax used for defining a Session. A Session is a list of user actions that a browser automatically performs to initiate the HTTP messages required for a Test, enabling the MIG-T to simulate user interactions and collect information about the web application's behavior in response. A Session can include `assert` statements which can be used to check whether a specific condition is true, such as validating that the flow is proceeding as intended. Additionally, a Session can include a variety of actions such as opening a URL, clicking a link or button, typing text into input fields, and also asserting. An example of a basic Session may involve opening a browser at a specific URL, clicking a `Login` button, inputting a username and validating with an `assert` by verifying the existence of specific HTML element. The syntax utilized in the Session is derived from the Katalon framework,³ but we have expanded upon it by adding new commands. The Katalon framework itself inherits the Selenium engine and its primitives. The Session is able to handle standard commands as dictated by the framework, but also includes additional ad-hoc commands defined by us such as `wait`, `assert`, and `clear cookies` to offer greater flexibility. It is possible to specify defined properties, referred to later as indicator, of an HTML page and CSS (which is used to define the style for a single HTML page)⁴ in commands such as `type`, `click`, and `assert` by providing them as key-value pairs.

The allowed key properties for indicator include:

- `name`: is the name of the tag used to create the HTML element.
- `class`: refers to the value assigned to the class attribute of the HTML element.
- `id`: refers to the value assigned to the id attribute of the HTML element.
- `xpath`: refers to the XPath expression used to locate an element within the HTML document.
- `link`: refers to the HTML element that creates a hyperlink to another webpage or resource.

³<https://katalon.com/>

⁴<https://www.w3.org/Style/CSS/Overview.en.html>

Details of standard commands are listed below:

- `open | http://www.website.com`: opens a web browser at the specified URL (in this example `http://www.website.com`).
- `type | indicator | text`: types the string text at the location identified by indicator.
- `click | indicator`: clicks the element on the page identified by indicator.

A list of ad-hoc commands along with their details is provided below:

- `wait t`: this delays the execution of the Session by a specified amount of time `t` milliseconds. It can be added manually in the Session to give the Security Tester time to solve a captcha or additionally automatically added by the Test to ensure that a request is properly processed.
- `clear cookies`: this results in the browser associated with the session clearing all the cookies stored within it. This can be useful in a Session for instance to test the login feature. Additionally, it can also be initiated by the Test as needed.
- `assert actions |...`: verifies whether a particular condition specified in actions is valid.

A list of assert statements are provided below, along with the details of allowed actions:

- `assert clickable | indicator`: ensures that the HTML object identified by indicator is clickable.
- `assert not clickable | indicator`: ensures that the HTML object identified by indicator is not clickable.
- `assert visible | indicator`: ensures that the HTML object identified by indicator is visible.
- `assert not visible | indicator`: ensures that the HTML object identified by indicator is not visible.
- `assert element content is | indicator | <content>`: ensures that the HTML object identified by indicator is `<content>`.
- `assert element content has | indicator | <content>`: ensures that the HTML object identified by indicator contains `<content>`.
- `assert element class is | indicator | <content>`: ensures that the class attribute of the HTML object identified by indicator is `<content>`.
- `assert element class has | indicator | <content>`: ensures that the class attribute of the HTML element identified by indicator includes the class `<content>`.
- `assert element has attribute | indicator | <content>`: ensures that one of the attributes of the HTML object identified by indicator is as specified in `<content>`.
- `assert element not has attribute | indicator | <content>`: ensures that the HTML object identified by indicator does not have the attribute specified in `<content>`.

In Listing 3.5 we report some examples of usage of assert.

```
1  assert clickable | xpath=/body/.. | //check that the element
    identified by the xpath "/body/.." is interactive
2  assert not visible | id="elem" | //check that the element
    with the id "elem" is not visible on the page
3  assert element content is | xpath=/body/div/label | text to
    match //check that the element identified by the xpath "/"
    body/div/label" has the text "text to match" as its
    content
4  assert element class has | xpath=/body/... | class_to_match
    //check that the element identified by the xpath "/body
    /..." has the class "class_to_match" in its classList
```

Listing 3.5: Examples of using the assert action.

The Listing 3.6 reports an example of a Session. This Session specifies the steps taken to log in to the website `www.example.com/login` by loading the login page, entering the email and password, clicking the login button, and verifying the presence of the logout button.

```
1  open | http://www.example.com/login // opens login page
2  type | email=username | bob@email.com // types the email
3  type | pwd=password | password // types the password
4  click | id=loginBtn // clicks on the login button
5  assert clickable | id=logoutBtn | // verifies the
    availability of the logout button
```

Listing 3.6: Example of a Session with assert action.

3.3.3 MIG-L Syntax

A declarative approach to security testing of IdM protocol solutions involves describing how the testing is performed. In the following sections, we will delve into the details of the syntax by providing a comprehensive explanation of all available instructions. In MIG-L, a Test would involve utilizing the various built-in functions and capabilities of MIG-T's three distinct APIs mentioned in Section 3.3. Using these APIs, in MIG-L, it is possible to manage and manipulate HTTP messages, handle and manipulate sessions, and integrate an oracle as a criterion against which the outcome of a test can be evaluated.

3.3.3.1 Test Suite Definition

A Test suite is an object that contains a collection of Tests. To define a `Test suite`, the following syntax should be used: `define_suite(name, description, filter)`. The instruction takes three parameters, all of which are required:

- `name` - the name to assign for the `Test suite`.
- `description` - the description to assign for the `Test suite`.
- `filter` - a boolean value parameter, which is optional and has a default value of `true`. If the value of this parameter is set to `false`, all HTTP messages will not be filtered. If the value is set to `true`, all HTTP messages will be filtered. The filter discards images, fonts, and other unwanted messages.

3.3.3.2 Test Definition

Each Test is represented as an object which contains a list of operations that are executed when the Test is run. To initialise a Test, the tag `start_test (name, description, type, s1, ..., sn, result)` should be used and the parameters are:

- `name` - the name of the Test.
- `description` - a description of the Test.
- `type` - can be either `active` or `passive`.
- `s1, s2, ...` - all the sessions involved in the Test.
- `result` - mandatory only with `active tests`. It specifies when a Test should be considered successfully accomplished.

This tag should be the first one because it initializes a new Test, and all the following instructions are considered instructions of this Test, until the end of the Test specified with the tag `end_test()`.

Listing 3.7 reports an example of a Test.

```
1 start_test(Modify the value of State parameter, This test
   tries to modify the value of the State parameter, active,
   s1, correct flow s1) // instruction to start a test
2 ... // other instructions of the test
3 end_test() // instruction to end the test
```

Listing 3.7: Example of a Test.

Result specifies the oracle of the Test, and the Security Tester can choose one of the following values according to the expected result of the Test. It could have one of the following values:

- `correct flow Session`: The Test succeeds only if all the user actions specified in `Session` are executed without errors.
- `incorrect flow Session`: The opposite of `correct flow`, the Test succeeds only if there is an error.
- `assert only`: The Test result ignores the validation of the `Session` flow but gives a result depending on the assertions defined in the `Session`. This means that if the execution of the `Session` fails, the result will not take it into account.

When `correct flow` (or `incorrect flow`) is used without specifying a `Session`, it verifies all the `Sessions` given as input. Additionally, the result of the Test may be changed whether other conditions stated in `preconditions` and `validate` operations are defined in the Test.

3.3.3.3 Operation

In the following sections, we will report on all the available operations that can be used in Tests. An operation refers to an object within a Test that represents a particular instruction. There are certain operations that can only be used in Active Tests, while others are specific to Passive Tests. A detailed list of all the operations together with the translation in machine readable format is available in Appendix B.

3.3.3.4 Message Type

With the parameter `message type`, it has to be specified on which HTTP message the given operation is applied. The following types are currently available but more could be defined:

- `request` - all requests.
- `response` - all responses.
- `OAuth request` - all the OAuth-related requests.
- `OAuth response` - all the OAuth-related responses.
- `SAML request` - all the SAML requests.
- `SAML response` - all the SAML responses.

3.3.3.5 Operation in Passive Test

The possible operations in a Passive Test are:

- regular expression - by using the instruction `regex(regexToApply, message_section, message_type)` and passing the regular expression between quotation marks "...". An example is listed in Listing 3.8. By using this operation, it is possible to apply a regular expression in a specific section of a message and type. The required parameters are:
 - `regexToApply`: a regex to be tested.
 - `message_section`: which part of the HTTP message to apply the regular expression. It assumes the values `url` or `head` or `body`.
 - `message_type`: the type of message on which the operation should be applied.
- check list - it is possible to define a list of checks that allows to analyze the intercepted messages. All the checks need to be specified between by using (i) `start_checks(message_type)` where `message_type` is the type of message on which the operation should be applied and (ii) `end_checks()`. The available checks are:
 - `check(string, action, in)` where (i) `string` is the string of the message, (ii) `action` is the operation to perform with possible values `is present` or `is not present` and (iii) `in` in which defines the part of the message to apply the check and can assume the value of `url`, `head` and `body`.
 - `check_parameter(param, in, action)` where (i) `param` is name of the message parameter, (ii) `in` defines the part of the message to apply the check and can assume the value of `url`, `head` and `body` and (iii) `action` is the operation to perform with possible values `is present`, `is not present`, `is value`, `is not value`, `contains value`, `not contains value`.

```

1  regex("code_challenge_method\\s*=(?!/s*\\ plain)", head,
    authorization_request) // regular expression to
    modify the code_challenge parameter

```

Listing 3.8: Example of regex operation.

An example of operations in Passive Test is listed in Listing 3.9.

```

1  start_checks(message_type) // starts the operation
2  check(code, is present, url) // verify the presence of
    code in URL
3  check_parameter(id, head, contains value) // verify
    whether the id in head contains $value
4  end_checks() // ends the operation

```

Listing 3.9: Example of check operation on Passive Test.

3.3.3.6 Operation in Active Test

In the following sections, we will report all the operations that can be used in Active Testing. These operations are divided into operations on the Test and operations on the Session.

3.3.3.7 Operation on Session

The possible operations in a Session in Active Test on which sessionname is the name of the Session to manage, are:

- `start(sessionname)` - starts a new session with the specified name.
- `stop(sessionname)` - stops the session with the specified name.
- `pause(sessionname)` - temporarily suspends the session with the specified name.
- `resume(sessionname)` - resumes a paused session with the specified name.
- `clear_cookies(sessionname)` - clears all cookies associated with the specified session.

The following operations are used to manipulate a Session:

- `mark(marker_name, action, sessionname)` - Instruction to mark a specific user action in a Session. The required parameter are (i) `marker_name` the name to be given to the new marker (ii) `action` the action to be marked and can assumes:
 - `last_action` to mark the last performed user action.
 - `last_open` to mark the last “open” performed user action (e.g., `open | url_example |`).
 - `last_click` to mark the last “click” performed user action (e.g., `click | link=login |`).
 - `all_assert` to mark all the assertion of the Session with the same marker.(iii) `sessionname` is the name of the Session.
- `save(variable_name, action, sessionname)` - Instruction to save in a variable some data about the user action of a Session. The required parameter are (i) `variable_name` is the name to give to the new variable, (ii) `action` the operations to be saved and can be followed by `.elem` to save only the element `_elem_` inside the user action. If `.elem` is followed by `.parent`, then only the parent div element is saved (e.g., `last_action.elem.parent`). It can assumes:
 - `last_action` to save the last performed user action.
 - `last_open` to save the last “open” performed user action.
 - `last_click` to save the last “click” performed user action.
 - `last_url` to save the the last url that was opened.

- `all_assert` to save in a variable all the user actions defined as assertion.
- `track` to save the user actions within a range where the interval limits are required (i.e. `save(variable_name, action, sessionname, [M0,M1])`).

and (iii) `sessionname` is the name of the Session.

- `add(string, marker_name)` - Instruction to add `string` as a user action in a Session at a position specified by `marker_name` (e.g., `add("type | id=login | me@email.com", marker_name)`).
- `delete(marker_name)` - Instruction to delete a user action in a Session at a position specified by a `marker_name` where `marker_name` is the name of the marker.

3.3.3.8 Operation on Test

The possible operations in a Test can be:

- **Validate Operation:** they represent the way to execute checks and regex even in Active Tests. All checks or regex need to be specified between the instruction `start_validate(message_type) end_validate()` where `message_type` the type of message on which the operation should be applied. All other possible instructions are the same as those for the Passive Test (see Section 3.3.3.2). Two examples are reported in Listing 3.10 and 3.11.

```

1 start_validate(request) // begins the validation process
2 start_checks(message_type) // begins the checks
3 check(code, is present, url) // verifies if the given
   code is present in url
4 check_parameter(id, head, contains "23") // verifies if
   the specified 'id' parameter is present
5 end_checks() // all the checks for the current message
   type have been completed
6 end_validate() // the validation process for the given
   request has been completed

```

Listing 3.10: Example of validate operation.


```

1  start_validate(request) // begins the validation
   process
2  regex("code_challenge_method\\s*=(?!\\s*plain)", head,
   request) // performs a regular expression match on the
   request header, looking for the string "
   code_challenge_method" followed by an equals sign and
   any characters except "plain" (i.e. ensuring that the
   code challenge method is not set to plain).
3  end_validate() // the validation process for the given
   request has been completed

```

Listing 3.11: Example of validate operation with regex.

- **Intercept Operation:** operations to modify intercepted HTTP messages. All the instructions related to the intercept must be declared between the instructions `start_intercept` (`message_type`, `from_session`, `then`) and `end_intercept()` where (i) `message_type` is the type of message on which the operation should be applied, (ii) `from_session` specifies which Session has to be intercepted to search for the specific message and (iii) then defines the action to be done on the intercepted message after the execution of the operation and can assume `forward` also by default or `drop`. Inside the tags `start_intercept` (`message_type`, `from_session`, `then`) and `end_intercept()` it is possible to add:
 - (i) `save_message(variable)` - Instruction to save an intercepted message in a new variable where `variable` is the name of the variable created for the purpose of saving the message,
 - (ii) `replace(saved_variable, isRequest)` - Instruction to replace the intercepted HTTP message request or response with a previously saved message where `saved_variable` is the name of the variable which contains the HTTP request or response to replace and `isRequest` is set to true if the HTTP message is a request, otherwise it is set to false in case of a response.
- **Preconditions:** it is possible to define preconditions with regex or checklist instructions (same reported in Section 3.3.3.2) to be specified inside the tags `start_precondition()` and `end_precondition()`. An example of using precondition operations is shown in Listing 3.12.

```

1  start_precondition() // begin precondition
2  start_check(oauth request) // begin operation
3  check(param, is present, head) // verify if parameter is
   present in header
4  end_check() // end operation
5  end_precondition() // end precondition

```

Listing 3.12: An example illustrating the usage of the precondition.

- **Message Operations:** these are operations that need to be performed on the intercepted HTTP message in an operation. All instructions for executing message operations should be defined between the tags `start_msg_operation(message_section)` and `end_msg_operation()` where `message_section` can assume the values of `url` or `head` or `body`. The possible instructions are:
 - `add_parameter(param_name, param_value, from)` - Instruction to add a new header or string to the intercepted HTTP message where (i) `param_name` is the name of the header to add, (ii) `param_value` is the value of the header to add, (iii) `from` is the HTTP message section with possible values `url`, `head` and `body`. This instruction serves in two purposes (i) if `from` equals to `head` or `url`, a new header is added to either the URL or the header section of the HTTP message, with `param_name` as the name of the new header and `param_value` as its value and (ii) if `from` equals to `body`, `param_value` is appended to the `body` section of the HTTP message, without taking into consideration the `param_name` value.
 - `edit_parameter(param_name, new_value, from)` - Instruction to edit the given parameter's value of the intercepted HTTP message where `param_name` is the name of the parameter to edit, `new_value` is the new value of the parameter and `from` is the message section with possible values `url`, `head` and `body`.
 - `edit_regex(regex, new_value, from)` - Instruction to edit the content of the HTTP message matched by the passed regex with a new value where `regex` is the regex to apply passed between quotation marks "...", `new_value` is the new value of the content matched by the regex and `from` is the message section with possible values `url`, `head` and `body`.
 - `remove_parameter(param_name, from)` - Instruction to remove a parameter from the intercepted HTTP message where `param_name` is the name of the parameter to remove and `from` is the message section with possible values `url`, `head` and `body`.
 - `remove_match_word(word, from)` - Instruction to remove occurrences of a string from the HTTP message where `word` is the string to delete and `from` is the message section with possible values `url`, `head` and `body`.
 - `save_parameter(param_name, variable, from)` - Instruction to save the value of an HTTP message parameter in a new variable where `param_name` is the name of the parameter to save, `variable` is the name of the new variable in which to save the parameter value and `from` is the message section with possible values `url`, `head` and `body`.
 - `save_match(string, variable, from)` - Instruction to save a string found in the HTTP message in a new variable where `string` is the string to save, `variable` is the name of the new variable in which to save the string value and `from` is the message section with possible values `url`, `head` and `body`.

- `gen_POC(pattern, name)` - Instruction to generate and save as a name the HTML POC with a HTML form of pattern.
- `decode_param(param_name, from, encoding1,...,encodingN, type, remove signature, self-sign)` - Instruction to decode a parameter of the intercepted HTTP message where `param_name` is a mandatory parameter which reports the name of the parameter to decode, `from` is a mandatory parameter which reports the message section with possible values `url`, `head` and `body`, `encoding1,...,encodingN` is a list of encoding standard with possible values `url`, `deflate`, `jwt` and `base64`, `type` is the type of the parameter once decoded with possible values of `xml`, `jwt` and `txt`, `remove signature` it means that the decoded parameter is a SAML request or response and its signature is removed and `self-sign` it means that the decoded parameter is a SAML request or response and its signature is resigned with a private key.

According to the chosen type, one of the following instructions can be used:

- * if type equals to `xml` it means that the decoded parameter is an xml, then the instruction is like `xml_op(op_tag,...,op_tagN)` and `op_tag` assumes:
 - `(remove tag, tag_name)`: the instruction allows to remove the tag `tag_name`.
 - `(remove attribute, attr, tag_name)`: the instruction allows to remove the attribute `attr` of the XML tag `tag_name`.
 - `(edit tag, tag_name, new_value)`: the instruction allows to set the value of the tag `tag_name` to `new_value`.
 - `(edit attribute, attr, tag_name, new_value)`: the instruction allows to substitute the value of the attribute `attr` of the XML tag `tag_name` with `new_value`.
 - `(add tag, new_tag, parent_tag, new_value)`: the instruction allows to insert a new tag `new_tag` with value `new_value` as a child of the tag `parent_tag`.
 - `(add attribute, new_attr, tag_name, new_value)`: the instruction allows to insert a new attribute `new_attr` with value `new_value` in the tag `tag_name` as an attribute.
 - `(save tag, tag_name)`: the instruction allows to save the value of the tag `tag_name` in a new variable created on purpose.
 - `(save attribute, attr_name, tag_name)`: the instruction allows to save the value of the attribute `attr_name` of the tag `tag_name` in a new variable created on purpose.
- * if type equals to `txt` which means the decoded parameter is a plain text, then the instruction is like `txt_op(op_tag,...,op_tagN)` and `op_tag` assumes:
 - `(txt remove, value)`: the instruction allows to remove all the values `value` matched in the `txt` parameter.
 - `(txt edit, text, new_text)`: the instruction allows to substitute all the occurrences of text matched in the `txt` parameter with the string `new_text`.

- (`txt add, text, new_text`): the instruction allows to insert the string `new_text` at the end of the string `text` matched in the `txt` parameter.
 - (`txt save, text, new_var`): the instruction allows to save the string `text` in a new variable called `new_var`, if the string `text` has matched at least once in the `txt` parameter.
- * if type equals to `jwt` which means the decoded parameter is a JWT, then the instruction is like `txt_op(op_tag,...,op_tagN)` and `op_tag` assumes:
- (`jwt remove, from, parameter`): the instruction allows to remove the parameter `parameter` from the specific `jwt` section specified in `from`. If the section is signature the entire signature is removed.
 - (`jwt edit, from, parameter, new_value`): the instruction allows to substitute the value of the parameter `parameter` with a new value `new_value` in the specific `jwt` section specified in `from`. If the section is signature then the entire signature is edited.
 - (`jwt add, from, parameter, new_value`): the instruction allows to add a new parameter `parameter` to the specific `jwt` section (specified by `from`), with value `new_value`.
 - (`jwt save, from, parameter, new_var`): the instruction allows to save in a new variable `new_var` the parameter `parameter` of the `jwt`. If the section is signature then the entire signature is saved.
 - (`jwt sign`): the instruction allows to sign the `jwt` with an invalid key.

Possible values of the field `from` are `header`, `payload`, `signature`
`raw header`, `raw payload` and `raw signature`.

3.3.4 MIG-L Semantics

In this section, we will present the semantics of MIG-L. We define the semantics of MIG-L by translating it into a machine readable language, which is the low-level input language for MIG-T. This language is in JSON format and leverages the primitives described in Section 3.3 (i.e. Browser API, Proxy API and Session API) and detailed in Section 3.3.4.3. A mapping between MIG-L operations and machine readable format expressed in JSON is reported in in Appendix B.

3.3.4.1 Translation of Testsuite

The translation of the object Testsuite defined in Section 3.3.3.1 is shown in Listing 3.13.

```
1 {"test suite": {  
2   "name": "Testsuite for SAML", // assigned label of the  
   testsuite  
3   "description": "Set of test to verify the SAML Request"  
   // define a description of the testsuite  
4   "filter messages": "true" } // allows the proxy to  
   intercept the traffic by invoking a Proxy API  
5 "tests": [...] } // list of tests
```

Listing 3.13: Example of Testsuite.

The `ProxyListener` interface is used to register a listener for the Proxy, which receives notifications of all requests and responses processed. The function `registerProxyListener()` is invoked from the Proxy API to enable the Proxy to intercept HTTP messages and enable the execution of the list of Test.

When the Testsuite is executed on the MIG-T, it follows a procedure where each Test is evaluated independently and the outcome of each Test is reported (passed or failed or not applicable).

Different scenarios have been outlined based on the type of Test contained in the Testsuite:

- If the Testsuite only contains Passive Test, they will only be executed whether the previous Session and its HTTP message flow is saved.
- If the Testsuite only contains Active Test, they will be executed one after another.
- If the Testsuite contains both Active Test and Passive Test, the Active Test will be executed first, as they may modify the message flow of the Session. Once all Active Test are done, the Passive Test will be run on the saved message flow.

3.3.4.2 Translation of Test

The translation of the object Test defined in Section 3.3.3.2 is different according to the value of the type parameter (i.e. `active` or `passive`).

In the following paragraphs we specified the translations for Passive and Active Test, respectively.

Passive Test. Passive Tests are executed on the saved flow of HTTP messages, thus the Session identified with tag `session` in the Test associated with the Test must be run beforehand to save the message flow.

If the Session has been executed, the Passive Test is carried out by iterating through the HTTP messages. For each HTTP message, the execution of each `operations` is evaluated by comparing it to the specified `types` in the operation. The outcome of this comparison is then used to determine the next step:

- If a match is found between the `types` and the HTTP message, and the operation results in a fail, the entire test is considered a failure.
- If a match is found between the `types` and the HTTP message, and the operation results in a pass, the iteration on the HTTP message continues as there may be other messages that can be processed by the same operation.
- If no match is found between the `types` and the HTTP message, the iteration on the HTTP message continues as a suitable message on which to apply the operation has not yet been found.

The final outcome of the test can be one of the following:

- Failed: If the result of any `operations` is failed.
- Not applicable: If no messages were found to apply at least one `operations`.
- Passed: If the outcome of each `operations` is passed for every message of the flow it is applicable to.

Listing 3.14 demonstrates how to create a Passive Test.

```
1 "test": {
2   "name": "Presence of state parameter",
3   "description": "Verify the presence of state parameter",
4   "type": "passive",
5   "operations": [//passive operations]}}
6 {//other test}]
```

Listing 3.14: Passive Test creation example.

Listing 3.15 provides an example of how operations can be used in a Passive Test.

```
1 {"operations": [{ //regex operation example
2   "message type": "Authorization request",
3   "regex": "code_challenge|code_challenge_method",
4   "message section": "body"} // check list operation example
5 { "message type": "authorization request",
6   "checks": [{
7     "in": "head",
8     "check": "response_type",
9     "is": "code"}, {
10    "in": "head",
11    "check": "client_id"}]]}
```

Listing 3.15: Example of adding operations in a Passive Test.

When a HTTP Message stored matches a specified types, the operations is applied to that HTTP Message. In scenario one, when an operations in a Passive Test is a regex, the following considerations are made:

- The message section tag indicates which part of the message (url, headers, or body) the regex should be applied to. In Listing 3.16 we report the Proxy API used.

```
1 //to get headers (interface AnalizeHttpRequestMessage)
2 List<String> getHeaders(HttpRequestResponse message);
3
4 //to get body offset (interface AnalizeHttpRequestMessage)
5 int getBodyOffset(HttpRequestResponse message);
6 //to get the entire message (request or response) as a
   byte[] array
7 byte[] msg = HttpRequestRequest.getRequest()
8 byte[] msg = HttpRequestResponse.getResponse()
9 /*NB: then thanks to the offset and the entire byte[]
   message, the byte[] body is retrieved with another
   function.*/
10
11 //to get url (interface AnalizeHttpRequestMessage)
12 String getUrl(HttpRequestResponse message);
```

Listing 3.16: Example of the Proxy API used.

- The relevant part of the HTTP Message is extracted as a string.

- The `regex` is applied to the string.

The result of the `operations` is then:

- Passed (true) if the `regex` matches at least one occurrence in the specified string.
- Failed (false) if the `regex` does not match any occurrences in the specified string.

In scenario two, when an `operations` in a Passive Test is a list of `checks`, the following considerations are made for each check in the list of `checks`:

- Tag `in` is used to determine which part of the message (`url`, `headers`, `body`) should be checked. The relevant part of the HTTP Message is then extracted as a string.
- Two main tags are considered, `check` and `check parameter`, each of which may be associated with additional tags:
 - The `check` tag with a value is specified, it verifies if the message string contains the specified value; if the string is found, the check results is Passed.
 - When the `check parameter` tag is specified with a value, the system checks if the message string contains the specified value as a parameter. If the parameter is present in the string, the check is considered Passed.
 - Additional tags can modify the behavior of the `check` and `check parameter` tags described previously. These tags include:
 - * `is:value` also used in conjunction with the `check parameter` tag.
 - * `not is:value` also used in conjunction with the `check parameter` tag.
 - * `contains:value` also used in conjunction with the `check parameter` tag.
 - * `not contains:value` also used in conjunction with the `check parameter` tag.
 - * `is present:true/false` also used in conjunction with the `check parameter` and `check` tags.

The outcome of the `operations` is:

- Passed (true) if all the `checks` are evaluated as true.
- Failed (false) if any of the `checks` are evaluated as false.

Active Test. Active Test is run in conjunction with the Session that are tagged to the Test, when the MIG-T is instructed to execute the Testsuite. The execution of the associated Session must be repeated for each Test. When running the Active Test, an iteration over the `operations` begins. Each `operations` is evaluated sequentially, meaning that the next `operations` will not be available until the previous `operations` has completed. Listing 3.17 provides an example of the structure of an active test.


```

1 {"test": {
2   "name": "Replay SAML Response",
3   "description": "Replay an existing SAML Response",
4   "type": "active",
5   "sessions": [
6     "s1"],
7   "operations": [...]}
8   "result": "incorrect flow s1"}}

```

Listing 3.17: Structure of an Active Test.

In an Active Test, the Proxy API is used to intercept HTTP Message. When an HTTP Message is intercepted, the function `ProxyListener.processProxyMessage` (`HttpRequestResponse message`) is called, and all the operations of the Active Test are executed within this function.

Oracle definition The oracle is responsible for determining the outcome of Active Test and takes into account the following three components of the language in the specified order:

- Evaluation of the `preconditions`.
- Evaluation of the full or partial execution of the `Session`.
- Evaluation the `operations` of `validate`.

Preconditions in an Active Test's operations are defined using the `precondition:[...]` tag, which contains a list of checks or a `regex`. If the list of checks or `regex` evaluates to false, the entire Test is considered not supported and the result of the entire Test is considered Not applicable. Even if an operation within an Active Test is deemed Not applicable, the entire test will still be considered as Not applicable. The `result` tag determines the outcome of the Test by evaluating the execution of the `Session` and if it is not specified then the execution of all the `Session` are taken in consideration.

If the `result` is set to `correct flow SessionName`, then the Test will be considered:

- Passed: if all the user actions specified in the `SessionName` are executed without errors
- Failed: if there are errors in the execution of the user actions specified in the `SessionName`.

If the `result` is set to `incorrect flow SessionName`, then the Test will be considered:

- Passed: if there are errors in the execution of the user actions specified in the `SessionName`.
- Failed: if all the user actions specified in the `SessionName` are executed without errors.

If the result is set to `assert_only`, the execution of all the user actions specified in a Session is skipped and the Test is evaluated based on the outcome of the operations in the `validate`. The test is considered:

- **Passed:** if all the results of the operations in `validate` are passed.
- **Failed:** if all the results of the operations in `validate` are failed.

When operations in `validate` are present in a Test, the final outcome of the Test is evaluated using the boolean operator **AND**:

- **Passed:** if (i) the result of the Test is Passed using the tag `result:correct flow SessionName` or `result:incorrect flow SessionName` **AND** each operations result in `validate` is passed and, (ii) if each operations result in `validate` is passed and tag is `result:assert_only`.
- **Failed:** in all other cases, meaning if either the result of the test is failed or at least one operations result in `validate` is failed or if tag `result:assert_only` and not all operations result in `validate` is passed.

There are two types of operations in Active Test, the one for Session and one for Test.

Handle the Session The following are the operations used to manage the different Session specified in an Active Test, identified by the tag `sessions`. It is important to note that each Session runs on a separate thread and is associated with a unique port number to allow for distinct message interceptions between Session. First, when it comes to Session Config Operations, the tag `session SessionName` is used to specify the particular Session on which actions will be performed. Then, based on the value of the tag `action` a specific action is executed:

- `start`: the Session is initiated. A new thread is created and the Session runs on it.
- `stop`: the Session is terminated. The Thread of the Session is interrupted.
- `pause`: the Session is temporarily suspended. The Thread of the Session is put on hold.
- `resume`: the Session resumes execution. The Thread of the Session is resumed.
- `clear cookies`: the cookies saved in the browser due to execution of the Session are deleted.

In Listing 3.18, an example to handle Session with operations.

```
1 "operations": [{
2   "session": "s2",
3   "action": "start",
4   "session operation": [
5     {"session": "s2",
6     "insert top": "open | www.example.com"}]]]
```

Listing 3.18: Example of session operation.

Operations on Session The operations in the Session are executed sequentially. For each operations, the `session` tag is used to indicate that all modifications will be carried out on the specified Session. The specific action to be taken depends on the other tag used, which must be one of the following options exclusively:

- If the tag `mark` is used in conjunction with a `MarkerName`, a new marker object with the value `MarkerName` is created to identify the index of the user action specified by the tag `mark`. The possible user actions that can be marked are:
 - `mark:last_action` to mark the last user action performed.
 - `mark:last_open` to mark the last open user action performed (i.e. `open | URL |`).
 - `mark:last_click` to mark the last click user action performed (i.e. `click | link=login |`).
 - `mark:all_assert` to mark all the assertions of the Session with the same marker.
- If the tag `save` is used in conjunction with the tag `as` with the value `variable`, a new variable is created. If the action is followed by `.elem` only the element within the user action will be saved. Furthermore, if `.elem` is followed by `.parent` only the parent div element will be saved. The new variable will be used to save a specific user action or element based on one of the following options:
 - `save:last_action` saves the last user action performed.
 - `save:last_open` saves the last open user action performed (e.g., `open | example.com |`).
 - `save:last_url` saves the last url that was opened.
 - `save:last_click` saves the last click user action performed (e.g., `click | id=login |`).
 - `save:all_assert` in this case all the action of the Session, which are considered assertion are saved as a concatenation in variable.

- `save:track` associated with the tag `range: [M0, M1]`, in this case all the action in a range defined with the use of markers are saved in variable.
- If the tag `insert` with `Variable1` and `Variable2` is used with the tag `at`, a new string is created by concatenating the text and the content of each `Variable` specified with the tag `insert`. This new string is then inserted into the `Session` at the location specified by the marker indicated in the `at` tag as a new user action.
- If the tag `remove` with the value `Marker` is used, the user action located at the position specified by the `Marker` is deleted within the `Session`.

Operations on Test In the case of an operations in a Test, the first step is to intercept a message on which the operation will be applied. An example of operations in Active Test is reported in Listing 3.19. If the tag `types` is used, the message must be of that specific type. If not, the message must be of a default type such as `request` or `response` or `oauth request` or `oauth response`. After intercepting the specific message, the tag action is evaluated:

- If the tag `action` is set to `validate` the operations are equivalent to operations in Passive Test (see Section 3.15).
- If the tag `action` is set to `intercept`, the execution proceeds with the following consideration:
 - if there are any preconditions specified in the `preconditions`, the execution will continue only if these preconditions are met.

At this stage, all other optional tags are evaluated in a specific order:

- The `from session` is used to specify on which `Session` should be searched the message. If an HTTP message is intercepted, the port number of the specified `Session-Name` must match the port number of the `Session` where the message was intercepted in the `operations`. If the port number does not match, the message will not be considered. If `from session` is not used, the message will be searched by default at port 8080.
- The `then` tag is used to specify the action that should be taken on an intercepted message after the `operations` is executed. The options are:
 - * `forward`: in this case, after all operations have been performed on the HTTP Message, it is sent to the server/client normally.
 - * `drop`: in this case, the HTTP Message is deleted and not forwarded to the client/server.
- `replace request: Variable` is used when the intercepted HTTP Message is a request and will be replaced by `Variable`. If the intercepted HTTP Message is not a

- request or the Variable containing the HTTP Message is not found, then the operation result is Not applicable. The `HttpRequestResponse.setRequest(byte[] message)` method of the Proxy API is used to update the request of the HTTP Message.
- `replace response: Variable` is used when the intercepted HTTP Message is a response and will be replaced by Variable. If the intercepted HTTP Message is not a response or the Variable containing the HTTP Message is not found, then the operation result is Not applicable. The `HttpRequestResponse.setResponse(byte[] message)` method of the Proxy API is used to update the request of the HTTP Message.
 - `message operations: [...]` All the specified operations to editing the HTTP Message are executed.
 - `save` An intercepted HTTP Message is stored in a new variable for use in future operations. It utilizes the following Proxy API methods: (i) to save a response through `byte[] msg = HttpRequestResponse.getResponse()` and, (ii) to save a request `byte[] msg = HttpRequestResponse.getRequest()`.

```

1  "operations": [{
2    "action": "intercept", // action to do
3    "then": "forward",
4    "message type": "authorization request",
5    "preconditions": [{
6      "check param": "state",
7      "is present": true}],
8    "message operations": [
9    // list of operations to do on message
10   { "from": "url",
11     "remove parameter": "code_challenge_method"}]]

```

Listing 3.19: Example of operations on Test.

HTTP Message Operations If the tag `message operations` is present, a loop is started to iterate over the list of message operations. For each operation, all the tags are evaluated. When evaluating a specific message operation, the mandatory `from` tag is first considered, which specifies the section of the message where the action should be performed.

The possible actions are then allowed:

- If the tag `remove parameter` is present with the `ParameterToRemove`, it removes the `ParameterToRemove` from the specified section of the HTTP Message.

- If the tag `add` is present with the `ParameterToAdd` and its `Value`, it adds `ParameterToAdd`, then:
 - If the message section is `url` or `head`, a new header called `ParameterToAdd` is added to the `url` or the `head` section of the HTTP Message with `Value`.
 - If the message section is `body`, the `Value` is appended to the `body`.
- If the tag `edit` is present with the value of `ParameterToEdit` and the tag `in` is present with the value `NewParValue`, then:
 - If the message section is `url` or `head`, the parameter `ParameterToEdit` is searched and its value is updated to `NewParValue`.
 - If the message section is `body`, a regular expression is used to substitute every occurrence of `ParameterToEdit` in the `body` with the value `NewParValue`.
- If the tag `edit regex` with the value `Regex` is present, followed by the tag `in` with the value `NewContent`, then a regex is used to replace any matching content in a specific message section with `newContent`.
- If the tag `remove match word` with the value `Word` is present, then a regex is used to remove the string `Word` from a specific message section.
- If the tags `save` with the value `Variable1` and `as` with the value `Variable2` are present, then the following actions will be taken:
 - If the message section is `url` or `head`, the `save` tag will be used to search for a specific `Variable1` and its value will be saved in `Variable2`.
 - If the message section is `body`, the `save` tag will be used with a regex to save any matching content in `Variable1` and saves in `Variable2`.
- If the tag `save matches` with the value `String` is present, followed by the tag `as` with the value `Variable`, then the specific message section is searched for `String` and if it is found, it is saved in a new variable named `Variable`.
- If the tag `decode param` with the value `Parameter` is present, as shown in Listing 3.20 then the following actions will be taken:
 - If the message section is `head` or `url`, the value of `Parameter` is extracted from the specific message section and decoded using a list of encoding.
 - If the message section is `body`, a regex is used to match content, and whatever is matched is decoded using the list of encoding.
- The `encoding` tag, which includes a list of values, determines the encoding that can be applied to the encoded parameter. The encoding are applied in the order they are listed, and a unique function is utilized to decode the parameter for each encoding.
- If the `remove signature` tag is set to `true`, the signature is removed from the decoded parameter.

- If the `self-sign` tag is set to `true`, the signature is removed and the decoded parameter is re-signed with a private key.
- If the `type` tag is set to `xml`, the decoded parameter will be treated as an XML document. The tags that specify the operations are:
 - `remove tag`: Delete an existing tag by specifying its name in the XML parameter.
 - `remove attribute`: Delete an existing attribute from a specific XML tag by specifying the attribute name and the tag name.
 - `edit tag`: Change the value of an existing tag by specifying the tag name and the new value.
 - `edit attribute`: Change the value of an existing attribute in a specific XML tag by specifying the attribute name, the tag name, and the new value.
 - `add tag`: Add a new child tag to an existing parent tag by specifying the new tag name, the parent tag, and the new value.
 - `add attribute`: Add a new attribute to an existing XML tag by specifying the new attribute name, the tag name, and the new value.
 - `save tag`: Save the value of an existing tag in a new variable by specifying the tag name.
 - `save attribute`: Save the value of an existing attribute in a new variable by specifying the attribute name and the XML tag name.
- If the `type` tag is set to `txt`, the decoded parameter is treated as a string and regex operations are applied. The following actions can be taken:
 - `txt remove`: Remove specified value from the parameter.
 - `txt edit`: Replace all occurrences of a specified text in the parameter with a new text.
 - `txt add`: Add a new text at the end of a specified text in the parameter string.
 - `txt save`: Save a matched string text in a new variable with a specified name.
- If `type` tag is set to `jwt` is present, it is possible to decode a JWT token using the following tags:
 - `jwt from`: This tag is mandatory and is used to specify the section of the token on which the action should be taken (possible values are: `header`, `payload`, `signature`, `raw header`, `raw payload`, `raw signature`).
 - `jwt remove`: Remove a specified parameter from a specific section of the JWT.
 - `jwt edit`: Replace the value of a specified parameter in a specific section of the JWT with a new value. (Note: If the section is the signature, the entire signature is edited.)

- `jwt add`: Add a new parameter with a specified value to a specific section of the JWT.
 - `jwt save`: Save the value of a specified parameter in a new variable. (Note: If the section is the signature, the entire signature is saved.)
 - `jwt sign`: Sign the JWT with an invalid key.
- If `type` is set to `gen_poc` by providing the message type and the value is, it is possible to generate an HTML Proof Of Concept as value of the HTTP message which corresponds to the message type.

```

1 "message operations": [ {
2   "from": "url",
3   "decode param": "SAMLRequest",
4   "encoding": [
5     "url",
6     "base64",
7     "deflate"],
8   "type": "xml",
9   "xml tag": "samlp:AuthnRequest",
10  "edit attribute": "ID",
11  "value": "newIDValue"}]

```

Listing 3.20: Example of using the `decode param`.

3.3.4.3 The Required APIs by MIG-T

As reported in Figure 3.3 we identified three different types of APIs (*i*) Proxy, (*ii*) Session and (*iii*) Browser.

The Proxy API refers to a set of interfaces and functions offered by the Proxy, which are essential for the proper functioning of MIG-T. These functions and interfaces are therefore necessary:

- Interface `HttpRequestResponse`: this interface is used to retrieve and update details about the intercepted HTTP messages, providing functions to modify the content of the message. The supported functions are:
 - `byte[] getRequest()`: this function retrieves the HTTP request as a byte array.
 - `byte[] getResponse()`: this function retrieves the HTTP response as a byte array.
 - `setRequest(byte[] request)`: this function updates the HTTP request passed as a `byte[]` array.

- `setResponse(byte[] response)`: this function updates the HTTP response passed as a `byte[]` array.
- **Interface ProxyListener**: this interface is used to register a Proxy listener. The listener will be notified of requests and responses being processed by the Proxy tool.
 - `registerProxyListener()`: this function enable the proxy to start intercepting HTTP messages.
 - `processProxyMessage(HttpRequestResponse message)`: function called when an HTTP message is being processed by the Proxy, all the information of the intercepted HTTP message are saved in the passed `HttpRequestResponse` object.
- **Interface AnalyzeHttpMessage**: This interface is used to provide function to retrieve key details about an HTTP message.
 - `List<String> getHeaders(HttpRequestResponse message)`: this function return the headers of an HTTP message.
 - `int getBodyOffset(HttpRequestResponse message)`: this function return the offset within the HTTP message where the message body begins.
 - `String getUrl(HttpRequestResponse message)`: this function return the url of the HTTP message in case of a request.

The Session API is a collection of functions utilized to manage the operation of sessions. Each Session is an object functioning as a thread, and therefore, the APIs consist of functions for thread management. The following APIs have been identified:

- `Thread start(Session sessionName)`: this function run a specific Session as a new thread object which is given in return.
- `void sleep(Thread thread1, int time)`: this function pauses the thread on which the Session is running, consequently pausing the execution of the Session.
- `void notify(Thread thread1)`: this function resumes the thread on which the Session is running, consequently resuming the execution of the Session.
- `void interrupt(Thread thread1)`: this function stops the execution of the thread on which the Session is running, consequently stopping the execution of the Session.

The Browser API refers to a collection of functions utilized to facilitate automated execution of user actions during a browser session. We identified the following functions:

- `WebDriver driver = new ChromeDriver(options)/new FirefoxDriver(options)`: this constructor initializes the browser according to the chosen browser.

- `driver.getCurrentUrl()`: this method retrieves the current url opened.
- `driver.get(url)`: this method opens in the browser the passed url.
- `driver.manage().deleteAllCookies()`: this method explicitly deletes all the cookies generated.
- `driver.findElement(by)`: this method finds an element in a HTML page.
- `WebDriverWait(driver, Duration.ofSeconds(TIMEOUT)).until(ExpectedConditions.elementToBeClickable(currentElement)).click()`: this method is called when a click user action is read.
- `WebDriverWait(driver, Duration.ofSeconds(TIMEOUT)).until(ExpectedConditions.elementToBeClickable(currentElement)).sendKeys(action.content)`: this method is called when a type user action is read.

3.3.5 Machine Readable Specification of Testsuite

We analyzed and collected existing BCPs, vulnerabilities, and attacks related to various sources of IdM protocols, particularly SAML and OAuth/OIDC protocols. Based on our analysis, we created testsuites defined in the machine readable specification described in previous section. In the following sections, we provide an example of Test for SAML, OIDC and OAuth protocols.

3.3.5.1 SAML Testsuite

An excerpt of a SAML test from [Fre04] in machine readable specification is displayed in Listing 3.21.

```
1  "test": {
2  "name": "Missing check on OneTimeUse in Response",
3  "description": "Checks whether the OneTimeElement element is
4     present in the SAMLResponse",
5  "type": "passive",
6  "operations": [{
7     "message type": "saml response",
8     "decode param": "SAMLResponse",
9     "encoding": [
10    "url",
11    "base64",
12    "deflate"],
13    "checks": [{
14       "in": "url",
15       "check param": "SAMLResponse",
16       "contains": "OneTimeUse"}],
17    "message section": "body"}]}
```

Listing 3.21: Example of a SAML Test in JSON Format.

3.3.5.2 OIDC Testsuite

An excerpt of a OIDC tests from [Mic15] in machine readable specification is displayed in Listing 3.22.

```
1  "test":{
2    "name": "Pl_a",
3    "description": "Authentication Request: client_id parameter
4      must be present in the JWT token",
5    "references": "OIDC CIE Core Web, Section 5, Table 6",
6    "violated_properties": "client_id parameter is required in
7      the JWT token",
8    "mitigations": "OP should validate the client_id parameter
9      in the JWT token",
10   "affected_entity": "OP",
11   "type": "passive",
12   "operations": [{
13     "message type": "authentication_request",
14     "message section": "url",
15     "decode param": "request",
16     "encoding": [
17       "jwt"],
18     "regex": "\"client_id\"\\s?:\\s?\".*\"}],
19   "result": "incorrect flow s1"}
```

Listing 3.22: Example of an OIDC Test in JSON Format.

3.3.5.3 OAuth Testsuite

An excerpt of a OAuth tests from [oaub] in machine readable specification is displayed in Listing 3.23.

```
1 "test": {
2   "name": "Parameter state is used",
3   "description": "check for the presence of the parameter state
4     in relevant messages",
5   "type": "passive",
6   "operations": [{
7     "message type": "authorization request",
8     "checks": [{
9       "in": "head",
10      "check": "state",
11      "is not": ""}], {
12     "message type": "authorization response",
13     "checks": [{
14       "in": "head",
15       "check": "state",
16       "is not": ""}], {
17     "message type": "token request",
18     "checks": [{
19       "in": "head",
20       "check": "state",
21       "is not": ""}]}}]}
```

Listing 3.23: Example of an OAuth Test in JSON Format.

Chapter 4

Micro-Id-Gym Design

In this chapter, we present the design of MIG, a powerful and user-friendly tool that assists system administrators and Security Testers in the deployment and security testing of instances of IdM protocols. The tool is designed to streamline and automate many of the tasks involved in the setup and configuration of IdM protocol instances, making it easier for administrators to deploy and manage these systems. It also provides an extensive set of tools for Security Testers to identify and exploit vulnerabilities in instances of IdM protocol. Some of its features include *(i)* automatic and unattended installation of IdM protocol, *(ii)* easy configuration of security settings *(iii)* automated security testing and, *(iv)* automated remediation of vulnerabilities, making it easy to fix issues as they are identified. With MIG, system administrators and testers can confidently deploy and manage instances of IdM protocol, knowing that their systems are secure.

4.1 Overview

To assist system administrators and testers in the deployment and security testing of IdM protocol instances we propose MIG. In this section we provide an overview of the tool before giving the details of the various components in the rest of this section.

The IdM protocols are designed specifically for the transfer of authentication information and consist of a series of messages in a preset sequence designed to protect data as it travels through networks or between servers. All the IdM protocols provide standards for security to simplify access management, help in compliance, and create a uniform system for handling interactions between users and systems.

Figure 4.1 shows a high level view of the architecture of MIG composed of three main components, namely MIG Backend (that supports the creation of a sandbox), Dashboard (that assists the users to set the tool) and, MIG Frontend (that supports security testing).

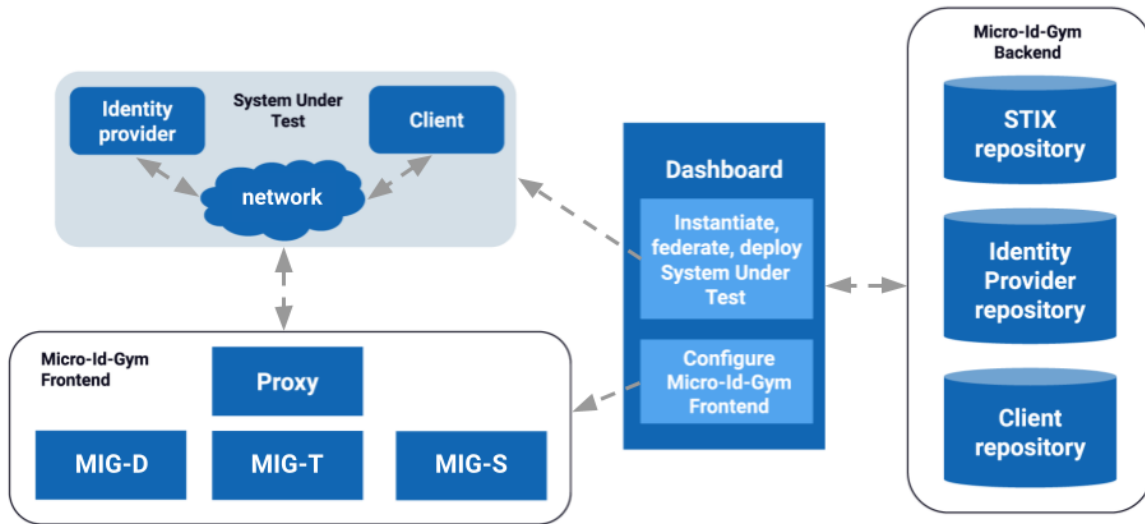


Figure 4.1: Overview of MIG.

Abstractly, MIG supports two main activities: security testing of IdM protocol deployments and creating sandboxes with an IdM protocol deployment. The former can be carried out on a deployment in the wild or one in a sandbox—say, in the laboratory—obtained by the second activity. We observe that the capability of creating sandboxes is useful to perform attacks with high impact like DoS or identity theft, the former dangerous for the service itself while the latter for legal and compliance issues.

4.2 MIG Backend

The MIG Backend is used to recreate locally a sandbox as an instance of an IdP and a Client and it can be done by uploading the own proprietary sandbox or by composing a new sandbox choosing the instances of IdPs and Clients provided by the tool as depicted in Figure 4.1 in the IdP and Client repositories. All the provided instances have been collected so far during our experience of using MIG and they satisfy the requirements to be compatible with MIG namely to have an instance of the IdP, at least one for the Client and to use SAML or OAuth/OIDC as IdM protocols. The backend is also in charge to instantiate the selected instances, to federate each other, to exchange the required metadata, to perform the deployment of the sandbox and to set up the local network.

The MIG Backend provides also Cyber Threat Intelligence (CTI) information useful for assessing vulnerabilities and threats related to the chosen instances. These data follow the Structured Threat Information Expression (STIX) format proposed by OASIS CTI TC.¹ This information is very useful since it immediately makes the Security Testers aware of possible specific attacks of that protocol known in the literature.

The goal of the MIG Backend is by construction to provide a test environment generator tailored to IdM protocols and deploy the environment in the SUT. Given a set of available IdM protocol implementations collected while using the tool for third parties, the SUT automatically sets-up a working environment in a local network. The main reason is to allow system administrators to recreate locally in the laboratory their production environments, being able to perform security testing them in sandboxes. As depicted in Figure 4.2, the MIG Backend is composed by a set of IdPs and Clients instances both for OAuth/OIDC and SAML, and a STIX notes repository. The set of available instances is indeed a work in progress, and can be easily extended/updated over the time. By design, the architecture allows continuous integration of newer and different implementations.

4.3 Dashboard

The Dashboard serves as a central hub for managing the entire process of creating and configuring sandboxes. It enables users to easily select the necessary components, such as the IdP and Clients instances, from a set of available options in the Clients and IdPs repositories (Figure 4.1). Once the selection and configuration of URLs and ports for the chosen instances is complete, the tool automates the remainder of the process, including the instantiation, federation, and deployment of the SUT. The Dashboard also enables users to customize the sandbox by adding test users.

Besides helping the creation of sandboxes, the Dashboard also facilitates the configuration of tools in the MIG Frontend. The Dashboard provides users with the ability to specify the ports and URLs at which the various tools of the MIG Frontend are running. This enables users to easily configure the different tools and settings of the MIG Frontend, as well as monitor the status of the tools and consult log files for troubleshooting any issues that may arise.

In summary, the Dashboard serves as a comprehensive and efficient means of managing the creation, configuration, and customization of sandboxes, as well as the configuration of tools in the MIG Frontend. The ability to specify the ports and URLs at which the various tools of the MIG Frontend are running, ensures a smooth and accurate configuration process.

The component editor (i.e. the person in charge to configure the SUT) can select the IdM protocols, an IdP instance and one or more Client instance(s) he wants the SUT to deploy, among the

¹<https://www.oasis-open.org/committees/cti/>

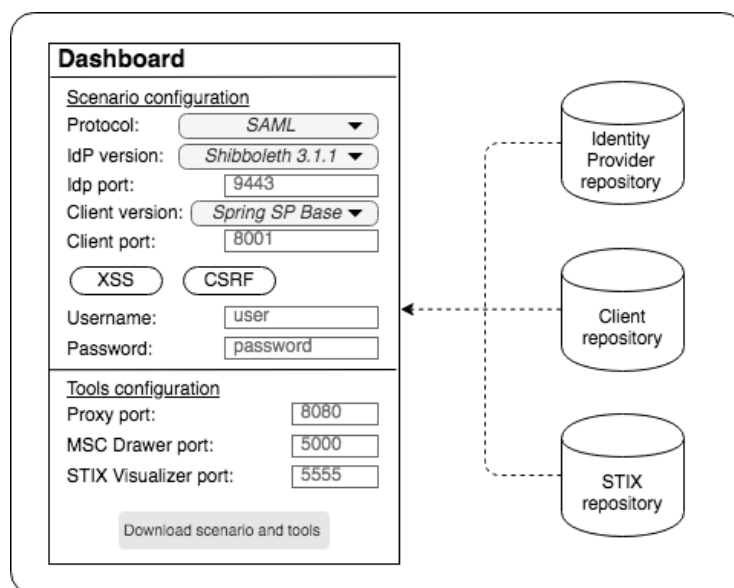


Figure 4.2: MIG Backend.

ones available. At the moment of the selection, the Dashboard consults the STIX notes repository and shows which known security issues the selected entities might suffer (e.g., XSS and CSRF in Figure 4.2). The user can also insert customized credentials to authenticate on the IdP and configure the ports where Client and IdP will run. Once the selection has been completed the SUT will (i) generate and deploy the sandbox, (ii) create a local network within the agents, (iii) perform the federation of the entities and (iv) set the credentials for each IdP instance.

4.4 MIG Frontend

The MIG Frontend is designed to support various user security testing activities on the SUT. It comprises of a set of tools that are tailored to meet the specific needs of Security Testers. These tools include (i) a Proxy which allows for the interception and manipulation of network traffic between the SUT and external systems, (ii) a MIG-T with a suite of security testing functions, (iii) MIG-D which allows users to easily visualize and analyze the results of security testing activities in the form of graphical representations and, (iv) MIG-S which facilitates the creation and sharing of cyber threat intelligence information. All of these tools are integrated and can be used together to provide a complete security testing solution for the SUT.

As depicted in Figure 4.3, the MIG Frontend is composed of MIG-T, MIG-D (consisting of MSC Logger and MSC WebApp), MIG-S and a Proxy.

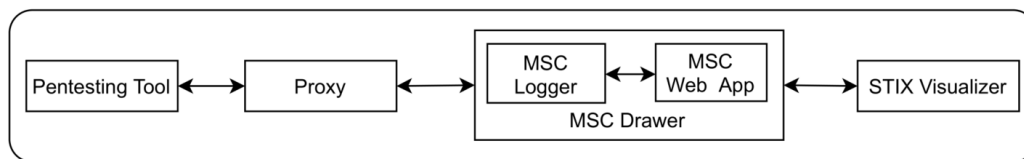


Figure 4.3: MIG Frontend.

4.4.1 Proxy

It is a web proxy tool used to intercept and manage the traffic between a user agent, such as a web browser, and a SUT. It offers a comprehensive set of APIs that are utilized by security testing tools to inspect, modify, replay, and drop intercepted messages.

4.4.2 MIG Tool

It supports a user to perform automatic security testing of an IdM protocol deployment, by providing instruments to automatically detect security issues. The MIG-T performs tests on both the IdP and Client 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.

In addition, for every detected vulnerability, our tool returns the HTTP messages that may cause the flaw and suggests mitigations so to allow users to understand how to (manually) fix the issue.

The tests executed by the MIG-T can be passive or active:

- **Passive tests:** tests done analyzing statically the intercepted HTTP messages without any interaction/modification of the HTTP messages during execution of the IdM protocol.
- **Active tests:** tests that need an interaction during the execution of the IdM protocol. After the initial execution, the user actions are stored and automatically re-executed while intercepting/changing the content of the messages before sending them to the Client and IdP.

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 makes 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 [And19]. In case the SUT notices the change, the test fails meaning that the SUT performs adequate verification on the `relaystate` parameter.

4.4.3 MIG Drawer

The intercepted messages by the proxy are processed by the drawer, which illustrates their flow using a Message Sequence Chart (MSC). MSCs can be utilized to depict the flow of messages between entities in a system, and can aid in the analysis and identification of potential vulnerabilities or attack scenarios. In particular, an MSC can be used to describe the various authentication, authorization, and encryption protocols and their respective impact on the security of the system and the exchanged messages. As such, the MSC can serve as a valuable tool for security analysts and developers in identifying security issues and designing and implementing effective countermeasures.

The MIG-D provides a graphical representation of the authentication flow and facilitates the inspection of exchanged messages by utilizing the information collected by the proxy. For each HTTP message, the Security Testers has the capability to examine headers, parameters, and body. Standards for IdM protocols typically dictate the mandatory/optional messages and their format, as well as the endpoints to be invoked. However, they generally do not specify the interactions between subsequent requests to an endpoint. As a result, standard messages may be interleaved with other non-essential messages. These “spurious” messages can make it a time-consuming task for a Security Tester to extract information about the standard. While current web proxy tools provide searching features, it can still be challenging to identify the relevant standard messages among the spurious ones. The MIG-D allows a Security Tester to quickly recognize whether the SUT adheres to the expected flow or not.

The Figure 4.4 depicts the main components of the MIG-D. The MSC Logger is a Proxy’s component responsible to capture a selection of the HTTP messages, to filter and parse them according to the specific configuration, and send them through API to the MSC WebApp, a web application exposing a set of Restful API and responsible to draw a MSC.

The MSC Logger has indeed the following functionalities:

- Configuration: it allows to set a password not allowing to overwrite a MSC already drawn and setup the URL of the MSC WebApp (the configuration information can be also provided through a configuration file, generated through the Dashboard of the MIG Backend).
- Interceptor: it collects all the intercepted HTTP messages.
- Filter: it allows to add filters in terms of keywords of the HTTP messages that will be collected and reported in the MSC. The keywords can refer to `Host`, `Request Headers`,

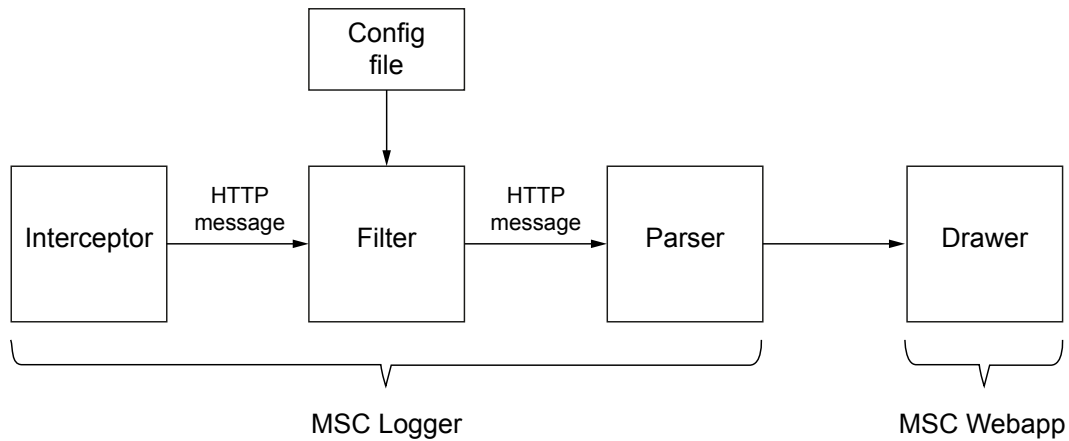


Figure 4.4: MIG-D components.

Request Parameters, Response Headers and Response Body.

- **Parser:** it allows to add rules for mapping keywords or sequence of keywords in new terms in order to further improve the readability of the MSC. For instance, in a training context, it is possible to provide a MSC closer to the abstract view of the protocol under test (e.g., by mapping the actual URL of a Client).

The MIG-D offers pre-configured filtering and parsing rules for commonly used protocols such as SAML and OIDC/OAuth. This allows for streamlined analysis of messages and identification of potential vulnerabilities. Additionally, the Security Tester has the capability to create custom rules, providing added flexibility and enabling the ability to analyze messages for protocols not already covered by the pre-configured rules.

4.4.4 MIG STIX Visualizer

One of the key elements in identifying vulnerabilities is the collection and analysis of Cyber Threat Intelligence (CTI) information. The MIG-S is a tool that provides a visual representation of CTI information extracted from the STIX vulnerability repository in relation to the intercepted authentication flow. This representation allows Security Testers to selectively examine CTI information at various levels of granularity, giving them greater flexibility in identifying vulnerabilities. For example, when assessing a system utilizing the SAML protocol, Security Testers are able to search for CTI information specific to the `RelayState` parameter or more broadly, for CTI information related to a SAML IdP. This can be particularly useful in identifying vulnerabilities that may be unique to certain protocols or systems. The combination of the MIG-S

tool with other security testing tools can facilitate the process of identifying vulnerabilities and conducting cyber risk assessments. This is particularly useful in identifying vulnerabilities that may not be easily detectable through other means.

In conclusion, the MIG-S is an important tool that can provide valuable insights into CTI information, and it can assist in identifying vulnerabilities and conducting cyber risk assessments. The ability to selectively examine CTI information at various levels of granularity provides Security Testers with greater flexibility in identifying vulnerabilities, in order to improve the security of a system.

4.5 MIG in DevSecOps

In recent years, the use of DevSecOps has become increasingly popular as a way to improve the security and reliability of software development pipelines. As part of this trend, the use of specialized tools to integrate security into the development process has also gained traction. In this section, we propose the design of a DevSecOps scenario that utilizes MIG as an integral part of the CI/CD pipeline.

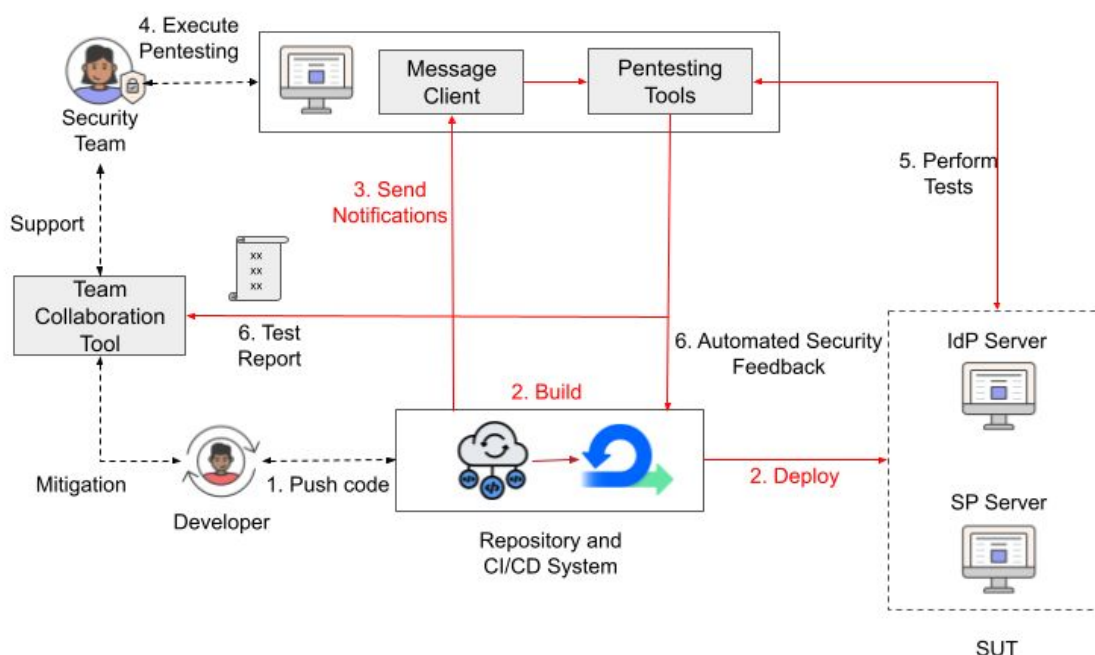


Figure 4.5: High Level Architecture of MIG in DevSecOps.

As depicted in Figure 4.5, 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 Client) and the CI/CD System and which was the starting point of a classic CI.

The latter 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 Security Team and the build and deployment phases will start.

The source code of the Client and IdP will be built, federated and both deployed in the SUT. The sent notification notifies the Security Team that the automatic penetration testing on the SUT has been executed and test reports are available.

The notification also contains some information needed to execute the MIG-T and to allow the communication between the repository, the CI/CD system, the SUT and the Pentesting Tools. Whether the Security Team decides to perform pentesting on the deployed solution, the MIG-T will automatically create security feedback and a test report with all the discovered vulnerabilities in a tool for team collaboration and accessible by both developers and Security Team. The tool for the team collaboration will be used by the developers to retrieve information about mitigations and by the Security Team 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 Software Development Life Cycle. The developers will have access to only a brief recap of the status of the vulnerabilities and mitigations while Security Team all the details about mitigations.

Chapter 5

Micro-Id-Gym Implementation

This chapter will provide an in-depth description of the implementation of MIG¹ and how it can be integrated into a DevSecOps scenario as an integral part of the CI/CD pipeline. The chapter will start by introducing MIG and its features, followed by a detailed explanation of the implementation process, including any necessary configurations and settings. The implementation of a DevSecOps scenario introduced in Section 4.5 that utilizes MIG will also be discussed, providing readers with a clear understanding of how to integrate MIG into their own development process and how it can improve the security of their software applications.

5.1 Overview

MIG supports two main activities: security testing of IdM protocol implementations and creating sandboxes with an IdM protocol deployment. The former consists of tools with a GUI to support security testing activities on SUT, namely a Proxy, a MIG-T, and two tools called MIG-D and MIG-S. 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 security testing activities that may cause severe disruptions such as Denial of Server (DoS) attacks.

Figure 5.1 illustrates a high-level overview of MIG, including the technology employed for each of its components. In the following sections, we give an in-depth explanation of how MIG was implemented. We will be covering the implementation of the MIG Backend, Dashboard, and MIG Frontend separately.

¹<https://github.com/stfbk/micro-id-gym>

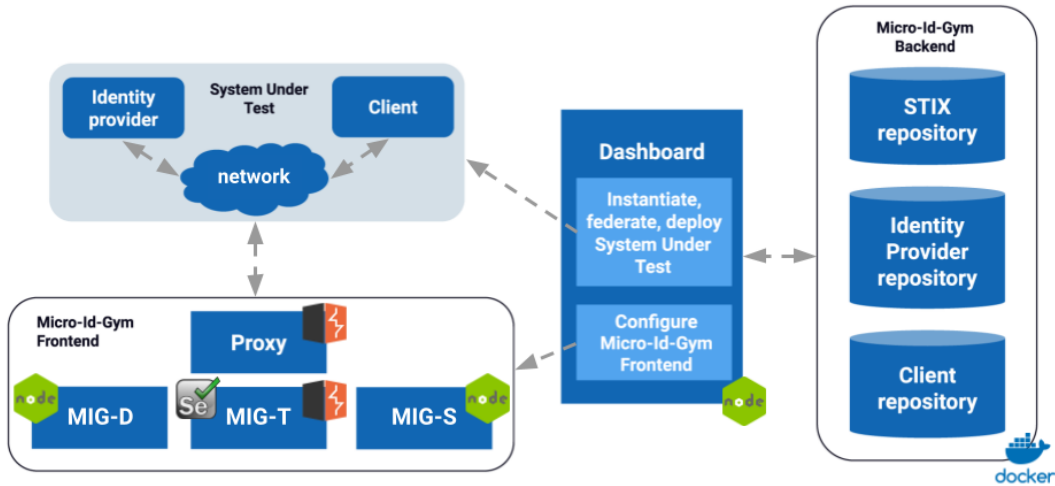


Figure 5.1: Details of technology used in MIG.

5.2 MIG Backend

The MIG Backend consists of repositories for Clients, IdPs, and MIG-S objects that adhere to the guidelines outlined in the MIG-S specification. In the upcoming sections, we will present the technical details of each.

5.2.1 Client and Identity Provider repositories

All the IdM instances presented in Section 4.2 are Docker-based and the currently available implementations are depicted in Table 5.1. For each instance, we report the version, the protocol running on that implementation and the technology used.

The container is generated from a `Dockerfile` where the steps for the initialization, the configuration and deployment are listed. The Listing 5.1 represents an example of the `Dockerfile` of a Client implementing SAML. In every `Dockerfile`, two important variables are set: `C_PORT` and `IDP_PORT`. These variables will contain the values of the ports where Client and IdP will run. Both are initialized using some arguments passed to the `Dockerfile` at the moment of the creation of the container.

These arguments are those preceded by the keyword `ARG` in Listing 5.1. In addition to Client and IdP ports, a third argument is passed to the `Dockerfile` which is `c_version`. Under

certain conditions, `c_version` specifies which version of the Client should be loaded in the container. However, on the contrary of `c_port` and `idp_port`, it is not always present. A similar configuration process has been also applied to OAuth/OIDC instances.

```
1 # load base image
2 FROM ubuntu:16.04
3
4 # initialize customized configuration parameters
5 ARG c_version
6 ARG c_port
7 ARG idp_port
8 ENV C_PORT=$c_port
9 ENV IDP_PORT=$idp_port
10
11 # install necessary packages
12 RUN apt-get update
13 RUN apt-get install -y default-jdk nano unzip curl
14
15 # copy chosen C implementation
16 COPY src/spring-security-saml-sp-1.0-$c_version.jar /spring-
    security-saml-sp-1.0.jar
17
18 # add metadata
19 RUN mkdir /metadata
20 COPY src/idp-metadata.xml /metadata
21
22 # customize metadata
23 RUN sed -i 's+localhost/idp/profile+localhost:$IDP_PORT/idp
    /profile+g' /metadata/idp-metadata.xml;
24
25 # deploy and run the C
26 CMD java -jar spring-security-saml-sp-1.0.jar --server.port=
    $C_PORT
```

Listing 5.1: Example of the Dockerfile of a Client implementing SAML.

5.2.2 STIX Repositories

The CTI information provided in the MIG has been semantically mapped to STIX objects according to the directives outlined in the STIX specification [JPD].

Table 5.1: Collection of Clients and IdP instances.

Version	Protocol	Technology	Provider
Base	S	Spring C	C
Supported c14n algorithm	S	Spring C	C
DTD enabled	S	Spring C	C
DTD disabled and c14nWithComments algorithm	S	Spring C	C
RelayState validation enabled	S	Spring C	C
RelayState validation disabled	S	Spring C	C
Sample Webapp	O	KeyCloak	C
Simple Webapp	O	MitreID	C
3.3.x	S	Shibboleth IdP	IdP
3.2.x	S	Shibboleth IdP	IdP
3.3.3 with RelayState sanitization enabled	S	Shibboleth IdP	IdP
1.3.3 without redirect_uri validation	O	MitreID	IdP
1.3.3 with redirect_uri validation	O	MitreID	IdP
10.0.1 without redirect_uri validation	O	Keycloak	IdP
10.0.1 with redirect_uri validation	O	Keycloak	IdP

Legenda: “S”: SAML, “O”: OAuth/OIDC

The four types of STIX objects used are:

- *Identity*: Representing actual individuals, organizations, or groups. For example, a university can be classified as an identity.
- *Vulnerability*: Describing weaknesses or defects in the requirements, designs, or implementations of the computational logic in software and hardware components that can be directly exploited.
- *Course of action*: Describing actions taken to prevent or respond to an attack, including technical and non-technical responses.
- *Attack pattern*: Describing ways that adversaries attempt to compromise targets, including specific attack methods such as spear phishing.

In the mapping process, vulnerabilities have been associated with the *Vulnerability* object type, threats with the *Attack pattern* object type, and mitigations with the *Course of action* object type. Each object contains a name, description, and external references for additional information. For vulnerabilities, additional attributes have been included to describe the previously defined class as outlined in Section 4.4.4. For threats, information about the risk level has been added. An example of a vulnerability encoded in the STIX format can be found in Listing 5.2.

```

1 {"type": "vulnerability",
2   "id": "vulnerability--cc39a29-ee9f-428c-a5a3-d48659c343dd",
3   "created": "2020-02-27T14:41:29.544Z",
4   "modified": "2020-02-27T14:41:29.544Z",
5   "name": "XML External Entity (XXE) processing",
6   "description": "XML Parser allows XML External Entities
7   inside SAML Messages and is wrongly configured to enable
8   large XML document consumption. ",
9   "labels": ["Web Browser SAML/SSO Profile", "SAMLRequest"],
10  "external_references": [{
11    "source_name": "XXE Cheat Sheet",
12    "url": "https://web-in-security.blogspot.it/2016/03/xxe-
13    cheat-sheet.html"}, {
14    "source_name": "Owasp XML External Entity",
15    "url": "https://www.owasp.org/index.php/
16    XML_External_Entity_(XXE)_Processing"}],
17  "x_FBK_vulnerabilityClass": "Missing XML Validation Class",
18  "x_FBK_vulnClass_description": "All the vulnerabilities
19  regarding a missing or an improper validation of the XML
20  message that may hide malicious content",
21  "x_FBK_provider": "IdP"}

```

Listing 5.2: Example of STIX JSON Object.

5.3 Dashboard

The Dashboard is designed as a component for establishing a local IdM instance within a sandbox and configuring the MIG-D and MIG-S. The Dashboard is a web application developed using NodeJS² that provides two main pages: a form for configuring the components and a page with instructions for setting up the MIG environment. The form displayed in Figure 5.2 enables the user to personalize the following settings:

- **Proxy port:** the port on which the Proxy runs.
- **MIG-D port:** the port on which the MIG-D runs.
- **MIG-S port:** the port on which the MIG-S runs.
- **Target:** whether the user is testing an IdM instances within a sandbox or a real system in the wild.

²<https://nodejs.org/>

- **Protocol:** the IdM protocol being analyzed.
- **Scenario name:** an identifier for the scenario being tested. The string entered in this field will be used as the `sessionId` for accessing the MSC in the MIG-D webapp.

If the User chooses to target an IdM instance within a sandbox, the user have the option to either test an instance that has been uploaded or select one from the instances available in MIG Backend. If the User opts to upload their own IdM instance the User must provide the following details:

The screenshot shows a web interface titled "Dashboard" with two main sections: "MIG Frontend configuration" and "IdM system configuration".

MIG Frontend configuration:

- Proxy port: 8080
- MSC Drawer port: 5000
- STIX Visualizer port: 5555
- Button: Upload additional tool

Target and Protocol:

- Target: Wild Sandbox
- Protocol: SAML (dropdown)
- Scenario name: saml-example
- Buttons: Create IdM Upload IdM

IdM system configuration:

- IdP version: Shibboleth 3.1.1 (dropdown)
- Idp port: 9443
- Client version: Spring SP Base (dropdown)
- Client port: 8001
- Buttons: XSS, CSRF
- Username: user
- Password: password
- Button: Download

Figure 5.2: Dashboard user interface.

- **Identity Provider:** select an IdP from the list of available instances.
- **Identity Provider port:** set the port on which the IdP runs.
- **Client:** select a Client from the list of available instances.
- **Client port:** set the port on which the Client runs.
- **Tester credentials:** insert customized credentials (username and password). The inserted credentials will be automatically added to the ones available to authenticate at the selected IdP.

The User can also upload their own IdM instance, which includes a pre-configured pair of IdP and Client. However, for compatibility with MIG, the uploaded IdM instance must adhere to a set of guidelines established by MIG.

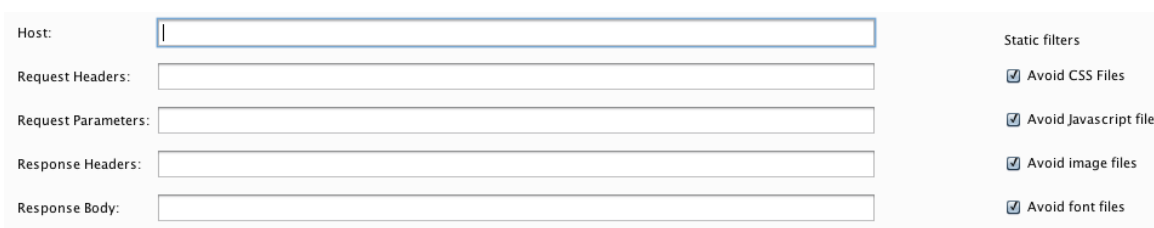
5.4 MIG Frontend

The MIG Frontend is made up of a Proxy, an MIG-D, a MIG-S, and MIG-T. In the following sections, we will provide technical details and the technologies used to develop each component.

5.4.1 Proxy

The Proxy adopted is Community Edition of Burp Suite (hereafter Burp)³, as it is compatible with all operating systems and offers a set of user-friendly APIs that aid in the development of security testing tools. The MSC Logger is an extension of Burp, utilized for communication with the MIG-D, utilizes the Burp's API to implement the features of *Intercept*, *Filter*, and *Parse* outlined in Section 4.4.1.

When a message is intercepted by the Proxy, the `processHttpRequestMessage` method from the Proxy API is activated. This method takes the intercepted raw HTTP Message as input, which is then examined and manipulated by MSC Logger. The raw data is then forwarded to the `filter` that has been implemented. The `filter` operates using a white-list approach, and makes use of the conditions specified by the User to determine whether the intercepted message should be rendered in the MIG-D. These conditions are specified using the input form depicted in Figure 5.3. Burp manages messages as pairs of requests and responses, so the user can specify conditions for the following fields: `hostname`, `request headers`, `request parameters`, `response headers` and, `response body`. For example, if the user enters *google* into the `hostname` field, all messages containing *google* in the `hostnames` will be rendered in the MIG-D.



Host:	<input type="text"/>	Static filters
Request Headers:	<input type="text"/>	<input checked="" type="checkbox"/> Avoid CSS Files
Request Parameters:	<input type="text"/>	<input checked="" type="checkbox"/> Avoid Javascript files
Response Headers:	<input type="text"/>	<input checked="" type="checkbox"/> Avoid image files
Response Body:	<input type="text"/>	<input checked="" type="checkbox"/> Avoid font files

Figure 5.3: Interface to add a new filter.

Before pushing data to the MIG-D, the intercepted message must be parsed and formatted appropriately if it meets the filter criteria. MSC Logger provides also the option to decode encoded parameters, such as `SAMLRequest` and `SAMLResponse`, through the use of an external service. Additionally, it is possible to define `aliases` to improve the readability of the MSC. For example, the User can replace the name of the IdP with a custom name to better understand the

³<https://portswigger.net/burp>

flow being represented. Once the selected parameters have been decoded and the aliases applied, MSC Logger formats the raw message and decoded data according to the MIG-D API specification (as detailed in Section 5.4.2) before pushing it to the MIG-D webapp. To facilitate the reuse of previously configured `filter` and `aliases`, MSC Logger allows for the export of the configuration in a JSON⁴ file that can be imported for future use.

In order for MSC Logger to process the configuration file, it must include the following parameters:

- **host**: The hostname where the MIG-D is running.
- **port**: The port where the MIG-D is running.
- **protocol**: The protocol supported by the MIG-D, which can be HTTP or HTTPS.
- **sessionID**: A string used to identify the MIG-D session where intercepted messages are represented in the message sequence chart.
- **readonly**: A property that can be used to make the drawn messages persistent. It is intended for sharing the message sequence chart among multiple users without the ability to modify it. The value must be set to `true` in this case.
- **filterItems**: This field contains an array of JSON Objects that describe the filters. Each JSON Object represents a filter entry and must be structured as follows:
 - **host**: A list of strings separated by a semicolon (;) that represent the values that need to be searched in the host of the intercepted message.
 - **request_header**: A list of strings separated by a semicolon (;) that represent the values that need to be searched in the headers of the intercepted request.
 - **request_params**: A list of strings separated by a semicolon (;) that represent the values that need to be searched in the parameters of the intercepted request.
 - **response_header**: A list of strings separated by a semicolon (;) that represent the values that need to be searched in the headers of the intercepted response.
 - **response_body**: A list of strings separated by a semicolon (;) that represent the values that need to be searched in the body of the intercepted response.
 - **draw**: Indicates which messages should be drawn by the MIG-D. 0 stands for both request and response, 1 for only the request, and 2 for only the response.

For example, if the `host` field contains the value `google; facebook`, MSC Logger will look for messages containing either `google` or `facebook` in the `host` field because the semicolon (;) represents the OR conjunction. Conversely, fields within the same JSON Object are evaluated jointly as if there is an AND conjunction between them. This means that if we add the string `token` to the `request_param` field in the previous filter, MSC Logger will

⁴<https://www.json.org/json-en.html>

match all messages containing either google or facebook in the `host` field and simultaneously having a `request parameter` containing the value `token`. Each JSON Object is evaluated independently, so each message is tested against all filters and if it matches one of them, it is drawn in the MIG-D webapp.

- **aliasItems**: it contains an array of JSON Objects that describe the aliases that can be applied to improve the readability of the MSC. For example, the user can decide to rename the `hostname` of the Client with a custom name. To do this, each alias entry must be a JSON Object structured as follows:
 - `target`: The string that needs to be replaced (i.e. Client).
 - `alias`: The string that replaces the target value (i.e. IdP).
- **parserItems**: it defines the external services used to decode encoded parameters. It contains an array of JSON Objects, each structured as follows:
 - `encoding`: An identifier of the encoding format the service can decode (e.g., `base64`).
 - `encodingURL`: The URL of the API endpoint providing the decoding service (e.g., `www.example.com/decode`).
 - `method`: The method used to call the API endpoint providing the service, which can be either GET or POST.
 - `paramName`: The name of the parameter where the API service expects the string that needs to be decoded (e.g., `inputValue`).

For instance, if we have the following values: `encoding` equal to `base64`, `encodingURL` equal to `www.example.com/decode`, `paramName` equal to `inputValue` and `method` equal to GET, this tells MSC Logger to decode in base64 strings by calling the API endpoint in the following manner: `GET www.example.com/decode?inputValue=StringToBeDecoded`

```

1 {"host": "localhost",
2   "port": "5000",
3   "protocol": "HTTP",
4   "sessionId": "test-logger",
5   "readonly": false,
6   "filterItems": [{
7     "response_header": "",
8     "request_header": "",
9     "host": "",
10    "request_params": "redirect_uri; code;",
11    "response_body": "",
12    "draw": 0}, {
13     "response_header": "",
14     "request_header": "",
15     "host": "",
16     "request_params": "",
17     "response_body": "code;",
18     "draw": 0}],
19  "aliasItems": [{
20    "target": "localhost:8888",
21    "alias": "Authorization Server"}],
22  "parserItems": [],
23  "parserMappingItems": []}

```

Listing 5.3: Example of MSC Logger configuration.

- **parserMappingItems**: it contains an array of JSON Objects that describe the parameters that need to be decoded and the corresponding decoding format to be used. For example, in a SAML deployment, we might find the mapping between the parameter SAMLRequest and the encoding format base64. Each JSON Object describing a mapping must be structured as follows:
 - **param**: The name of the parameter that needs to be decoded (i.e. SAMLRequest).
 - **encoding**: The encoding format that needs to be used (i.e. base64). Note that this field takes the value from the list of encoding formats defined in the *parserItems* field.

For new users, configuring MSC Logger through the user interface or a JSON file can be challenging. To simplify the process of creating a MSC for IdM protocol, MSC Logger includes pre-configured configuration files for OAuth/OIDC and SAML authentication flows. An example of the OAuth/OIDC configuration file is shown in Listing 5.3.

5.4.2 MIG MSC Drawer

The MIG-D is a web application built in NodeJS that is responsible for displaying the MSC of the flow intercepted by the MSC Logger. The MIG-D allows multiple MSC to be displayed simultaneously and is organized into sessions, each of which can be accessed using a user-defined `sessionId`. All messages added to the MIG-D web application are stored in a MongoDB-based database⁵. Given the lack of requirement for table joins and the need for a large number of objects in a consistent format, we decided to utilize a non-relational database that stores data in JSON-like documents. This approach is well-suited for integration with NodeJS and offers a fast and comprehensive solution that satisfies the requirements of the MIG-D. The web application offers two API endpoints to interact with the database. One endpoint is used to add a new message to the MSC of a session, and the other is used to delete all messages currently displayed in a session.

The MIG-D webapp has an API that allows messages to be added to a specific session by sending a POST request to the following URL: `http://MSC_DRAWER_HOST/api/message`, with the following body parameters:

- `sessionId`: the session identifier where the message will be displayed.
- `from`: the sender entity.
- `to`: the receiver entity.
- `label`: the text to be displayed on the arrow.
- `headers`: a JSON array of header objects, each with properties name and value.
- `params`: a JSON array of parameter objects, each with properties name, type (can be URL, Body, or Cookie), and value.
- `body`: the body of the message.

An example of a call to the API for adding a new message is reported in Listing 5.4.

⁵<https://www.mongodb.com/>

```

1 Method: POST
2 URL: http://localhost:5000/api/message
3 Body: {
4   sessionId: test,
5   from: fbk.eu,
6   to: User Agent,
7   label: 1. HTTP/1.1 302 fbk.eu/idp/profile/SAML2/Redirect/
   SSO,
8   headers: [{
9     name: HTTP/1.1,
10    value: 302 Found}, {
11    name: Date,
12    value: Tue, 04 Feb 2020 15:30:17 GMT}],
13  params: [{
14    name: RelayState,
15    type: URL,
16    value: cookie%3efrw1580830217_305f}, {
17    name: SAMLRequest,
18    type: URL,
19    value: <samlp:AuthnRequest>...</samlp:AuthnRequest>}],
20  body: ""}

```

Listing 5.4: Example of an API call for adding a new message to a session.

To delete all messages in a session, the API requires the following information:

- Method: DELETE.
- URL: `http://MSC_DRAWER_HOST/api/message/sessionId`.

Where `<sessionId>` is the session identifier from which all messages will be removed.

An example of using the API to delete all messages in a session is illustrated in Listing 5.5, showing the deletion of all messages in a session identified as `test`.

```

1 Method: DELETE
2 URL: http://localhost:5000/api/message/test


```

Listing 5.5: Deletion of all messages in the session test.

The process of adding or deleting messages to the MSC is handled automatically by MSC Logger, making it easy for the User to view the HTTP Message intercepted. To access the MSC, the User can visit the MIG-D webapp and log in using their chosen `sessionId` as shown in Fig-

ure 5.4. The `sessionId` is established when configuring MIG through the Dashboard. More details can be found in Section 5.3.

Please login to the MSC Drawer:

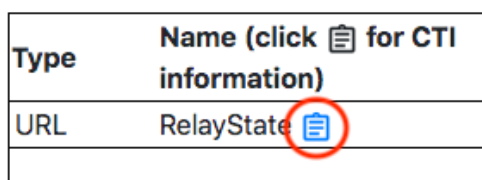


A login form with a text input field labeled "Password" and a blue "Login" button below it.

Figure 5.4: MIG-D login page.

To view the message sequence chart, the User must visit the MIG-D webapp homepage and insert the `sessionId`. The webapp uses the JS Sequence Diagrams library⁶ to convert the messages into figures. MIG-D also includes two additional features: (i) the ability to inspect the message content and, (ii) search for specific strings. The User can hover over an arrow to see a preview of the message content, or click on the arrow to view all the information about the message. The search feature allows users to look for specific strings in different fields such as the URL, headers, and parameters.

The User can access this information in different ways from the MIG-D. When the User visits the page displaying the details of a message, there is a button with the icon of a report beside each parameter as depicted in Figure 5.5. Clicking the button initiates a search for CTI information related to that parameter. If the search generates any results, they will be displayed by MIG-S as depicted in Figure 5.6. If not, an error message will be returned.



A table with two columns: "Type" and "Name (click [report icon] for CTI information)". The first row has "URL" in the "Type" column and "RelayState" in the "Name" column. A red circle highlights a report icon (a document with a checkmark) to the right of "RelayState".

Type	Name (click [report icon] for CTI information)
URL	RelayState [report icon]

Figure 5.5: Button to look up CTI information.

5.4.3 MIG STIX Visualizer

MIG-S is a web application that provides a graphical representation of MIG-S objects and the relationships between them.

The User can access CTI information about the overall intercepted HTTP Message or a specific entity by using the three red buttons located at the top of Figure 5.7. These buttons provide access to CTI information at different levels:

⁶<https://github.com/bramp/js-sequence-diagrams>

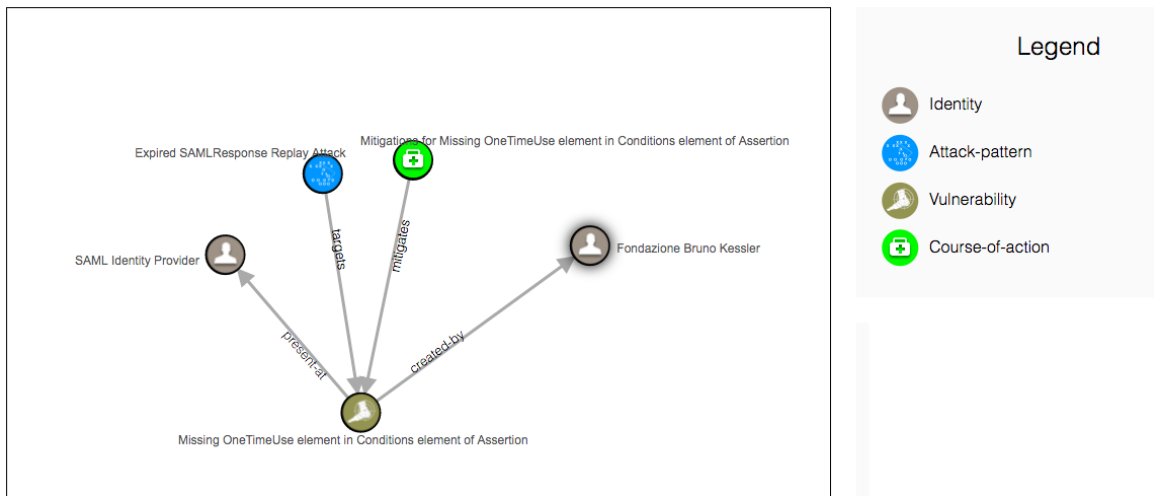


Figure 5.6: MIG-S search result.

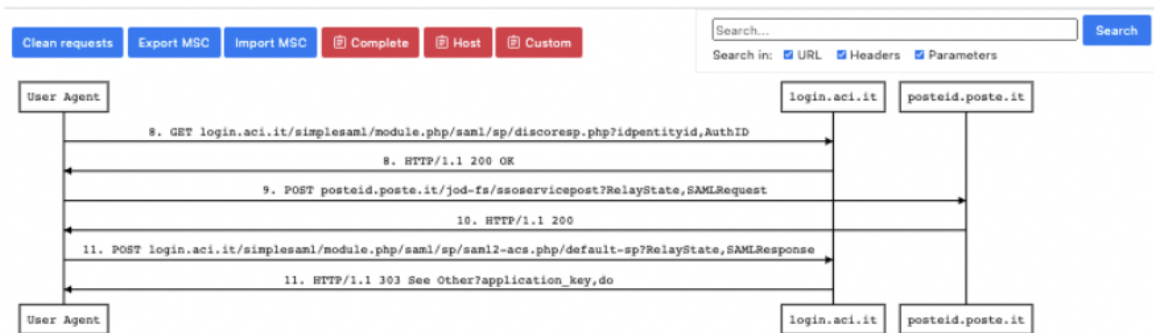


Figure 5.7: Example of an MSC draw by MIG-D.

- **Fully:** by clicking the `Complete` button, it retrieves the CTI information related to all the messages displayed.
- **Related:** by clicking the `Host` button, it retrieves CTI information related to the messages generated from a specific host involved in the authentication flow.
- **Custom:** by clicking the `Custom` button allows the User to retrieve custom CTI information based on a string input provided by the User.

MIG-S runs on NodeJS and uses the official library developed by OASIS to graphically represent CTI information⁷.

⁷<https://docs.oasis-open.org/cti/stix/v2.1/cs01/stix-v2.1-cs01.html>

5.4.4 MIG Tool

MIG-T is a security testing tool designed as an extension for Burp. It allows for easy inspection and manipulation of exchanged messages using Burp's APIs.

MIG-T is developed in Java and uses Burp's interface classes for interaction. Burp is typically used by connecting a browser to its proxy, allowing for the interception, viewing, and editing of packets. MIG-T automates this process, so the tester does not have to manually check or edit the messages.

Test name	Description	References	Statement in Ref. to Test	Affected Entity	Result
P1_a	Authentication Request: client_id parameter must be present in the ...	OIDC CIE Core Web, Section 5, Table 6	client_id parameter is required in the JWT token	OP	passed
P1_b	Authentication Request: code_challenge parameter must be present...	OIDC CIE Core Web, Section 5, Table 6	code_challenge parameter is required in the JWT token	OP	passed
P1_c	Authentication Request: code_challenge_method parameter must be...	OIDC CIE Core Web, Section 5, Table 6	code_challenge_method parameter is required in the JWT token	OP	passed

Figure 5.8: Test results example.

MIG-T's interface is shown in Figure 5.9. In the top left corner, there is an input area where users can paste the Session. To the right of this, there are several buttons that allow for different configurations:

- Use Chrome or Use Firefox buttons allows User to select the browser to be used.
- Select driver button allows the user to choose the driver that will be used to automate actions on the browser according to the browser that has been selected.
- Record button allows User to save the HTTP Message stored since now.
- Load messages button allows User to load previously saved messages for offline testing.
- Offline mode button allows the user to test previously loaded messages instead of live ones.
- Execute track button allows the User to run a session without executing any tests, which is useful when the original messages need to be saved.
- Test track button allows the User to test the Session without saving or executing any test.

The interface includes multiple tabs that can be accessed at the bottom, including:

- Input JSON tab includes an input area where User can input Test written in MIG-L. It also has three buttons located in the bottom right corner, including:
 - Stop button, which is used to stop the current execution.
 - Read JSON button, which is used to read and validate the syntax of the Test suite.
 - Execute Test Suite button, which is used to execute the reported tests.
- Test Suite Result tab contains the details of all the results of the executed tests.
- Test Result tab allows User to view the specific result of a test, including all the intercepted messages related to it.

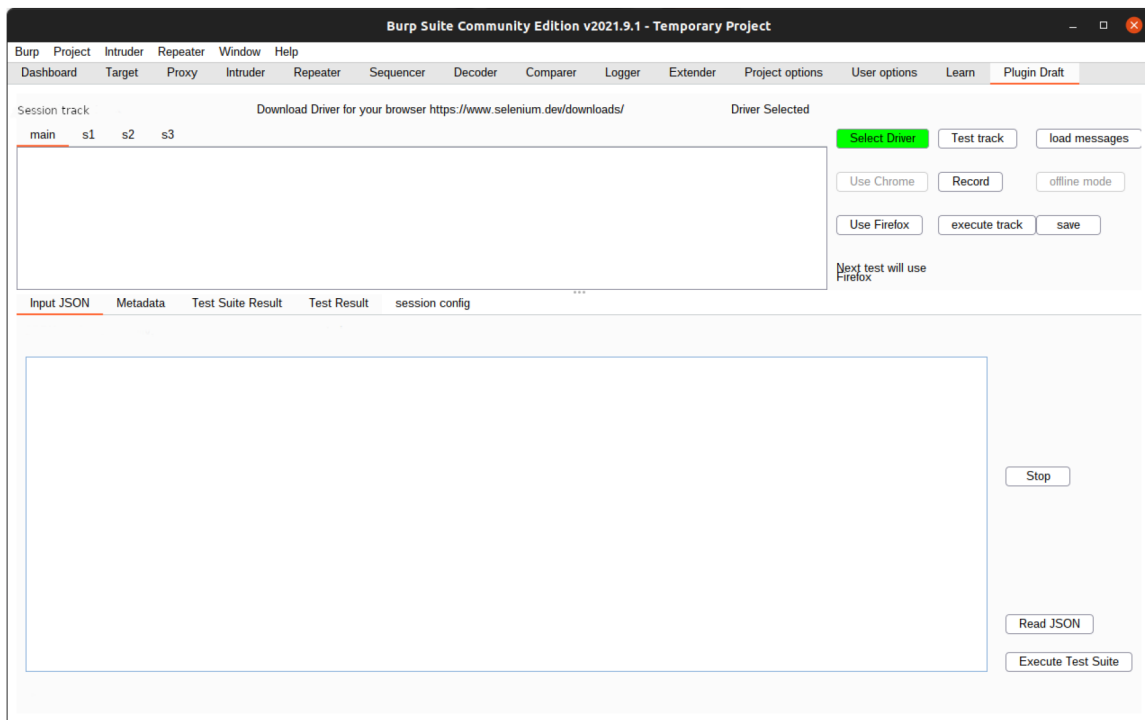


Figure 5.9: MIG-T user interface.

- Session Config tab allows User to configure the ports of the Session that will be used in the tests.

MIG-T independently manages each Session as a separate browser instance that is launched when a Session is started. Each Session corresponds to the tabs that are defined for it, and it runs in its own separate thread, allowing for concurrent execution of multiple Sessions. Additionally, each browser uses a unique proxy port, which allows for identifying and targeting specific Sessions during the testing process.

Regarding the Tests, they must be specified using the MIG-L declarative language defined in Chapter 3, which is a powerful language for describing the test scenario and its expected outcomes. To ensure that the Test is valid and accurate, it is important to follow the rules specified in the language and adhere to the correct syntax and structure when writing the Test. Concerning the Passive Tests, they can be completed after the messages are saved and the Test are then run on the saved HTTP Message while for the Active Test, since the HTTP Message need to be edited, the execution of the Session must be repeated for each Test.

5.5 Usage of MIG

To use MIG, it is necessary to have NodeJs, Java, a web browser (such as Chrome or Firefox), and Docker installed on the system. To download MIG, the user can clone the git repository by running the command `git clone https://github.com/stfbk/micro-id-gym/`. Once the framework has been downloaded, it can be used by opening a new terminal, navigating to the `dashboard` folder, and running the command `npm install` to install the necessary NodeJS packages. Then, the dashboard can be launched by using the command `node app` and visiting `localhost:2020` to configure the MIG tools. The Dashboard provides a configuration interface, where the `ports` for the SUT and the tools can be customized. Once the configuration is complete, the `Download scenario` and `tools` button can be clicked to generate a folder with the customized SUT and tools. The instructions provided by the webapp and available in the `README` file within the folder can then be followed to run the testing environment.

When the Dashboard homepage is loaded, the User has the option to use MIG in either a real-world scenario or a lab setting. To use the tool in a real-world scenario, the user must choose the option `wild` in the Dashboard. On the other hand, to use MIG in a lab setting, the user must choose the desired IdM protocol instance to be created by selecting one of the available IdP and Client provided by the MIG Backend. In both cases, the user will have access to the MIG-D, MIG-S, and MIG-T which are already set up and available for use.

5.6 MIG in DevSecOps

In this section, we will outline how we implemented MIG in the DevSecOps process. We adopted GitLab CI/CD⁸ as a tool to support software development and CI/CD.

In the root of the repository in GitLab CI/CD, there is a `configuration` file that serves as the foundation for creating a CD pipeline. This pipeline executes a series of jobs in distinct stages. The `configuration` file enables the setup of the pipeline and specifies the jobs that will run at each stage. By automating the process of releasing code changes, it ensures that the code is thoroughly tested and deployed in a consistent and reliable manner. To fully support the DevSecOps, we implemented the three operations written in red in Figure 4.5 by interpreting three stages in the pipeline: *(i)* Send Notifications, *(ii)* Build, and *(iii)* Deploy. The first stage is responsible for notifying the Security Team about changes in the repository by sending an email containing the Session and parameters required by GitLab to create Issues, namely the `Project Id` and `Host URL`. The second stage builds the source code of the IdM instance, and the third stage sets up the webserver and deploys the solution in the SUT. These jobs are executed by the

⁸<https://www.gitlab.com>

GitLab Runner agent, which is triggered by every push to the central repository.

Given the interoperability of MIG-T with any operating system we integrated it into our CI/CD solution and use it to perform security testing activities in the SUT. The tool automatically creates GitLab issues in the repository for any vulnerabilities discovered. To grant MIG read and write access to the GitLab Repository, a GitLab token, Project Id, and Host URL are required. The User must retrieve the GitLab token from his GitLab profile.

MIG will also generate a report of the test results, which will be automatically added to a Slack channel, a collaboration tool accessible by developers and Security Team.

Chapter 6

Experiments

This chapter focuses on evaluating the effectiveness of MIG-D in identifying vulnerabilities in IdM protocol implementation by conducting an experiment that highlights its precision. The chapter presents several use cases for IdM deployment, demonstrating how MIG can be used to perform security assessments conducted in various corporate scenarios. Additionally, we conducted security issues in the implementation of the Italian digital identity's CIE OIDC and SPID/CIE OIDC Federation deployments, a PSD2 service provided by an important Italian IdP based on OAuth, deployments based on SAML, and several SSO-based account linking procedures of online services.

6.1 User Validation of MIG Drawer

In the following sections, we present the results of our assessment of the environment for security testing and the effectiveness of the MIG-D in the laboratory experience. We create a realistic experience by incorporating vulnerable scenarios using MIG and evaluated the efficiency of MIG-D in identifying potential vulnerabilities.

6.1.1 Evaluation of the Effectiveness

Accurate detection of relevant vulnerabilities in the implementation of IdM protocols should be paired with effective presentation to enable security reviewers to work with ease when making informed decisions on protocol implementations. In this section, we focus on the effectiveness of MIG in presenting understandable results to Security Testers. This support their decision-intensive tasks through informative and easy-to-understand visualizations. To this aim, we will present and conduct a user study to empirically assess the *effectiveness* of the MIG-D component

of MIG in facilitating decision-intensive tasks of Security Testers, by comparing it with one of the most popular mainstream alternative security tools, namely OWASP ZAP¹.

A set of participants have been asked to play the role of a security analyst who performs security testing. In particular, participants are presented with an execution trace (visualized either with MIG-D or with OWASP ZAP) that was collected during the execution of a scenario that revealed a vulnerability on the implementation of an IdM protocol. Supported by these trace visualization tools, participants are asked to answer security related questions. Based on the ratio of correct answers, we estimated their level of comprehension of the trace and we related their security awareness about the IdM protocol implementation with the used trace visualization tool.

This experiment has been designed following the guidelines proposed by Wohlin et al. [Cla01].

6.1.2 Experimental definition and Context

To design this study, we formulate the following research question: *How does MIG-D compare to OWASP ZAP in supporting security awareness of analysts who conduct security testing on vulnerable implementations of IdM protocols?*

The *goal* of the study is to investigate the differences between MIG-D and OWASP ZAP, with the purpose of evaluating how well they support security awareness of Security Testers, when they are asked to review the implementation of IdM protocols. The *quality focus* regards how MIG-D and OWASP ZAP affect the capability of Security Testers to correctly understand the protocol execution traces. The results of the experiments are interpreted regarding two *perspectives*: (i) a researcher interested in empirically validating MIG-D and (ii) a security analyst who wants to understand which tool to adopt when security testing complex protocols such as those for IdM.

The *context* of the study consists of the *participants* involved in the experiment and the *software systems* to review.

Participants. We involved 42 participants in this study. They are 8 Bachelor and 34 Master students from the Department of Information Engineering and Computer Science of the University of Trento.

4 Bachelor students are attending the second year and 4 are attending the third year. They have quite a broad background on information security, because they attended (among others) the course *Introduction to Computer and Network Security*, that covered the OIDC protocol.

Most of the Master students (i.e, 25 of them) attend the first year and 9 of them attend the second year. They already attended other courses about security, including the course *Security Testing* that covered, in particular, attacks to OIDC implementations.

¹OWASP ZAP <https://www.zaproxy.org/>

Only a small minority of participants (i.e. 8 of them) reported no prior knowledge of authorization protocols such as OAuth, OIDC. Most of the participants (i.e. 29) reported an elementary knowledge and 5 of them reported a good knowledge. For this reason, we planned a training session to make sure that all the participants had a basic understanding of the protocol before the actual experiment. 36 participants already used a proxy tool to intercept and inspect network traffic, such as Burp Proxy, OWASP ZAP or Fiddler. 27 participants already used the OAuth protocol and 6 of them actually implemented such protocol.

We are aware that the expertise of students may be different than professionals. However, finding professionals available to conduct a demanding experiment as the one we designed is not easy. We mitigated this limitation by considering students with different levels of education and by making sure that participants had enough knowledge on OIDC protocols and its related vulnerabilities. All in all, the use of undergraduate students as a proxy of junior developers to draw conclusions is a common practice in empirical software engineering that is largely accepted and validated [Mar00, SAW08, SMJ15].

System. The software systems used in this study are two different deployments of the OIDC protocol:

- S_1 is an OIDC implementation that is vulnerable to a vulnerability related to a missing sanitization of the `redirect_uri` parameter.
- S_2 is an OIDC implementation that is vulnerable to a vulnerability related to a not adequate protection of the `state` parameter.

The selected systems are similar in terms of their complexity and in terms of the implemented IdM protocol. Moreover, their vulnerabilities are also comparable in terms of complexity and in terms of the operations required to be revealed. The attacks to exploit these vulnerabilities are described in Section 6.3.1. It is important to note that systems and vulnerabilities are representative of real world OIDC implementations.

Experiment Design and Procedure. We adopt a counter-balanced experimental design intended to fit two lab sessions. Participants are randomly assigned to four groups (despite they work alone), each one working in two labs on different systems with different tools, according to the schema summarized in Table 6.1. This design allows for experimenting with different combinations of *Systems* and *Tools* in different order across the two *Labs*.

Table 6.1: Experimental design.

	Group A	Group B	Group C	Group D
Lab 1	S_1 with TR _{mig}	S_2 with TR _{zap}	S_2 with TR _{mig}	S_1 with TR _{zap}
Lab 2	S_2 with TR _{zap}	S_1 with TR _{mig}	S_1 with TR _{zap}	S_2 with TR _{mig}

Before our experiment, participants were properly trained with lectures and exercises on OIDC protocol, on MIG-D and on OWASP ZAP, to cover and recall the required background. The

purpose of training is to make participants confident about the kind of tasks they are going to perform and the environment they will have available.

The pre-experiment profiling survey collects background knowledge about the participants, such as their previous experience with Proxy tool and their knowledge of OIDC protocol.

The experiment was carried out according to the following procedure. Participants had to (i) complete a pre-experiment profiling survey questionnaire, (ii) perform the comprehension tasks for their first system with their first tool (first lab), (iii) perform the comprehension tasks on their second tool with their second tool (second lab), and (iv) complete a post-experiment survey questionnaire.

To perform the experimental tasks, subjects received the following material:

- An initial questionnaire to gather data for grouping based on participants' knowledge.
- A document guide with theoretical concepts, explanations of vulnerabilities, and comprehension questions.
- Task instructions presented in a Google Form.
- Questionnaires that participants had to fill out in a Google Form.

According to the design, each participant has been involved in two experimental sessions (labs), each lasting approximately 2 hours. The comprehension task involved answering 9 questions about the assigned System, consisting of 2 closed-ended and 4 open-ended questions.

Comprehension tasks involve tasks that require the comprehension of specific information or concepts. These tasks are designed to simulate realistic scenarios, and can include tasks such as identifying parameters or searching for key URLs in an OAuth/OIDC flow. These tasks are based on real-world scenarios, and can help identify potential security issues if read and interpreted correctly. To answer these tasks, participants may need to utilize tools such as OWASP ZAP or read MSC from the MIG-D, and may be asked to provide information such as the URL of the request sent from the user-agent to the authorization endpoint. Visualizations of the task may be used to aid in answering these questions.

Most of the questions refer to realistic program understanding scenarios. In fact, the questions are designed to test the participant's ability to comprehend and analyze code in practical settings, such as identifying errors or vulnerabilities in software systems. By using real-world examples and scenarios, participants are challenged to apply their understanding of security testing concepts to practical problem-solving situations. These questions may require the use of security testing tools and techniques, and may involve analyzing code snippets or larger software systems to identify issues and suggest solutions. Overall, the goal of these questions is to assess the participant's ability to apply programming knowledge in practical settings. To answer the questions, participants had the possibility of looking at the document guide with theoretical concepts, explanations of vulnerabilities, and comprehension questions.

Post-experiment survey questionnaire (reported in Appendix A) deals with the clarity of the tasks,

cognitive effects of the treatments on the behavior of the participants and perceived usefulness of the trace visualization tools.

Variables Selection. The main factor of the experiment, that acts as an independent variable, is the particular trace visualization *Tool* used during the execution of the comprehension tasks. The *base case* TR_{zap} consists of using OWASP ZAP; and the treatment case TR_{mig} consists of using MIG-D, that includes not only the list of the HTTP intercepted messages, but also generates a MSC of the intercepted traffic.

Figure 6.1 depicts the visualization of OWASP ZAP while the Figure 6.2 shows the visualization of MIG-D.

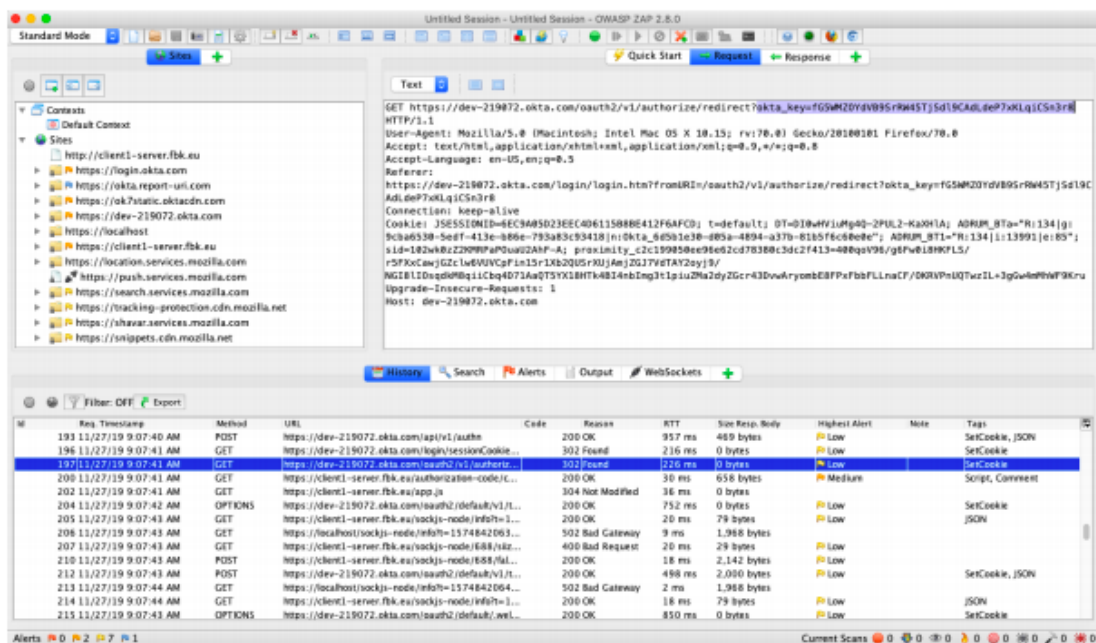


Figure 6.1: OWASP ZAP visualization.

The main outcome observed in the study was the level of security awareness. To evaluate it, the subjects were asked to answer some questions and the correctness of their answers was assessed. The evaluation was objective, as an URL reported was either correct or incorrect.

Other than the *Tool*, we can also measure the influence of other factors:

- *Lab*: whether there is any learning effect across consecutive labs, so that the experience gained working with the first tool in the first lab could improve the performance when working with the other tool in the second lab.
- *System*: if security assessment is intrinsically easier on a system than in the other system.
- *Question*: if there are specific questions that are harder to answer.

Table 6.3: Analysis of co-factors.

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.6357	0.0662	9.61	0.0000
Tool	-0.1345	0.0340	-3.96	0.0001
Lab	0.0402	0.0340	1.18	0.2369
System	-0.0693	0.0340	-2.04	0.0420
Question	0.0202	0.0065	3.09	0.0021

short). This consists of fitting a model of the *dependent* output variables (answer correctness) as a function of the *independent* input variables (all the factors, including the main factor, i.e. the used tool). A general linear model allows to test the statistical significance of the influence of all factors on the correctness and time of corrective tasks. Also for this test, we assume the threshold $\alpha = 0.05$.

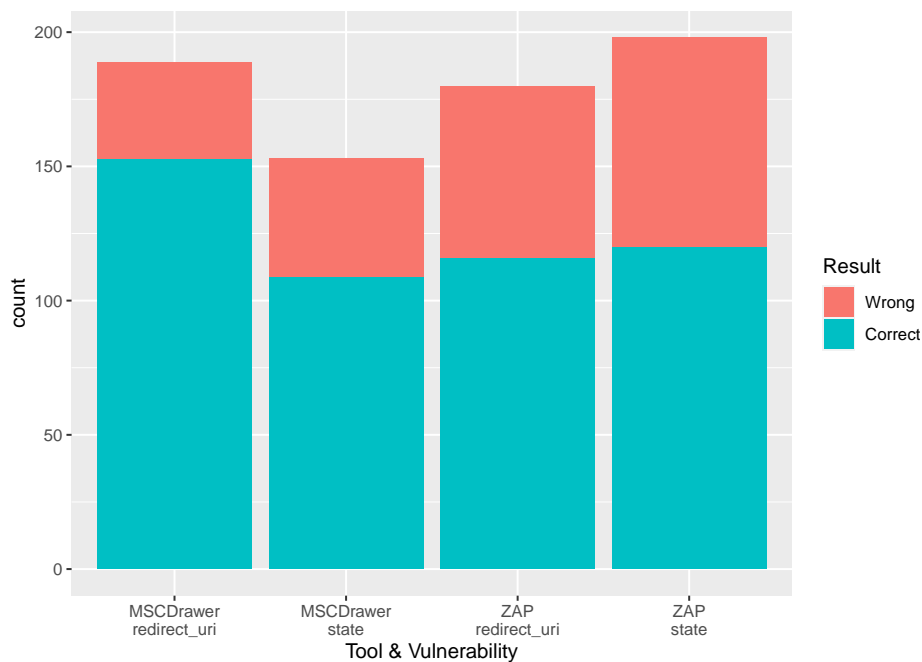


Figure 6.3: Number of correct and wrong answers by vulnerability and tool.

Table 6.3 shows the results of the GLM analysis, with p-values for statistical significant factors highlighted in boldface. First of all, we observe that the *Tool* has a insignificant impact on the correctness of answers, and this result is consistent with the previous analysis.

We can observe that also the *System* under analysis has a significant influence on the correctness of comprehension tasks. Figure 6.3 can be used to interpret this fact. When MIG-D is used

the system with the vulnerability related to *redirect_uri* seems to be easier to understand than the system with the vulnerability related to *state*. However, the two vulnerabilities look equally different when OWASP ZAP is used instead.

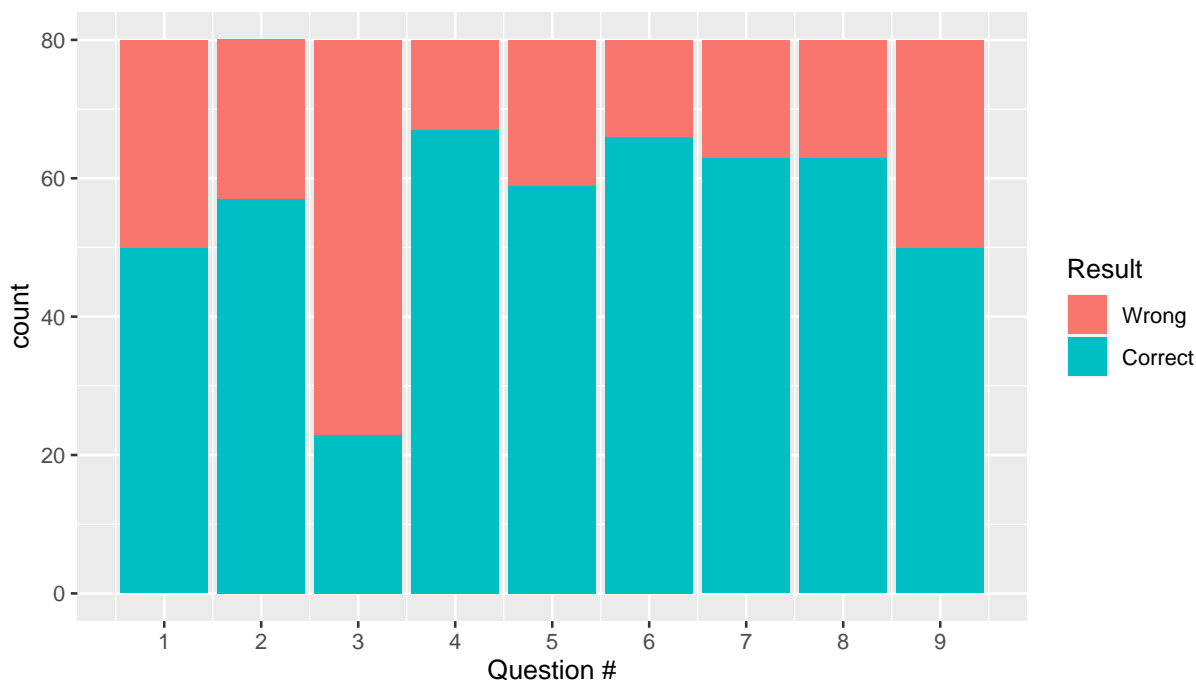


Figure 6.4: Number of correct and wrong answers by question.

The other significant factor is the *question* that was asked during the labs. Figure 6.4 visualize this trend. Question 3 seems to be the most hard to answer correctly. In fact, while the other questions just required a local understanding on a single message or message parameter in the trace, question 3 required to acquire a more global understanding of the trace, that involved multiple messages. For this reason, it was harder to answer question 3 correctly.

Feedback questionnaire: The answers to feedback questionnaire are reported on Table 6.4. While questions Q1-Q4 have been asked twice (once per lab), questions Q5-Q10 have been asked only once, at the end of the experiment.

Most of the participants consider that they have had enough time to complete the task (Q1), thus confirming the correctness of the experimental procedure. However, answers to Q2 support our speculation that security testing an IdP protocol is hard, in fact the majority of participants experienced difficulty on it. Additionally, the tools used to visualize the protocol traces were quite clear, because answers could be formulated in large majority using these tools, and only few participants had to look for information on other sources (Q4) e.g., on the Internet.

Most of the participants consider MIG-D the most useful tool to answer the comprehension tasks (Q5). Additionally, MIG-D was considered the most intuitive to use. OWASP ZAP, conversely, was considered the most difficult to use when finding relevant information (Q6).

When asked what tool would participants potentially use in their work (Q7), 4 participants preferred MIG-D and 4 participants preferred OWASP ZAP, while the majority might use both of them.

Most of the participants do not know alternative tools (Q8), few participants mentioned relevant alternative, such as Burp or Wireshark.

On question Q9, participants did not find any serious limitation on the compared tools, however they suggested to add the possibility to copy-paste from the GUI, because it could be useful to fill a security report. One participant suggested to support a textual *search* feature on the visualized trace.

On question Q10, participants suggested not to add more information to the visualization, because it would risk to compromise its simple and clean presentation. In fact, participant identified the main advantage of MIG-D as its simple and clear presentation of the protocol flow, that makes it easy to identify and go straight to the needed information.

Table 6.4: Answers to feedback questionnaire.

Question	Yes	No/Uncertain
Q1	66	12
Q2	25	53
Q4	6	72

Question	MIG-D	OWASP ZAP
Q5	32	4
Q6.a	32	4
Q6.b	5	31

Question	MIG-D	OWASP ZAP	Both	None
Q7	4	4	16	1

Question	No	Burp	Wireshark	(Others)
Q8	25	3	3	2

Considering these results, we can answer the initial research question as follows: *MIG-D is more effective than the state-of-the-art competitor OWASP ZAP in supporting security awareness of analysts who conduct security testing on vulnerable implementations of IdM protocols. In fact,*

participants who visualized execution traces with *MIG-D* delivered 262 correct answers and 80 wrong answers, while those who visualized traces with *OWASP ZAP* could deliver only 236 correct answers and 142 wrong answers.

Threats to Validity. The main threats to the validity of this experiment belong to the internal, construct, conclusion and external validity threat categories. We discuss each one of them in the following.

Internal validity threats concern external factors that may affect the independent variable. The chosen design allowed us to control a number of factors, namely participants background, system and learning across experimental sessions. Participants were not aware of the experimental hypotheses, participants were not rewarded for the participation in the experiment and they were not evaluated on their performance in doing the experiment.

Construct validity threats concern the relationship between theory and observation. They are mainly due to how we measure the correctness of tasks. We considered real vulnerabilities and we used a sound procedure to objectively evaluate whether the answers were correct.

Conclusion validity threats concern the relationship between treatment and outcome. We used statistical tests (Fisher exact test and General Linear Models) to draw our conclusions.

External validity concerns the generalization of the findings. In our experiments we considered two major vulnerabilities against IdM protocols, related to *redirect_uri* and *state* parameters. Although different vulnerabilities might occur, the results obtained with these vulnerabilities, already support well our interpretations.

Our experiment involved execution traces collected on real-world implementations of Identity Provider Protocols. Despite we consider that these implementations are representative of other Identity services, in principle different results could be obtained for different implementations.

The study was performed in an academic environment, which may differ substantially from an industrial setup. However, we mitigate this threat by using subjects with different seniority, including bachelor and master students.

6.2 Security Testing in the Lab using MIG

In the following sections, we present two use cases from our lab that demonstrate the usage of MIG in a controlled environment. In both scenarios, we performed an analysis of an OIDC deployment based on the Italian electronic identity card (CIE 3.0 - Carta d'Identità Elettronica) (CIE).² These use cases showcase the capability of MIG to effectively detect vulnerabilities in a secure and efficient manner.

²<https://www.cartaidentita.interno.gov.it/>

6.2.1 OIDC for CIE deployment

In a joint collaboration between FBK (acronym for “Fondazione Bruno Kessler”) [FBK] and IPZS (acronym for “Istituto Poligrafico e Zecca dello Stato”) [Pol], which is the Italian state printing office and mint, we have designed a web authentication mechanism based on the CIE [Min] and developed by a private company. The security analysis involves conducting a subset of tests, specifically those related to OIDC as outlined in Section 3.3.5.2, with the aim of evaluating the implementation of an OIDC Provider (OP) and a mocked RP provided by IPZS using MIG-T and determining their compliance with the specifications outlined in CIE OpenID Connect [Dev23]. The results of the tests provide valuable insights into the capabilities of the MIG-T and the implementation of the OP. These results are further described in detail. Overall, the tests were successful in providing an understanding of the performance and functionality of the MIG tool and the OP implementation, which will help guide future development efforts. We performed both Passive and Active tests on that scenario using the MIG-T tool.

The Figure 6.5 reports the Active Test results on the OP and RP obtained by running MIG-T. The following is a description of each column:

- **Test ID:** A unique identifier for each individual test case.
- **Description:** A brief description of the purpose or objective of the test.
- **References:** Any related documentation or resources that were used in the testing process.
- **Statement in the References:** The specific statement or requirement being tested from the referenced document.
- **Affected Entity:** The component or system that was tested and impacted by the test results.
- **Mitigations:** Any actions taken to address or resolve any issues discovered during the testing process.
- **Results:** The outcome of the test, including any observed behavior, failures, or pass conditions.

Test ...	Description	References	Statement in Ref. to Test	Affected E...	Mitigations	Result
T1_a	Authentication Request: Modify client_id in the URL	OIDC CIE Core Web, Section 5, Table 6	client_id must be an HTTPS URL that matches a value in the CIE registry	OP		passed
T1_b	Authentication Request: Remove client_id in the URL	OIDC CIE Core Web, Section 5	response_type and client_id SHOULD be passed using the OAuth 2.0 request parameter syntax	OP		passed
T1_c	Authentication Request: Modify response_type in the URL	OIDC CIE Core Web, Section 5, Table 6	response_type must be code	OP		passed
T1_d	Authentication Request: Remove response_type in the URL	OIDC CIE Core Web, Section 5	response_type and client_id SHOULD be passed using the OAuth 2.0 request parameter syntax	OP		passed
T1_e	Authentication Request: Modify scope in the URL	OIDC CIE Core Web, Section 5, Table 6	scope should be openid	OP		passed
T1_f	Authentication Request: Remove scope in the URL	OIDC CIE Core Web, Section 5	scope parameter MUST always be passed using the OAuth 2.0 request parameter syntax	OP		failed
T1_g	Authentication Request: Modify signature of JWT	OIDC CIE Core Web, Section 5	The request object must be a signed JWT.	OP		passed
T1_h	Authentication Request: Change client_id in the JWT	OIDC CIE Core Web, Section 5, Table 6	client_id must be an HTTPS URL that matches a value in the CIE registry	OP		passed
T1_i	Authentication Request: Change the signing alg of the JWT to none	OIDC CIE Core Web, Section 5	The request object must be a signed JWT.	OP		passed
T1_m	Authentication Request: Remove the signature of the JWT	OIDC CIE Core Web, Section 5	The request object must be a signed JWT.	OP		passed
T2_a	Authentication Response: Remove iss parameter	OIDC CIE Core Web, Section 6	The Authorization Response MUST return the parameters defined in Section 4.1.2 of OAuth 2.0 [RFC6749]	RP		passed
T2_b	Authentication Response: Modify iss parameter	OIDC CIE Core Web, Section 6	The RP MUST validate this parameter precisely and MUST NOT allow multiple OPs to use the same issuer identifier.	RP		failed
T2_c	Authentication Response: Modify state parameter	OIDC CIE Core Web, Section 6	A string with at least 32 alphanumeric characters that MUST be the same value given by the RP in the Authentication request.	RP		failed
T2_d	Authentication Response: Remove state parameter	OIDC CIE Core Web, Section 6	The Authorization Response MUST return the parameters defined in Section 4.1.2 of OAuth 2.0 [RFC6749]	RP		failed
T2_e	Authentication Response: Modify code parameter	OIDC CIE Core Web, Section 6	The obtained authorization code from the OPs Authorization Endpoint that will be used later within the Token Endpoint to exchange it for the OIDC tokens.	RP		passed
T2_f	Authentication Response: Remove code parameter	OIDC CIE Core Web, Section 6	The Authorization Response MUST return the parameters defined in Section 4.1.2 of OAuth 2.0 [RFC6749]	RP		passed

Figure 6.5: Active Test results of CIE OIDC deployment.

Concerning the OP test results, MIG-T reports that:

- The test T1_f failed because when the test removed the scope parameter in the Authentication Request as query parameter the OP did not return any error and the

authentication process continued successfully. The presence of the `scope` parameter in the `Authentication Request` as query parameter is cited in [Dev23], “*The parameter `scope` MUST be sent both as a parameter in the HTTP call, and inside the request object. The two values MUST be the same*”.

About the mocked RP test results, MIG-T reports that:

- The test `T2_b` failed because the RP must validate the `iss` parameter in the `Authentication Response`. The test edited the `iss` parameter with an arbitrary value and the RP accepted the modified `iss` value.
- The test `T2_c` failed because the RP did not check that the `state` parameter in the `Authentication Response` has the same value of the `Authentication Request` as reported in [Dev23]. The test changed the `state` parameter in the `Authentication Response` with an arbitrary value and the RP did not report an error in the authentication process.
- The test `T2_d` failed because as reported in [Dev23], the RP must validate the `state` query parameter in the `Authentication Response`. The test removed the `state` query parameter in the `Authentication Response` and the RP did not trigger an error.

In Figure 6.6, the results of Passive Test on both the OP and RP are presented, obtained through the execution of MIG-T.

Test	Description	References	Statement in Ref. to Test	Affected Entity	Mitigations	Result
P1_a	Authentication Request: client_id parameter must be...	OIDC CIE Core Web, Section 5, Table 6	client_id parameter is required in the JWT token	OP	OP should val...	passed
P1_b	Authentication Request: code_challenge parameter must be...	OIDC CIE Core Web, Section 5, Table 6	code_challenge parameter is required in the JWT token	OP		passed
P1_c	Authentication Request: code_challenge_method parameter must be...	OIDC CIE Core Web, Section 5, Table 6	code_challenge_method parameter is required in the JWT token	OP		passed
P1_d	Authentication Request: nonce parameter must be...	OIDC CIE Core Web, Section 5, Table 6	nonce parameter is required in the JWT token	OP		passed
P1_e	Authentication Request: prompt parameter must be...	OIDC CIE Core Web, Section 5, Table 6	prompt parameter is required in the JWT token	OP		passed
P1_f	Authentication Request: redirect_uri parameter must be...	OIDC CIE Core Web, Section 5, Table 6	redirect_uri parameter is required in the JWT token	OP		passed
P1_g	Authentication Request: response_type parameter must be...	OIDC CIE Core Web, Section 5, Table 6	response_type parameter is required in the JWT token	OP		passed
P1_h	Authentication Request: scope parameter must be...	OIDC CIE Core Web, Section 5, Table 6	scope parameter is required in the JWT token	OP		passed
P1_i	Authentication Request: acr_values parameter must be...	OIDC CIE Core Web, Section 5, Table 6	acr_values parameter is required in the JWT token	OP		passed
P1_j	Authentication Request: claims parameter must be...	OIDC CIE Core Web, Section 5, Table 6	claims parameter is required in the JWT token	OP		passed
P1_m	Authentication Request: state parameter must be...	OIDC CIE Core Web, Section 5, Table 6	state parameter is required in the JWT token	OP		passed
P1_n	Authentication Request: iss parameter must be present in...	OIDC CIE Core Web, Section 5, Table 6	iss parameter is required in the JWT token	OP		passed
P1_o	Authentication Request: aud parameter must be present in...	OIDC CIE Core Web, Section 5, Table 6	aud parameter is required in the JWT token	OP		passed
P1_p	Authentication Request: exp parameter must be present in...	OIDC CIE Core Web, Section 5, Table 6	exp parameter is required in the JWT token	OP		passed
P1_q	Authentication Request: iat parameter must be present in...	OIDC CIE Core Web, Section 5, Table 6	iat parameter is required in the JWT token	OP		passed
P1_r	Authentication Request: code_challenge_method parameter must be...	OIDC CIE Core Web, Section 5, Table 6	code_challenge_method parameter in the JWT token must be S256	OP		passed
P1_s	Authentication Request: nonce parameter must be at least 32 alphanumeric characters	OIDC CIE Core Web, Section 5, Table 6	nonce parameter in the JWT token should be at least of 32 alphanumeric characters	OP		passed
P1_t	Authentication Request: response_type parameter must be "code"	OIDC CIE Core Web, Section 5, Table 6	response_type parameter in the JWT token should be "code"	OP		passed
P1_u	Authentication Request: scope parameter in the J...	OIDC CIE Core Web, Section 5, Table 6	scope parameter in the JWT token should can be openid, offline_access, profile, email	OP		passed
P1_v	Authentication Request: acr_values parameter in the J...	OIDC CIE Core Web, Section 5, Table 6	acr_values parameter in the JWT token should be one of the supported	OP		failed
P1_z	Authentication Request: state parameter in the J...	OIDC CIE Core Web, Section 5, Table 6	state parameter in the JWT token should be at least of 32 alphanumeric characters	OP		failed
P1_k	Authentication Request: aud parameter in the JW...	OIDC CIE Core Web, Section 5, Table 6	aud parameter in the JWT token must be https://login.interno.gov.it/	OP		failed
P2_a	Authentication Response: code parameter must be...	OIDC CIE Core Web, Section 6	code parameter must be present in URL	RP		passed
P2_b	Authentication Response: state parameter must be...	OIDC CIE Core Web, Section 6	state parameter must be present in URL	RP		passed
P2_c	Authentication Response: iss parameter must be...	OIDC CIE Core Web, Section 6	iss parameter must be present in URL	RP		passed
P2_d	Authentication Response: state parameter must be...	OIDC CIE Core Web, Section 6	state parameter must be a string with at least 32 alphanumeric characters that MUST be the same value given by the RP in the Authentication request	RP		failed
P2_e	Authentication Response: iss parameter complian...	OIDC CIE Core Web, Section 6	iss parameter must be https://login.interno.gov.it/	RP		failed

Figure 6.6: Passive Test results of CIE OIDC deployment.

The test results for OP are presented as follows:

- The test `P1_v` failed but it was a false positive because the definition of the test was not correct. After fixing the test, the test result was passed.
- The test `P1_z` failed because the OP accepted a `state` parameter in the JWT token in the `Authentication Request` with the character “-”. The [Dev23] reports that the `state` parameter in the JWT token in the `Authentication Request` should be at least 32 alphanumeric characters.

- The test P1_k failed because the `aud` parameter in the JWT token in the `Authentication Request` is not equal to `https://login.interno.gov.it/` as reported in [Dev23]. The result is a false positive because the actual production differs from the one being analyzed and the `aud` value assumed has changed.

Concerning the RP, MIG-T reports:

- The test P2_d failed because the RP accepted a value of the `state` URL query parameter in the `Authentication Response` containing the character “-”. According to the [Dev23], the `state` URL query parameter in the `Authentication Response` should only contain alphanumeric characters.
- The test P2_e failed because the `iss` parameter in the `Authentication Response` did not match the expected value of `https://login.interno.gov.it/` as indicated in [Dev23]. This result is considered false positive because the value of the `iss` parameter was determined based on the analyzed OP.

6.2.2 OIDC for Developers Italia deployment

We conducted a security analysis of the SPID and CIE OIDC Federation,³ which are Django applications designed to simplify the construction of an OIDC Federation, in a collaborative effort between FBK and IPZS. These applications are offered by Developers Italia,⁴ a joint initiative between AgID (Agenzia per l’Italia Digitale)⁵ and the Italian Department for Digital Transformation (Dipartimento per la Trasformazione Digitale).⁶ The security analysis covered a set of tests, particularly those associated with OIDC as described in Section 3.3.5.2, with the aim of conducting a comprehensive assessment of the OP and RP implementation using MIG-T, and verifying their adherence to the specifications outlined in CIE OpenID Connect [Dev23]. The test results offer valuable insights into the capabilities of MIG-T. The purpose of this analysis was directed towards the `Authentication` endpoint, where Active and Passive tests were carried out. The Passive Test results are presented in Figure 6.8 and the Active Test results are depicted in Figure 6.7.

Figure 6.7 depicts the Active Test results on the OP and RP obtained by running MIG-T. Regarding the OP test results we noticed that:

- The tests T1_A, T1_B, T1_C, T1_D, T1_E and T1_F failed because the implementation is not reflecting the specification reported in [Dev23].

³<https://github.com/italia/spid-cie-oidc-django>

⁴<https://developers.italia.it/>

⁵<https://www.agid.gov.it/>

⁶<https://innovazione.gov.it/>

Test name	Description	References	Statement in Ref. to Test	Affected Entity	Result
T1_a	Authentication Request: Modify client_id in the URL	OIDC CIE Core Web, Section 5, Table 6	client_id must be an HTTPS URL that matches a value in the CIE registry	OP	failed
T1_b	Authentication Request: Remove client_id in the URL	OIDC CIE Core Web, Section 5	response_type and client_id SHOULD be passed using the OAuth 2.0 request parameter syntax	OP	failed
T1_c	Authentication Request: Modify response_type in the URL	OIDC CIE Core Web, Section 5, Table 6	response_type must be code	OP	failed
T1_d	Authentication Request: Remove response_type in the URL	OIDC CIE Core Web, Section 5	response_type and client_id SHOULD be passed using the OAuth 2.0 request parameter syntax	OP	failed
T1_e	Authentication Request: Modify scope in the URL	OIDC CIE Core Web, Section 5, Table 6	scope should be openid	OP	failed
T1_f	Authentication Request: Remove scope in the URL	OIDC CIE Core Web, Section 5	scope parameter MUST always be passed using the OAuth 2.0 request parameter syntax	OP	failed
T1_g	Authentication Request: Modify signature of JWT	OIDC CIE Core Web, Section 5	The request object must be a signed JWT.	OP	passed
T1_h	Authentication Request: Change client_id in the JWT	OIDC CIE Core Web, Section 5, Table 6	client_id must be an HTTPS URL that matches a value in the CIE registry	OP	passed
T1_i	Authentication Request: Change the signing alg of the JWT to none	OIDC CIE Core Web, Section 5	The request object must be a signed JWT.	OP	passed
T1_m	Authentication Request: Remove the signature of the JWT	OIDC CIE Core Web, Section 5	The request object must be a signed JWT.	OP	passed
T2_a	Authentication Response: Remove iss parameter	OIDC CIE Core Web, Section 6	The Authorization Response MUST return the parameters defined in Section 4.1.2 of OAuth 2.0.	RP	failed
T2_b	Authentication Response: Modify iss parameter	OIDC CIE Core Web, Section 6	The RP MUST validate this parameter precisely and MUST NOT allow multiple OPs to use the	RP	passed
T2_c	Authentication Response: Modify state parameter	OIDC CIE Core Web, Section 6	A string with at least 32 alphanumeric characters that MUST be the same value given by the R.	RP	passed
T2_d	Authentication Response: Remove state parameter	OIDC CIE Core Web, Section 6	The Authorization Response MUST return the parameters defined in Section 4.1.2 of OAuth 2.0.	RP	passed
T2_e	Authentication Response: Modify code parameter	OIDC CIE Core Web, Section 6	The obtained authorization code from the OPs Authorization Endpoint that will be used later wit.	RP	passed
T2_f	Authentication Response: Remove code parameter	OIDC CIE Core Web, Section 6	The Authorization Response MUST return the parameters defined in Section 4.1.2 of OAuth 2.0.	RP	passed

Figure 6.7: Active Test results of Developers Italia OIDC deployment.

- The tests T1_A, T1_B, T1_C and T1_D edited and removed `client_id` and `response_type` parameters and failed in all the cases because the OP did not return an error since the [Dev23] stated *“The parameters `client_id` and `response_type` SHOULD be sent both as parameters in the HTTP call, and inside the request object”*.
- The tests T1_E and T1_F verify the presence and correctness of the parameter `scope` in the URL stated as mandatory. The tests respectively edited the `scope` parameter with a random value and removed it. In both cases the OP did not return an error even if the [Dev23] stated *“The parameter `scope` MUST be sent both as a parameter in the HTTP call, and inside the request object. The two values MUST be the same”*. We noticed that the JWT request object suppresses the values contained in the URL.

Test name	Description	References	Statement in Ref. to Test	Affected Entity	Result
P1_a	Authentication Request: client_id parameter must be present in the	OIDC CIE Core Web, Section 5, Table 6	client_id parameter is required in the JWT token	OP	passed
P1_b	Authentication Request: code_challenge parameter must be present.	OIDC CIE Core Web, Section 5, Table 6	code_challenge parameter is required in the JWT token	OP	passed
P1_c	Authentication Request: code_challenge_method parameter must be	OIDC CIE Core Web, Section 5, Table 6	code_challenge_method parameter is required in the JWT token	OP	passed
P1_d	Authentication Request: nonce parameter must be present in the J...	OIDC CIE Core Web, Section 5, Table 6	nonce parameter is required in the JWT token	OP	passed
P1_e	Authentication Request: prompt parameter must be present in the J...	OIDC CIE Core Web, Section 5, Table 6	prompt parameter is required in the JWT token	OP	passed
P1_f	Authentication Request: redirect_uri parameter must be present in th...	OIDC CIE Core Web, Section 5, Table 6	redirect_uri parameter is required in the JWT token	OP	passed
P1_g	Authentication Request: response_type parameter must be present i...	OIDC CIE Core Web, Section 5, Table 6	response_type parameter is required in the JWT token	OP	passed
P1_h	Authentication Request: scope parameter must be present in the J...	OIDC CIE Core Web, Section 5, Table 6	scope parameter is required in the JWT token	OP	passed
P1_i	Authentication Request: acr_values parameter must be present in th...	OIDC CIE Core Web, Section 5, Table 6	acr_values parameter is required in the JWT token	OP	passed
P1_j	Authentication Request: claims parameter must be present in the J...	OIDC CIE Core Web, Section 5, Table 6	claims parameter is required in the JWT token	OP	passed
P1_m	Authentication Request: state parameter must be present in the JW...	OIDC CIE Core Web, Section 5, Table 6	state parameter is required in the JWT token	OP	passed
P1_n	Authentication Request: iss parameter must be present in the JWT t...	OIDC CIE Core Web, Section 5, Table 6	iss parameter is required in the JWT token	OP	passed
P1_o	Authentication Request: aud parameter must be present in the JWT t...	OIDC CIE Core Web, Section 5, Table 6	aud parameter is required in the JWT token	OP	passed
P1_p	Authentication Request: exp parameter must be present in the JWT t...	OIDC CIE Core Web, Section 5, Table 6	exp parameter is required in the JWT token	OP	failed
P1_q	Authentication Request: iat parameter must be present in the JWT t...	OIDC CIE Core Web, Section 5, Table 6	iat parameter is required in the JWT token	OP	passed
P1_r	Authentication Request: code_challenge_method parameter in the J...	OIDC CIE Core Web, Section 5, Table 6	code_challenge_method parameter in the JWT token must be S256	OP	passed
P1_s	Authentication Request: nonce parameter in the JWT token should b...	OIDC CIE Core Web, Section 5, Table 6	nonce parameter in the JWT token should be at least of 32 alphanumeric characters	OP	passed
P1_t	Authentication Request: response_type parameter in the JWT token ...	OIDC CIE Core Web, Section 5, Table 6	response_type parameter in the JWT token should be "code"	OP	passed
P1_u	Authentication Request: scope parameter in the JWT token should b...	OIDC CIE Core Web, Section 5, Table 6	scope parameter in the JWT token should be "openid"	OP	passed
P1_v	Authentication Request: acr_values parameter in the JWT token sho...	OIDC CIE Core Web, Section 5, Table 6	acr_values parameter in the JWT token should be one of the supported	OP	passed
P1_z	Authentication Request: state parameter in the JWT token should b...	OIDC CIE Core Web, Section 5, Table 6	state parameter in the JWT token should be at least of 32 alphanumeric characters	OP	passed
P1_k	Authentication Request: aud parameter in the JWT token must be ht...	OIDC CIE Core Web, Section 5, Table 6	aud parameter in the JWT token must be https://login.interno.gov.it/	OP	failed
P2_a	Authentication Response: code parameter must be present in URL	OIDC CIE Core Web, Section 6	code parameter must be present in URL	RP	passed
P2_b	Authentication Response: state parameter must be present in URL	OIDC CIE Core Web, Section 6	state parameter must be present in URL	RP	passed
P2_c	Authentication Response: iss parameter must be present in URL	OIDC CIE Core Web, Section 6	iss parameter must be present in URL	RP	passed
P2_d	Authentication Response: state parameter compliance	OIDC CIE Core Web, Section 6	state parameter must be a string with at least 32 alphanumeric characters that MUST be the s...	RP	passed
P2_e	Authentication Response: iss parameter compliance	OIDC CIE Core Web, Section 6	iss parameter must be https://login.interno.gov.it/	RP	failed

Figure 6.8: Passive Test results of Developers Italia OIDC deployment.

About the RP test result, one issue emerged:

- The test T2_A failed because the `iss` parameter has been removed from the Authentication Response and the RP did not return any error. During the testing activities, we found that editing the `iss` parameter in the Authentication

Response sent by the OP to the RP with an arbitrary value did not trigger an error, the RP did not correctly validate the `iss` parameter as reported in [Dev23]. The issue has been promptly fixed by Developers Italia in the release version 0.7.1 and an acknowledgment of the finding has been received as reported in Figure 6.9.

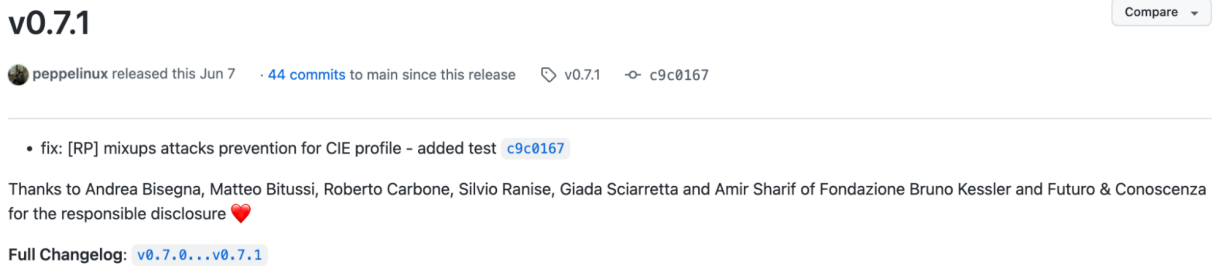


Figure 6.9: Acknowledgment received by Developers Italia.

6.3 Security Testing in the Wild using MIG

The following sections highlight three real-world examples of security testing in a production environment. The first scenario involves the evaluation of a PSD2 service provided by a prominent Italian IdP. The second scenario focuses on the analysis of SAML deployments. The third scenario demonstrates the detection of a CSRF vulnerability in an SSO-based account linking process. These use cases aim to evaluate the feasibility and efficacy of MIG as a means to assess the security and reliability of production systems.

6.3.1 OAuth for PSD2 deployment

In the context of an European project we had to assess a PSD2 service provided by an important Italian identity provider. PSD2 is the second Payment Services Directive, designed by the countries of the European Union [Off15]. Strong Customer Authentication (SCA) is one of the fundamental building blocks for building secure PSD2 deployments composed by Multi Factor Authentication and dynamic linking. The best solution to implement this scenario (such as proposed by many like the Berlin group) can be done by adopting OAuth which allows delegation to third parties to access resources. It could revolutionize the payments industry, affecting everything from the way we pay online, to what information we see when making a payment. Security of electronic payments is fundamental for ensuring the protection of users and all payment services offered electronically should be carried out in a secure manner, adopting technologies able to guarantee secure user authentication.

We conducted a security analysis performing a specific set of tests, which pertain to OAuth as described in Section 3.3.5.3, with the objective of assessing the implementation of the IdP and verify the compliance with the OAuth standard stated in [oaub]. To perform the security assessment, we leveraged MIG, and we were able to identify some vulnerabilities and a misconfiguration of the OAuth protocol. In particular, we followed the following two steps:

- Being the SUT an OAuth implementation, we set the MIG-D on the OAuth mode, in such a way to collect all the relevant traffic concerning OAuth. We have been thus able to spot very quickly the differences between the expected OAuth flow and the one shown by the MIG-D in terms of the intercepted HTTP messages. Thanks to MIG-D we were able to identify two HTTP messages exchanged between entities not encrypted using Transport Layer Security (TLS). The usage of HTTPS protects against man-in-the-middle attacks, eavesdropping and tampering. Thus this provides a reasonable assurance that one is communicating with the intended website without interference from attackers.
- Then, a finer-grained security analysis has been conducted by using MIG-T. The details of the tests automatically executed are reported in [Anda].

About the compliance, some required parameters according to the OAuth standard were not present in the protocol. On the contrary, some parameters which are not expected in the OAuth standard have been included in the protocol. In detail the tool detected that `redirect_uri` and `state` parameters were not set properly and both were not verified in different steps of the process by the IdP.

Regards the vulnerability in the `redirect_uri` parameter, AS, authorization endpoint, and client redirection endpoint can be improperly configured and operate as open redirects. An open redirect is an endpoint using a parameter to automatically redirect a user-agent to the location specified by the parameter value without any validation. Open redirects can be used in phishing attacks, or by an attacker to get end-users to visit malicious sites by using the URI authority component of a familiar and trusted destination. In addition, if AS allows Client to register only part of the redirection URI, an attacker can use an open redirect operated by Client to construct a redirection URI that will pass the authorization server validation but will send the authorization code or access token to an endpoint under the control of the attacker [Mik08].

Regards the `state` parameter, it is a recommended parameter in OAuth [LBLE]. It is an opaque value used by Client to maintain state between the request and callback. The AS includes this value when redirecting the user-agent back to the Client. The `state` parameter is set by Client in the Authorization Request, in step 1 in Figure 2.2, and it is checked by Client when Client receives it in step 3. If the integrity of the `state` parameter field is not adequately protected, it may allow hackers to mount CSRF attacks [Wan14]. To exploit these vulnerabilities an attacker causing the target browser to send the target site a request containing the attacker's own authorization code or access token. As a result, the target site might associate the attacker's protected resources with the target user's current session; possible undesirable effects could include saving user credit card details or other

sensitive user data to an attacker-controlled location.

These results pointed out the need of tools, like MIG-T, capable to support developers in order to properly implement, configure and test the OAuth implementations. Another interesting consideration emerges from our assessment. Some parameters were included in the protocol, meaning that we could detect them while checking the HTTP traffic. Yet, some of them were not used in end. For instance, the values of some parameters were not checked at all by the IdP. This highlights the need of active tests to check whether the parameters are indeed used properly.

All the discovered issues were notified to the company and fixed by their developers. In detail, about the HTTP messages exchanged not over TLS, the developers reviewed all the URL inside the application and modified the web server so that all messages will pass in HTTPS. Regards the missing sanitization of the `redirect_uri` parameter, the developers implemented a method inside the application to sanitize the parameter and do not allow any manipulation from an attacker. For the not adequately protection of the `state` parameter, the developers added some code to adequate protect the parameter. Lastly, about the redundant parameters not included in the protocol, the developers removed all of them from the flow.

6.3.2 SAML deployment

In the context of a collaboration with an important Italian IdP, we present the results of our analysis in this Section. Our study focused on 6 SAML-based scenarios, with 5 of them based on SPID. We involved two distinct IdP and six Client, utilizing MIG-T for the analysis. The results of our analysis are thoroughly outlined in this Section, providing clear insights into the security levels provided by the IdP and Clients. This study serves as a valuable resource for understanding the effectiveness of the security measures in place and identifying potential areas for improvement.

We carried out a security analysis that involved performing a specific set of tests related to SAML as described in Section 3.3.5.1. Our aim was to assess the implementation of both the IdP and Client by verifying their compliance with the SAML standard stated in [HPM] and identifying the presence of specific vulnerabilities. To perform the security assessment, we utilized MIG, which allowed us to identify several vulnerabilities.

In our analysis, we also identified some false positives, in particular:

- *Canonicalization form*: in all scenarios, the usage of a canonicalization algorithm that may lead to a canonicalization attack [Blo]. In all of the scenarios analyzed, an unsafe algorithm was used, but additional protections were in place so that they were not vulnerable.
- *NotOnOrAfter*: in a specific scenario the `NotOnOrAfter` test failed even if, examining the message, the attribute was set to the recommended value and was found both in the `SubjectConfirmationData` element and in the `Conditions` element, so no risk of

Replay Attack [Avi16] has been detected.

- *OneTimeUse*: all scenarios results vulnerable to a Replay Attack because of the failed `OneTimeUse` test [Avi16]. However the browser Session Protection's active test showed that other protections for this vulnerability were involved.
- *RelayState protection*: in a particular scenario, a vulnerability to RelayState tampering was found due to the absence of the `RelayState` element. Upon testing this vulnerability, we were able to alter the value of the `RelayState` in the request. The response included the manipulated value without any error, but the authentication process still followed the original path, indicating that additional measures were in place to disregard the tampered value [And19].
- *Signature*: according to the guidelines stated in the SAML core document [Con08], all messages transmitted through insecure channels (e.g., the user's browser) must be signed, while messages that pass through secure channels (such as SSL/TLS) do not have to be signed. To demonstrate this, a XML Signature Exclusion test was conducted and the scenario was found to be not vulnerable.

The experimental analysis has led also to important results and revealed the following attacks:

- *Certificate Faking attack resulting in Impersonation attack* [Vla]: this attack consists in removing the `Signature` element in the SAML `Response`, modifying some user's personal information, signing the assertion again with a locally generated private key and applying a self-signed certificate derived from the original. The modified SAML `Response` is then sent by the browser to the Client. If the Client does not carry out the checks described above, the attacker is authenticated as the victim on the Client. A Client that behaves in this way does not verify correctly the digital signature that guarantees the integrity and authenticity of the SAML `Response`, therefore being vulnerable to an attack of Impersonation type. In relation to the severity and the type of breaches that can be made by changing personal information, a potential *very high* risk is estimated.
- *SAML Response Cross Site Request Forgery*: this attack consists in intercepting and copying the SAML `Response`, dropping the request and closing the browser [Avi16]. In a new browser session we have to create another request and replace it with the replayed message previously intercepted. So the tester is logged into his/her account in a new browser session, unrelated to the original. It is important to notice that the replayed message can be forwarded with or without cookies, the result is the same. In a real world scenario, the victim may believe they are using their own account when they are actually using the attacker's.

We encountered some issues in the crucial task of responsible disclosure of discovered vulnerabilities, particularly in regards to its complexity in Italy.

Unlike overseas realities, although reporting vulnerability is considered a good rule, Italian laws do not include responsible disclosure, indeed it is often ethical hackers who are condemned for abusive access to computer systems. The commitment of American government companies

in the field of vulnerability research (also thanks to *Bug Bounty Programs*), demonstrates the importance of keeping high attention on cyber-security in all areas that are increasingly subject to attacks [Age19].

Some of the stakeholders in the Italian digital transformation firmly believe in responsible disclosure as the primary tool for communication with ethical hackers, particularly when the security, privacy, and personal information of citizens are at risk. A responsible disclosure program can also facilitate the prompt resolution of security issues with the goal of minimizing risks. Timely resolution of vulnerabilities is crucial for reducing exposure to malicious attackers [BV16].

Even if AgID reserved an important place to the cataloguing of IT vulnerabilities through the implementation of the *National Vulnerability Database* financed by the *Triennial Plan for IT 2019-2021*, in Italy much progress is yet to be made in this area, primarily by developing a critical and result-oriented approach and secondly by regularizing a national responsible disclosure policy.

6.3.3 SSO-based account linking process

This analysis, derived from a collaboration with SAP and Meta, focuses on the SSOLinking and on an overlooked CSRF attack vector for SSO which was introduced by Rich Lundeen at the BlackHat conference in 2013 [Lun]. SSOLinking allows users to link their accounts at Client websites, to their IdP account. This enables them to authenticate at Clients through an IdP account, thereby eliminating the need to maintain separate set of credentials for each Client. This added advantage encourages Client to support the SSOLinking process.

```

1  define_suite(SSOLinking Testsuite, Testsuite to verify the
    vulnerability in SSOLinking, true) // Testsuite defines
2  start_test(SSOLinking Account Hijack, Test to mount the
    SSOLinking Account Hijack attack, active, s1, s2, correct
    flow s2) // Test defines
3  start($s1) // Session (s1) runs
4  // Inference begins
5  start_intercept($C_acc, s1, forward)
6  start_msg_operations()
7  mark($L, <IdP_redirection>, s1)
8  save($C_login_act, [1, index($L)], s1)
9  save(<IdP_redirection>, $C_acc)
10 gen_poc($C_acc, $POC)
11 mark($A, contains{assert, @assertion})
12 save($C_assert, $A)
13 end_msg_operations()
14 start_session_operations()
15 add($C_login_act, $s2)
16 add(instance_credentials(<IdP_login_act>), $s2)
17 add("open | http://localhost/" + $POC, $s2)
18 add("click | id=SSOLinking", $s2)
19 add("wait | 2000", $s2)
20 add($C_assert, $s2)
21 end_session_operations()
22 end_intercept()
23 // Inference ends
24 start($s2) // Session (s2) runs
25 end_test() // Test ends

```

Listing 6.1: Test procedure for SSOLinking Account Hijack with inference procedure.

Exploiting the SSOLinking Account Hijack relies on two vulnerabilities: a Authentication CSRF on the IdP side and a SSOLinking-Init-CSRF on the SSO-based account linking process on the Client side.

The Authentication CSRF has been studied in various works [Ada08, Avi17, Mic21] and many of such vulnerabilities have been reported, also to IdPs. However, in most of the cases the IdP opted to not fix such a vulnerability to allow users to make use of the one-click login feature. If Authentication CSRF vulnerabilities are not fixed, then investing further effort in developing testing techniques for this is not so worth. We validated these two hypothesis—(I1) IdPs tend to be vulnerable to Authentication CSRF and (I2) IdPs tend to be reluctant to fix these

vulnerabilities—in a specific experiment over the four major IdPs identified in [Moh18]. The results of this experiment are detailed in Section 6.3.3.8 and confirm our hypothesis.

In our approach we thus assume that websites and, more specifically, IdPs can be vulnerable to Authentication CSRF. Under this assumption we design a testing technique to detect the SSOLinking Account Hijack on the Client side.

While the Session (s1) reported in Section 3.3.1 is executed, MIG-T infers the attack steps to be run to detect the SSOLinking Account Hijack at a specific Client and saves them in a new Session (s2). Listing 6.1 reports the Test written in MIG-L to perform the entire SSOLinking Account Hijack at any Client: commands are presented in red, macros in purple, and comments in blue.

The first `define_suite` action just indicates that the Test suite (SSOLinking Test suite) is executed. Then, the procedure starts and proceeds as following line by line:

Line 2. The Test (SSOLinking Account Hijack) is defined and proceed to execute the following operations.

Line 3. The first `start` action just indicates that the Session (s1) is executed.

Line 7. At first, the inference procedure observes the HTTP messages and marks with `L` the user action of (s1) whose corresponding HTTP messages comprise the value of the macro `<IdP_redirection>`. The macro is extracted from the IdPs metadata database that is described in the next section.

Line 8. All the user actions in (s1) from the first till the one marked with `L` are saved into the variable `$C_login_act`. Note that closing round bracket indicates that the action marked with `L` is excluded. All these actions capture the Client login activity.

Lines 9-10. The first HTTP message request whose HTTP response comprises the macro `<IdP_redirection>` is saved into variable `$C_acc` and then an HTML POC is generated by the “HTML POC Generator” module (details provided below) to enable later on the sending of a cross-site request to that URL. The path to the HTML POC is saved into `$POC`.

Lines 11-12. All user actions in (s1) that contain any of the keywords `assert` or `@assertion` as a comment are marked with `A` and then saved in variable `$C_assert`.

Lines 15-20. The new Session (s2) is now built. First, all the user actions related to Client login (saved into `$C_login_act`) are added. Then, the user actions related to IdP login that are stored in the IdP metadata as macro `<IdP_login_act>` are added upon proper instantiation of the testing user credentials at the IdP. Indeed, `<IdP_login_act>` comprises itself two macros about the user credentials that are automatically instantiated by fetching the real values from (s1). Then three actions are added into (s2): the first will open the HTML POC local page (notice that the path depends on `$POC`), the second will run the cross-site request, and the third just add some waiting time to ensure the request is properly processed. Last, but not least, the

assertions collected in `$C_assert` are added to (s2).

Now that the inference is completed and (s2) is created, the test procedure simply proceeds in running (s2) in line 16 and inspecting the results of that execution in line 17 to output the verdict to the tester.

6.3.3.1 IdPs metadata

Our approach relies on few IdP-related metadata. This is an offline activity that requires little effort and can be shared among few researchers to collect metadata for many IdPs. The procedure to support new IdPs is straightforward and amount to append in the IdPs metadata database the following two IdP information.

IdP Redirection signature. This is the signature that our approach expects to find in the redirection response that the Client generates to reach a specific IdP. In most of the cases, this metadata entry will be something like `v2/dialog/oauth` or `v1/dialog/oauth`. It is referred as macro `<IdP_redirection>` in Listing 6.1.

IdP Login. This comprises the list of user actions to perform the login at the IdP. It is referred as macro `<IdP_login_act>` in Listing 6.1. The user credentials are not hardcoded, but they are rather specified as macros that will be instantiated by our approach as in Listing 6.1 by fetching the right credentials for the IdP from (s1).

6.3.3.2 Generate the HTML POC

The aim of the operation `gen_poc` is to generate a proof-of-concept to exploit the SSOLinking for a given request to be processed during the execution of s2 and to provide to the Client to verify the vulnerability. In details, it creates a HTML page with the HTML form to send the HTTP request in `$C_acc` properly changed. Indeed, according to the HTTP method of `$C_acc`, the module builds the page and the HTML form. In case of a GET message, only the url has been sent in the form while for a POST message the content-type, url and body are sent.

6.3.3.3 Output

The output of our testing approach comprises two elements. The first element is the test result that provides the verdict whether the attack was successful or not. In rare cases our approach can return a not applicable result when one or more conditions built-in in our commands are not satisfied. For instance, if the marking operation in line 3 of Listing 6.1 is not finding the `<IdP_redirection>`, then the process is simply not executed as expected. If all the user actions in (s1) and those created for (s2) are successfully executed, then the attack is successful.

If some user actions of (s2) were unsuccessful (e.g., an unexpected page is displayed, an element which should be clicked is not present in the page), then the attack is unsuccessful.

The second element of the output is the HTML POC useful to reproduce the attack (when the attack was reported). The last element is a log file with all the HTTP messages saved during the execution of the test.

6.3.3.4 Experimental analysis

We demonstrated the effectiveness of our approach and the pervasiveness of the SSOLinking Account Hijack against a selection of 40 popular websites featuring the SSO-based account linking process with major IdPs. In doing so our approach discovered 19 vulnerable web sites over 40 analyzed. In the rest of this section, we detail the selection of the major IdPs and popular websites (see Section 6.3.3.5) we used in our experiments, the methodology used for our experiments (see Section 6.3.3.6), and the results obtained (see Section 6.3.3.7).

6.3.3.5 Dataset selection

We selected 4 major IdPs and for each of them 10 Clients supporting the SSO-based account linking process. Our selection leverages on the dataset released by [Moh18] in 2018, where the authors crawled the most popular one million websites to infer whether they integrate the SSO login process. This is very valuable information for our study as websites featuring the the SSO-based account linking process are clearly a subset of those having SSO login. Even if some of the data from [Moh18] is clearly outdated, this did not impact our study and we were able to select our candidates easily.

Figure 6.10 reports an example of an entry of the dataset from [Moh18]. Each entry specifies in the field `sso` which IdPs that website is using (if any). For this example `workable.com` is inferred to have SSO login with Google and LinkedIn IdPs.

```
{"rank": "4824", "url":  
  ["https://www.workable.com/signin",  
   "http://workable.com",  
   "https://www.workable.com/signup",  
   "https://www.workable.com/"], "sso":  
  ["google", "linkedin"], "redir_from": []}
```

Figure 6.10: Entry (excerpt) from [Moh18].

For each IdP mentioned in the dataset we counted how many websites were making usage of them in the 2018 dataset and selected the 4 IdPs with the highest number of occurrences.

We targeted the first 6 IdPs in order of appearance in the 2018 dataset and then we counted how many websites were having SSO login with them. We selected the four IdPs used by the highest number of websites. Table 6.5 presents the IdPs candidates and the number of occurrences reported for each of them. The selected IdPs for our experimental analysis are then Facebook, Google, Twitter and LinkedIn.

Table 6.5: IdPs candidates.

IdP Name	Occurrences
Facebook	43,333
Google	26,186
Twitter	12,465
LinkedIn	3,939
Amazon	1,459
Yahoo	1,924

For each one of the selected IdP, we then proceeded in selecting 10 Clients having SSO-based account linking with the IdP. We considered one by one the website entries in their ranking order from the 2018 dataset for which one of our selected IdP occurred. We manually checked whether the SSO-based account linking process was implemented by that website with the IdP. As shown in Table 6.6 we manually analyzed a corpus of 462 Clients to discover 59 websites supporting the SSO-based account linking process with the selected IdPs. Among the 59 Clients, we discarded 19 websites due to web issues that occurred in the execution of the process (15) and automation detection issues (4). With web issues, we refer to issues in the Client website that leads to e.g., broken pages and links during the account linking or unlinking processes. For automation detection issues we mean those Clients that e.g., identify the use of Selenium libraries to automate browser actions as a bot issue, which denies access to the website. The Clients selection procedure is successfully finished by obtaining 10 fully working Clients for each selected IdP. Some of these Clients are used in our dataset for more than one IdP. For instance, the Nicovideo website occurs in our dataset as paired with Facebook, Google, and Twitter IdPs. The number of unique Clients is 16.

6.3.3.6 Methodology

Here we describe the methodology we followed to perform the experiments on our selected dataset.

We manually analyzed the four IdPs in our dataset against the Authentication CSRF and discovered that three of them are vulnerable and that they do not plan to fix the problem. More details are given in Section 6.3.3.8.

Table 6.6: Clients selection per IdP.

	Clients per IdP	Clients with SSOLinking	Clients with issues	
			Web	Autom.
Facebook	59	16	4	2
Google	117	10	0	0
Twitter	35	11	1	0
Linkedin	251	22	9	3
Total	462	59	14	5

While analyzing the IdPs we also collected the few IdPs metadata required for our approach. Listing 6.2 provides a concrete example of the IdP metadata for Facebook.

Note that the user credentials are specified as macros that are automatically instantiated by our approach for the specific testing user and Client. We make available the other IdPs metadata for Google, Twitter and Linkedin in [Andb].

```

1 {"facebook": {
2   "<IdP_redirection>": "dialog/oauth/?\?\?",
3   "<IdP_login_act>": [
4     "open | https://facebook.com/login.php",
5     "click | xpath=/html/.../button[2] |",
6     "click | id=email |", "type | id=email | <email>",
7     "click | id=pass |", "type | id=pass | <password>",
8     "click | id=loginbutton |"]}

```

Listing 6.2: IdP metadata for Facebook.

The last step in our methodology aims to evaluate the pervasiveness of SSOLinking Account Hijack. In this respect, we run our MIG-T against the 40 pairs of Clients and IdPs in our dataset. For each pair of Client and IdP, we played the role of a tester at the Client and we created the Selenium script to execute the SSO-based account linking process. In doing so we followed the steps described in Section 3.3.1. All the scripts are available in [Andb]. Our MIG-T detected the account hijack in almost 50% of our dataset. More details are given in Section 6.3.3.9, while responsible disclosures and confirmations from vulnerable Clients are discussed later. All in all, our results indicate that this attack may really be overlooked by the web community.

Ethical considerations and disclosure. We make sure that our tests did not cause any harm to the tested websites. For instance, we neither injected any code in the HTTP requests nor tried to have unauthorized access to user accounts that are not under our control. All tests were

performed using test accounts created on the websites by ourselves. Since we considered only authentication and identity management processes and replayed only HTTP messages (belonging to the user accounts we created), our testing is different from that of [PB14]. Additionally, when we conducted further tests with the plugin, we made sure that the MIG-T did not send too many HTTP requests, to avoid a potential denial of service.

We contacted all the 15 vendors of all the vulnerable web sites through bug bounty platforms like Hackerone⁷ and Bugcrowd⁸ if available, otherwise through the contact information available on the corresponding web sites.

On web sites having well-defined communication channels to report security vulnerabilities (precisely websites including Workable, Naver), we filed vulnerability reports. To contact the others, we used the information available on their web sites for general enquiry.

The communication with the vendors was difficult in terms of explaining them the complexity of the attack and the impact, since it involved two different vulnerabilities. We received some positive responses for our reports. For instance, the Naver Client patched the vulnerability and pay us a bug bounty of up to 100\$. Another Client, Goodreads owns by Amazon, has confirmed the vulnerability, which has already been patched, and confirmed a reward of 200\$. Workable acknowledged the problem, and offered us non-monetary rewards for our findings.

However, no information regarding these vulnerabilities is publicly available. For all other vendors, we are either waiting for the acknowledgments or working closely with them to fix the issues. This is mainly due to the fact that the experiments concluded recently and it has not been long since we reported our findings to the affected vendors. For this reason, we will use aliases instead of the real names of the Clients.

For now, 46% of vendors responded, and from the one that responded, 43% did acknowledged the vulnerability. The results of the disclosure procedure can be found in Table 6.8.

6.3.3.7 Results

We report here the results obtained by following the experiments described in our methodology.

6.3.3.8 Manual testing of the IdPs

We analyzed manually the four IdPs in our dataset to validate our two hypothesis:

- (I1) IdPs tend to be vulnerable to Authentication CSRF.
- (I2) IdPs tend to be reluctant to fix these vulnerabilities.

⁷<https://hackerone.com/>

⁸<https://bugcrowd.com/>

We detected the vulnerability in 3 over 4 of our IdPs, namely Facebook, Google and LinkedIn. The details are below for each vulnerable IdP. We reported the vulnerability to the vendors, but even if they recognize the vulnerability they do not plan to fix the issue.

Facebook. The following steps enable the exploitation of Authentication CSRF in Facebook (let <FB> be the url `https://www.facebook.com/recover`):

1. the attacker opens the link <FB>/initiate/ in a browser.
2. the attacker provides his facebook account email.
3. the attacker enters the code received by email.
4. the attacker presses the “continue” button.
5. the attacker intercepts and drops the HTTP GET request to this url <FB>/password/?u={UCODE}&n={CODE}&fl=default_recover&sih=0&msgr=0 saving the parameter’s value <UCODE> and <CODE> respectively.
6. the attacker forges the following URL <FB>/password/?<QS> where the query string <QS> comprises u=<UCODE>, n=<CODE>, ars=one_click_login, fl=one_click_login, spc=1, ocl=1, sih=0, and msgr=0.
7. the attacker lures the victim to click on the link created at step 6.
8. the victim is now authenticated in Facebook as the attacker.

Google. Calzavara et al. [Ste18b] managed to mount an attack against Google that allows a web attacker to authenticate any user on Google under the attacker’s account. Once accessed, a malicious web page can then cause a victim’s browser to issue the attacker’s request to the Google assertion of SAML consumer service, thus forcing the victim inside the attacker’s controlled authenticated session. The vulnerability can be exploited by any attacker with a valid account on a third party IdP that uses Google as Client. We leveraged this vulnerability to gain access to the victim Google account which plays the role of the IdP in our scenario. We tested this scenario and the vulnerability is still exploitable.

LinkedIn. To exploit the vulnerability in LinkedIn the following steps are sufficient:

1. the attacker opens in a browser the following link `https://www.linkedin.com`.
2. the attacker performs login using his credentials.
3. the attacker clicks “Sign Out” button on the profile page.
4. the attacker opens the URL `https://www.linkedin.com/login?trk=homepage-basic_intl-segments-login` and clicks the button “Sign in with one-time link”.
5. an email with a link has been sent from LinkedIn to the attacker email account to login with one click.
6. the URL to login is `https://www.linkedin.com/e/v2?e=...&lipi=...&a=checkpoint-otp-submit&midToken=...&ek=email_`

```
security_one_time_sign_in_link_checkpoint&sig=...&loginToken=
...&submissionId=...&sessionRedirect=...&fromSignIn=....
```

7. the attacker lures the victim to click on the link created at step 6.
8. the victim is now authenticated in LinkedIn as the attacker.

Table 6.7: Overview of the Client test results per IdP.

	Clients per IdP		Tranco range
	Vulnerable	Not Vulnerable	First-Last
Facebook	5	5	32-322
Google	4	6	32-399
LinkedIn	5	5	3312-32779
Twitter	5	5	32-666
Total	19	21	-

6.3.3.9 Testing of the Clients

By experimenting our MIG-T against our dataset, 19 web sites out of the 40 analyzed (47.5%) were reported vulnerable. For two Clients, namely Naver and Nicovideo, our MIG-T required manual intervention to overcome a captcha and a two-factor authentication check. These classical obstacles to automated testing that are normally present in the productive systems (the one we could analyze), but not in the testing systems that a tester at the Client would deal with. We solved those obstacles manually, assuming that in the testing systems of those websites the testing automation obstacles would not be there. Table 6.7 details the results per IdP, indicating how many Clients are vulnerable and their popularity ranges in the Tranco list [Vic18] as reported in [Moh18].

We analyzed the results of our MIG-T manually, as a tester at the Client would do to be certain of the outcomes. This allowed us to ensure the correctness of our approach in the dataset.

Though the severity of the attack strongly depends on the vulnerable web site, these results are quite alarming. For instance, in the case of Barchart, a Client which supports electronic futures trading, the attacker can sniff the trading strategies of the victim. In the case of Workable, a world’s leading hiring platform, the attacker can hire talents on behalf of the victim.

Table 6.8: Overview of the Client disclosure procedure.

Client Name	responded	vuln confirmed	issue fixed	reward
workable.com	yes	yes	yes	no
naver.com	yes	yes	-	yes
goodreads.com	yes	yes	yes	yes
leetcode.com	no	-	-	-
barchart.com	no	-	-	-
statusbrew.com	no	-	-	-
annualreviews.org	no	-	-	-
nicovideo.jp	no	-	-	-
dailymail.co.uk	no	-	-	-
9gag.com	no	-	-	-
medium.com	yes	waiting	-	-
kaskus.co.id	yes	waiting	-	-
myanimelist.net	no	-	-	-
premierleague.com	yes	waiting	-	-
asos.com	yes	waiting	-	-
Total	7	3	2	2

Chapter 7

Related Work

This Chapter provides an overview of related work of IdM protocols and their associated security risks. The methodology used in the study is a qualitative approach that reviews literature to understand the methods used to evaluate the security of IdM protocols and identify vulnerabilities in the systems. The study leverages the diverse design choices presented in the literature and the valuable insights provided to improve the security of IdM systems. In the last section we provide a comparison between MIG and the other state-of-the-art tools.

7.1 State of the art

The widespread adoption of authentication schemes by numerous websites on the Internet was facilitated by the improved usability brought about by IdM protocols. Allowing users to access multiple services with a single username and password streamlined their browsing experience. However, this improved usability also introduced certain security risks that must be taken into account. Having a single set of credentials for multiple services makes it easier for attackers to access sensitive information if they were to obtain a user's login information. This is only the tip of the iceberg, as many IdM systems also have implementation vulnerabilities that can lead to significant harm if exploited. As a result, security experts have conducted analyses of IdM protocols and identified several weaknesses. The following sections outline the methodology employed and the different approaches used to analyze IdM protocols.

7.2 Methodology

The research we conducted employed a qualitative approach to gain a thorough understanding of the different methods used to analyze IdM protocols. A quantitative approach based on numerical data would not have been appropriate for our goals as we did not require statistical data about IdM protocols, but rather sought to gather in-depth information on the most common methods, their benefits and limitations, and recommendations from prior researchers who have studied this topic.

Thus, we began our analysis by reviewing literature. We used Google Scholar [goob] as a search engine to compile an initial list of papers that studied IdM protocols. However, as our definition of IdM protocols did not match the identity management definition commonly used in literature, we had to start with a broader scope. Our aim was to find studies that analyzed IdM protocols, resulting in the identification of unnoticed or uncovered vulnerabilities. To this end, we started by examining general vulnerability scanners reported in literature, then refined our results by selecting those that aligned with our research focus.

To determine if an article was relevant to our research, we read its abstract and evaluated if it concerned IdM protocols. Specifically, we focused on methods that evaluated the security of IdM protocols and reported new vulnerabilities in IdM systems. To obtain the most relevant information, we ordered the results by relevance, according to Google Scholar. Relevance takes into account factors such as the full text of the article, where it was published, the authors, and its citation frequency by other scholarly literature. These criteria were deemed reasonable by Google Scholar to order the articles and provide the most relevant works.

To begin, a search was conducted using Google Scholar [goob], with initial keywords “vulnerability scanner” and “web scanner”. However, the results obtained were too general and broad. To refine the search, we modified the keywords to “vulnerability scanner sso”, and out of the 4000 pages obtained, the first twelve pages were analyzed, yielding nine relevant results. Further refinement was done by expanding the acronym sso to single sign-on, which resulted in 10000 pages and six additional relevant articles. We also searched with the keywords “black box web vulnerability scanner”, with four potentially useful results obtained from ten pages analyzed. Finally, we tried the keywords “vulnerability scanner” with the names of specific IdM protocols, such as “saml vulnerability scanner”, “oauth vulnerability scanner” and “openid vulnerability scanner”, which produced no significant results, five and seven articles respectively, within our scope.

With approximately thirty papers found, the research was continued by reading the articles. The cross-referencing of articles allowed us to identify additional relevant information and eventually, the cited works in different papers were already in our list, marking a reasonable termination point for our paper review.

The literature on the topic was further enhanced by contributions from Github [git], the premier software development platform, and an international conference on security [oauc]. The extensive research on the security of IdM protocols highlights the importance of our study. Over the years, various aspects of IdM protocols have been investigated, with a common consensus that the implementation of these protocols is often prone to errors. Thus, finding innovative solutions to improve the security of IdM systems continues to be a challenge for the research community. The wealth of research provides valuable insights and helps us understand the crucial aspects to consider when analyzing IdM protocols. We leverage the diversity of design choices presented in the literature as the foundation for our solution. The conducted review is discussed in detail in the following sections.

There are several methods for evaluating the security of IdM protocols, but we decided to not use formal analysis because formal analysis involves using mathematical techniques to model the protocol's behavior and identify potential vulnerabilities or attacks. Formal analysis allows for a thorough and comprehensive evaluation of the protocol's security properties, such as confidentiality, integrity, and availability, and can uncover subtle or complex security flaws that might be missed by other evaluation methods.

Such analysis has uncovered hidden vulnerabilities in various IdM protocols, including SAML Web Browser SSO Profile, OpenID, and OAuth.

In 2008, Armando et al. found a significant vulnerability in the scheme used by Google services after conducting a formal analysis of SAML Web Browser SSO Profile [Ale08]. This vulnerability allowed a malicious Client to impersonate a user at another Client. In 2013, Armando et al. used the model checker from [Ale07] to identify another vulnerability in SAML SSO flow [Ale13]. This vulnerability arose from the lack of a specification mandating that the initial authentication request and the relative assertion be transmitted over the same SSL connection, leaving room for attackers to redirect victims or launch XSS or CSRF attacks. The same check was applied to OpenID and the vulnerability was also present in that protocol.

Fett et al. presented a formal analysis of OIDC in [Dan17]. They formalized the OIDC protocol and included security measures to prevent previously discovered attacks and their variants, demonstrating that their guidelines are effective in ensuring central security properties for OIDC.

Pai et al. used the Alloy framework to discover vulnerabilities in OAuth in 2011 [Suh11] and identified a vulnerability in the client credential flow. In the same year, Chari et al. introduced the Universal Composability Security Framework to study OAuth security [Sur11]. They focused on the authorization code flow and proved that to ensure the security of OAuth, all communications must be over an SSL channel.

7.3 Automated tools

The studies outlined in Section 7.2 all rely on formal models that abstractly examine the IdM protocol. Although the specification of an IdM protocol may be labeled as secure in theory, in practice, developers must contend with various nuances such as different technologies and heterogeneous scenarios that are not considered by formal models. This often results in IdM systems being prone to errors, even if security experts are involved in the development process, leading to the discovery of numerous vulnerabilities (as documented in [San12, ZE14, Kar19, Wan19, Ale08, Avi17]). Additionally, identifying these flaws is a time-consuming process, as testers have to spend a significant amount of time manually checking the IdM system for potential vulnerabilities. This is prone to mistakes by the testers, which can have serious consequences such as non-compliance with regulations like PSD2 [Off15]. Hence, testers must possess expert knowledge and skills to ensure a proper security assessment of the IdM solution. In light of these challenges, researchers have started developing automatic and semi-automatic tools to speed up pentesting and ensure proper implementation flaw detection in IdM systems.

7.3.1 General purpose tools

The initial automated tools centered on web applications where users authenticate to access services. However, these tools are not designed to detect vulnerabilities specific to IdM protocols like SAML or OAuth/OIDC. Instead, they identify common implementation issues in authentication mechanisms.

One of the first tools in this category is **BLOCK** [Xia11], introduced in 2011. Developed by Li and Xue, it operates as a proxy and aims to detect state violation attacks by comparing web traffic generated by the tester machine with a set of expected behaviors, called invariants. If an anomaly is detected, **BLOCK** discards the suspicious message. During the training phase, the tester performs multiple authentications to let **BLOCK** learn the authentication pattern and establish the reference invariants. The testing phase can then commence on the target system, however, this process must be performed manually. The authors confirmed the effectiveness of the tool through tests, but it has limitations. It can only test PHP web applications and search for specific session-targeted attacks, and the completeness and accuracy of the inferred invariants cannot be guaranteed, potentially rendering the tool ineffective.

In 2013, Xing et al. introduced **InteGuard** [Luy13] which follows the footsteps of **BLOCK** [Xia11]. Like **BLOCK**, **InteGuard** acts as a proxy that inspects web traffic and compares intercepted messages against a set of invariants. However, unlike **BLOCK** which analyzes communication between a user agent and a PHP website, **InteGuard** faces a higher level of complexity as it tests IdM systems involving three entities (user agent, Client and IdP). To generate the invariants, **InteGuard** requires four different test accounts and four distinct authentication flows to be

created, with the aim of generating as much diversity as possible in the flows. The HTTP traffic observed during these flows is analyzed, message parameters are extracted, and used to establish rules that can distinguish legitimate messages from illegitimate ones. These invariants are then used to form a set of security policies, and in case of any violations, InteGuard raises an alarm to notify the user of any suspicious activity in their system.

The analysis of web traffic generated by a user's browser has been a popular approach in other tools, with **BRM Analyzer** [Rui12] being one such example. This tool specifically focuses on IdM systems and extracts protocol specifications from messages produced during the authentication flow. It does this by treating a SSO authentication process as a series of browser-relayed messages (BRM) exchanged between the Client and IdP. Upon analysis of these messages, the key elements (parameters) of the IdM system being tested are identified, labeled with a type and semantic, and used to generate a report for the user. While the extraction of specifications is automated, the tester still needs to manually examine the generated report to identify any potential vulnerabilities in the analyzed IdM solution.

A year later, Wang et al. introduced **AUTHSCAN** [BLM⁺13]. This tool obtains information from analyzing HTTP traces and client-side Javascript code to create an initial abstraction of the protocol specification, which is then refined through both black and white box techniques until no further information can be inferred. The refined protocol specifications are represented using an abstract language called Target Model Language and then converted to applied pi-calculus for automated verification against a set of attack candidates. If an attack is detected, it is verified against an oracle using real HTTP traffic to avoid false positives. Using this tool, the authors discovered 7 vulnerabilities in real-world applications. However, it should be noted that the authors point out the need for manual effort to check inconsistencies between the actual implementation and the extracted formal model due to the issue of false positives. In contrast, our approach does not have such a strong requirement.

Prior to examining tools specifically designed for pentesting a particular IdM protocol, literature suggests the use of **CSRF-checker** [Avi17] for performing a general vulnerability assessment of an IdM system. The authors of the study conducted a large-scale analysis of websites with identity management capabilities to determine their susceptibility to authentication CSRF attacks. These types of attacks occur when an attacker logs into a victim's account or the victim is logged into an attacker-controlled account. The authors first manually analyzed the patterns of these attacks, and then developed testing strategies to detect the vulnerabilities causing authentication CSRF. These strategies were integrated into CSRF-checker, a semi-automated tool that operates as an extension of the OWASP ZAP open-source penetration testing tool [zap]. CSRF-checker leverages the API provided by ZAP to identify candidate messages, make necessary modifications, and verify the success of the tests. The results indicate that CSRF-checker is almost as efficient as manual testing, but it only reduces manual effort to a certain extent. The tool's general approach, not targeting a specific protocol, requires a minimum amount of manual operations. As stated by the authors, if a tool for general scenarios is to be developed, automation cannot be

as advanced as for a tool targeting a specific protocol. This is why the following tools concentrate on a specific IdM protocol and aim to uncover vulnerabilities related to its implementation.

7.3.2 Tools for OAuth and OpenID implementations

Starting with OAuth, the first tool to analyze IdM systems using this protocol is **Impersonator** [San12]. Impersonator checks the feasibility of impersonation attacks by sending a stolen or guessed credential to the Client's sign-in endpoint. It does so by exploiting the GeckoFX library [Sof] to simulate a browser, monitor, and modify HTTP messages. Impersonator creates a custom message for the Client being tested and sends it. If the Client does not generate an error, it is considered vulnerable to impersonation attacks. The authors found that 64 out of 96 tested sites were vulnerable, proving the tool's effectiveness in discovering such vulnerabilities. However, Impersonator's limited scope to only one vulnerability makes it less practical for real-world applications.

SSOScan [ZE14] made great strides in 2014, providing a better solution for testing OAuth implementations. SSOScan is an open-source tool available as a web service that performs a vulnerability scan on Clients using Facebook IdP for SSO. In this work, the authors focus only on Facebook SSO, but their approach could be used to check SSO integrations using other IdP or other protocols. The approach simulates attacks and monitors traffic to verify if authentication is successful. SSOScan performs five different attacks, including `access_token` and `signed_request` misuse, `app_secret` parameter leakage, and user credentials leakage through referer headers or third-party scripts. The first two attacks are performed through crafted requests, while the last three are deduced from the authentication process traffic analysis. The first two attacks can only be performed if the authorization code flow is not used. The other three attacks exploit vulnerabilities that give the attacker access to the victim's confidential information. SSOScan is fully automated and protects user privacy, as it automatically generates test accounts. However, while SSOScan is able to automatically synthesize basic user interactions and analyze traffic patterns, this approach is not suitable for detecting all types of vulnerabilities, especially those involving deep server-side scanning or complicated interactions, such as the leakage of app secrets, which requires program analysis techniques.

OAuth Detector [Eth15] focuses on all IdM scenarios using OAuth, searching for vulnerabilities leading to CSRF. The tool is a web crawler that examines website source code to discover OAuth URL links, defined as those containing the parameters `client_id` and `grant_type`. The links are analyzed to determine if they use the authorization code flow, and if so, whether the `state` parameter is present. If `state` is not present, the website is considered vulnerable to CSRF. If no OAuth URL is found, links are followed up to two redirections, with the detection process repeated each time. The authors found that 25% of the domains in the Alexa top 10000 list [ale] using OAuth did not properly implement CSRF protection mechanisms. However, their estimate is considered conservative as the tool does not consider OAuth URLs called via

Javascript and some websites might have non-conventional CSRF protection mechanisms.

The approach taken by **OAuthTester** [Ron16] is similar to those of [Rui12, BLM⁺13]. It starts by constructing a model based on the OAuth RFC [Har] that abstracts and defines the expected behavior of the IdM under test. The model is refined using network traces to better fit the actual behavior of the IdM system. Test cases generated by OAuthTester are then submitted to the real system and the responses are compared with the expected behavior to identify any anomalies and deviations. These deviations are reported as warnings to the user and used to further refine the testing model. Unlike [Rui12] and [BLM⁺13] that only build their models based on network traffic analysis, OAuthTester incorporates the protocol specification in constructing a more comprehensive system model. However, the effectiveness of OAuthTester is limited by the granularity of the model, for example, it does not take into consideration the implementation of SSL or detailed HTTP headers such as cookies, potentially leading to missed vulnerabilities in those areas.

In recent times, two tools for pentesting the OAuth protocol have been introduced: OAuthGuard [Wan19] and OVERSCAN [Kar19]. **OAuthGuard** is a vulnerability scanner and protector for OAuth and OIDC systems that use Google as the IdP. It focuses on the authorization code grant and implicit grant and is the first solution capable of blocking potential attacks in real-time. OAuthGuard works as a Google Chrome extension and protects or warns the user from five different attacks, including CSRF attacks, impersonation attacks, authorization flow misuse, unsafe token transfer, and privacy leaks. All of these attacks are detected through the analysis of messages exchanged through the browser. However, it is important to note that there are certain limitations in the analysis. Specifically, the analysis only focuses on OAuth and OIDC and is limited to the authorization code grant and implicit grant. This means that other authorization protocols and grant types are not covered by the analysis.

OVERSCAN is a tool that is meant to be compatible with any OAuth implementation and serves as a Burp proxy [bur] extension that scans web traffic to identify missing parameters or HTTP headers that might lead to security issues. Using the API provided by Burp, it analyzes messages to identify those that belong to the OAuth protocol. OVERSCAN supports both the OAuth implicit and authorization code grant. The headers and parameters of these messages are then checked to ensure that fields that might allow attackers to perform certain types of attacks are not missing. Based on the analysis results, vulnerable messages are logged in the extension and color-coded to indicate the severity of the identified vulnerability. OVERSCAN does not provide real-time protection, but instead summarizes the results in a report. However, due to its limitations in analyzing the message directly sent between the Client and authorization server, OVERSCAN is unable to provide real-time protection and warnings, instead presenting a comprehensive technical report of any detected vulnerabilities.

The most recent tool for testing OAuth implementation is **OAuch** [Pie], a tool introduced at the 2020 OAuth Security Workshop [oauc]. OAuch is a web application that acts as a malicious client

to find security weaknesses. Each test performed corresponds to one security-related requirement of the OAuth specification, at any requirement level. This means that each *MUSTs*, *SHOULDs*, and *MAYs* present in the OAuth specifications represents a test case. Once basic information is entered into the OAuth interface, the tests are automatically executed. The results of the tests are then displayed in a report, which shows information about each test. For each test, the user can inspect a log with detailed information. However, it is important to acknowledge that the analysis has certain limitations. One such limitation is that it is limited to the examination of OAuth and OIDC, and does not cover other types of authorization protocols.

Regarding OpenID [Ope14a], the initial tool for automated pentesting reported in the literature is **OpenID Attacker** [Chr16a]. As OpenID is an open system, each Client can choose its own IdP, leading the authors to examine a set of attacks that target a malicious IdP. They manually tested four novel attacks on existing OpenID systems, then developed OpenID Attacker to automate the pentesting process. OpenID Attacker operates as a malicious IdP and runs in three modes: analysis, manual attack, and fully automated attack. In the analysis mode, OpenID Attacker functions as an honest IdP creating valid tokens and gathers information about the Client, such as the OpenID version and parameters used. The manual attack mode enables the user to control the malicious IdP and manipulate messages to observe the outcome. The recorded manual attacks can be reused to test other OpenID systems in the future. In the fully automated mode, OpenID Attacker executes the recorded manual attacks and four provided by the tool, including token recipient confusion, ID spoofing, discovery spoofing, and key confusion. These attacks exploit the malicious IdP to impersonate the victim at an honest IdP. The authors tested OpenID Attacker against existing OpenID systems, discovering vulnerable Client on popular services like Sourceforge and Drupal. However, it should be noted that the analysis has certain limitations. In particular, it is focused exclusively on OAuth and OIDC, and does not cover other types of authorization protocols. Furthermore, the analysis only considers a limited number of specific attacks, which means that it may not provide a comprehensive understanding of the full range of security issues and vulnerabilities that could arise in the use of these protocols.

However, OpenID Attacker must be downloaded and manually configured. Mainka et al. proposed **PrOfESSOS** [Chr16b] as an alternative, which is an open-source, fully automated testing tool for IdM systems using OIDC provided as a web service, eliminating the need for configuration. Similar to OpenID Attacker, PrOfESSOS acts as a malicious IdP and performs two types of attacks: single-phase and cross-phase. Single-phase attacks exploit an insufficient or missing verification step on a security-relevant parameter, while cross-phase attacks require the manipulation of messages during multiple phases of the SSO authentication flow. PrOfESSOS is ideal for testing OIDC scenarios, but not intended for large-scale analysis like Alexa top sites [ale]. However, it is important to recognize that the tool has certain limitations. One of these limitations is that it is restricted to the examination of OIDC and does not cover other types of authorization protocols. Additionally, it is important to note that the tool was not intended for large-scale analysis, which means that its applicability may be limited in certain scenarios.

FAPI conformance suite [fabp] is an automated tool for compliance testing of Financial-grade API (FAPI) [fapa], a high-security profile built on OAuth and OIDC for financial scenarios. FAPI is supported by the OpenID Foundation [Ope] and companies like Microsoft and Nomura Research Institute. The OpenID Foundation offers FAPI certification to systems that successfully pass the FAPI conformance suite's compliance tests, which also verify compliance with OAuth and OIDC standards. This suite is designed for testing FAPI implementations but can also be used to assess compliance with OAuth and OIDC standards. However, it is important to note that the tool has some limitations. In particular, it is restricted to the examination of OIDC and does not cover other types of authorization protocols.

The **MoScan** tool [Han21] is designed for software testers and security analysts to detect and report security vulnerabilities in SSO implementations. It works by analyzing network traces and incrementing a state machine, which is then used to generate payloads for testing the protocol participants. However, it is essential to recognize that while the tool was able to identify three known weaknesses and one new logic fault, there may be additional vulnerabilities that it was not able to detect.

OAuthShield [Tam21] is an automated static analyzer for finding logical flaws and vulnerabilities in OAuth authorization server libraries. It checks the security of service provider implementations for OAuth authorization. However, it is essential to recognize that the tool has some limitations. One of the significant limitations is that it is restricted to the evaluation of OAuth and does not cover other types of authorization protocols.

OCSRF [Mic21] is a Python-Selenium tool for automatically testing different OAuth CSRF scenarios. It has a repeatable methodology for discovering and validating OAuth CSRF vulnerabilities in targeted sites, and presents the most comprehensive set of test cases to exploit these vulnerabilities, including novel attack strategies. It complements and integrates guidelines for OAuth developers to mitigate implementation mistakes. However, it is crucial to recognize that the tool has certain limitations, and one of these limitations is that it is limited to examining OAuth.

S3KVetter [Ron18] (Single-Sign-on SDK Vetter) is an automated, efficient testing tool for checking the logical correctness and identifying vulnerabilities of SSO SDKs. However, it is essential to recognize that the tool has certain limitations, and one of these limitations is that it may produce false negatives in certain scenarios.

7.3.3 Tools for SAML implementations

Automated pentesting tools for IdM systems implementing SAML have been developed. The first such tool was **WS-Attacker** presented by Mainka et al. in 2012 [Chr12]. The authors described a framework for testing various XML-specific vulnerabilities, but the automated tool they implemented only checks for SOAPAction spoofing and WS-Addressing spoofing, which

are based on vulnerabilities in the SOAP protocol. The tool requires a WSDL [wsdl] file as input and generates two requests – one correct and one tampered – to compare and present the results to the user. However, WS-Attacker is limited in that it is not useful for testing current IdM systems using SAML, as they mostly use HTTP to transfer authentication messages.

In contrast, **SAML Raider** [PS18] and **EsPRESSO** [MMGS15] are both Burp extension tools that test for XML-specific vulnerabilities. SAML Raider verifies if the target SAML IdM system is vulnerable to XML Signature Wrapping (XSW) attacks, which allow an attacker to modify XML messages and inject malicious information. SAML Raider also allows the user to remove, edit, and resign XML signatures. EsPRESSO, on the other hand, identifies and decodes messages belonging to IdM protocols (such as SAML and OAuth), and tests for vulnerabilities leading to XML External Entities (XXE) attacks. A similar tool, **SAMLyze** [Jon], is not currently available. However, these tools provide pre-compiled attack vectors, but manual testing is still required, as the user must manually select the attack vector and determine the outcome of the authentication flow.

7.3.4 Tools for OAuth/OIDC and SAML

If we search for comprehensive automated penetration testing tools for OAuth/OIDC and SAML vulnerabilities, we find only one option: **WPSE** [Ste18b]. WPSE is a security monitor implemented as a Google Chrome extension designed to ensure compliance with the intended protocol flow and the confidentiality and integrity properties of messages. It requires a protocol flow specification and a security policy as inputs to evaluate traffic generated by the user's browser and block navigation in case of anomalies. The tool can detect attacks that cause protocol flow deviations, secrecy violations, and integrity violations. Protocol flow deviations occur when the user's browser skips required messages or processes messages in the wrong order (e.g., in some CSRF attacks, the victim completes an authentication flow that wasn't initiated in their browser). Secrecy violations occur when confidential information is leaked (e.g., via third-party code), and integrity violations occur when the integrity constraints of the protocol are not satisfied (e.g., the `redirect_uri` being modified during the authentication flow). The authors acknowledge that WPSE still needs improvement, for instance, it blocks all solutions for CSRF causing protocol flow deviation in OAuth that do not use the `state` parameter, even if the website implements other countermeasures. Improving the way of detecting CSRF (as proposed in [Wan19]) or providing a warning and letting the user decide whether to proceed could be potential solutions. However, it is important to acknowledge that the current prototype of WPSE is limited by the Google Chrome extension APIs.

Sudhodanan et al. proposed **BLAST** [Avi16] a tool that aims to improve the security of MPWAs by automatically testing them for security vulnerabilities. The authors were motivated by the prevalence of attacks against MPWAs and the lack of a general-purpose technique to support their discovery. The tool is based on attack patterns and uses a black-box testing approach

to target six different replay attacks, a login CSRF attack, and a persistent XSS attack. The proposed methodology allows security experts to create attack patterns from known attacks, and the framework automatically generates test cases for security testing. However, the approach heavily relies on predefined attack patterns inspired by known attacks, limiting its ability to detect attacks that do not fit into these patterns and it is also subject to false positives.

7.3.5 Sandboxing and education

All the tools discussed so far share a common characteristic: they have validated their proposed solutions through tests conducted on real-world identity management (IdM) systems. This is to show that the solutions are effective both theoretically and in practice. Major companies such as Google, Facebook, and Microsoft run bug bounty programs [gooc, fac, mic] where rewards are offered to individuals who discover and report security flaws in their systems. However, pentesting real-world applications may raise legal and safety concerns, especially for companies like banks or public organizations who may not appreciate unauthorized access to their systems, and for attacks like Denial of Service that may cause serious harm to the impacted business. To address this, it's crucial to have a secure environment for assessing the security of IdM systems, such as sandboxing the tested environment on a local machine to keep it isolated.

Dashevskiy et al. proposed **TESTREX** [Sta14] as a testbed for repeatable exploits with features like packing and running applications with their environments, injecting exploits, monitoring their success, and generating security reports. Instead of using heavy virtual machines, TESTREX uses Docker [doc], a software that provides virtualization at the OS level using containers, to isolate the application being tested. The user writes the exploits using a reference class, which are then executed using Selenium Web Driver [sel]. The results are recorded in a report that the user can examine. TESTREX is suitable for both industrial and educational use and provides a set of vulnerable web applications for various web programming languages, partly developed by the authors and partly sourced from existing environments like BugBox [Gar13] and WebGoat [web]. However, it is important to note that the tool has certain limitations, and one of these limitations is that it is not designed to focus on IdM protocols.

BugBox [Gar13] is a framework for collecting, sharing, and experimenting with vulnerabilities, and for educating users on common vulnerabilities. It provides environments with each vulnerability coupled to an automated exploit executed using Selenium, and the full source code of the vulnerable applications is provided. However, it is important to note that the solution being discussed is limited to PHP web applications.

WebGoat [web] is an insecure web application for learning and testing common web vulnerabilities. The user can choose a vulnerability to learn and is guided through the process with descriptions and instructions. However, it is worth noting that the solution being discussed has limitations when it comes to real environments. While it may work well in controlled testing

conditions, there could be unforeseen challenges or performance issues when the solution is implemented in a real-world setting.

For educational purposes, two online services for learning the IdM protocol are worth mentioning: **Okta OAuth Playground** [oaua] and **Google OAuth Playground** [gooa]. Okta OAuth Playground provides practical examples to learn OAuth grant types, while Google OAuth Playground offers a sandbox for playing with OAuth and Google APIs supporting it. However, it is worth highlighting that the tools being discussed have certain limitations. Specifically, they focus solely on OAuth authentication and do not include the capability to perform security testing on production environments.

7.3.6 Considerations

We have conducted a thorough examination of the state of the art and organized the gathered information in Table 7.1. The tools have been categorized into three distinct sub-groups, namely, *General purpose*, which refers to tools utilized for testing web applications with user authentication, *OAuth/OIDC*, which pertains to tools specifically designed for OAuth/OIDC implementations, and *SAML*, which includes tools intended for SAML implementations. The following features were selected for comparison:

- **(Semi) Automated**: automatic or semi-automatic testing with limited user interaction.
- **Common Vulnerability**: tests for vulnerabilities commonly found in web applications.
- **Compliance**: verification of compliance with the standards specified in the identity management (IdM) protocol.
- **IdM protocol**: testing for vulnerabilities specific to IdM protocol implementation.
- **Active tests**: modifying messages to examine system reactions.
- **Passive tests**: analyzing authentication flow traffic to detect anomalies.
- **MSC**: provision of message sequence charts and/or lists of messages exchanged during authentication flow.
- **Vulnerable instances**: provision of instances for pentesting and learning.
- **Automatic Deploy**: ease of deployment of vulnerable instances through simple commands or scripts.
- **Learning Information**: provision of information to enhance understanding of IdM protocols and improve security awareness.

Table 7.1: Comparison between MIG and the other state-of-the-art tools.

	Pentesting							Sandboxing & Education		
	(Semi) Automated	Web Application Security	Compliance	IdM protocol	Active tests	Passive tests	MSC	Vulnerable instances	Automatic deploy	Learning information
General purpose										
AUTHSCAN [BLM ⁺ 13]	■	■	-	■	■	-	-	-	-	-
BLAST [Avi16]	■	■	-	■	■	-	-	-	-	-
BLOCK [Xia11]	■	■	-	-	-	■	-	-	-	-
BRM Analyzer [Rui12]	-	-	-	-	-	-	■	-	-	-
BugBox [Gar13]	■	■	-	-	-	-	-	■	-	-
CSRF-checker [Avi17]	■	-	-	-	■	-	-	-	-	-
InteGuard [Luy13]	■	■	-	-	-	■	-	-	-	-
TESTREX [Sta14]	■	■	■	-	-	■	-	■	■	-
WebGoat [web]	-	-	-	-	-	-	-	■	-	■
OAuth/OIDC										
Google Playground [gooa]	-	-	-	-	-	-	-	-	-	■
FAPI conformance suite [fapb]	■	-	■	-	-	■	-	-	-	-
Impersonator [San12]	■	-	-	■	■	-	-	-	-	-
OAuch [Pie]	■	-	■	-	■	■	-	-	-	-
OAuth Detector [Eth15]	■	-	-	■	-	■	-	-	-	-
OAuthGuard [Wan19]	■	-	-	■	-	■	-	-	-	-
OAuthTester [Ron16]	■	■	-	-	-	■	-	-	-	-
Okta Playground [oaua]	-	-	-	-	-	-	-	-	-	■
OpenID attacker [Chr16a]	■	-	-	■	■	-	-	-	-	-
OVERSCAN [Kar19]	■	-	-	■	-	■	■	-	-	-
PrOfESSOS [Chr16b]	■	-	-	■	■	-	-	-	-	-
SSOScan [ZE14]	■	-	-	■	■	■	-	-	-	-
WPSE [Ste18b]	■	■	■	-	-	■	-	-	-	-
OCSRF [Mic21]	■	■	■	-	■	-	-	-	-	-
MoScan [Han21]	■	■	■	-	■	-	-	-	-	-
OAuthShield [Tam21]	■	■	■	■	■	-	-	-	-	-
S3kVetter [Ron18]	■	■	■	■	■	-	-	-	-	-
SAML										
EsPReSSO [MMGS15]	■	-	-	■	■	-	■	-	-	-
SAMLyze [Jon]	■	-	-	■	■	-	-	-	-	-
SAML Raider [PS18]	■	-	-	■	■	-	-	-	-	-
WPSE [Ste18b]	■	■	■	-	-	■	-	-	-	-
WS-Attacker [Chr12]	■	-	-	■	-	-	-	-	-	-
Micro Id Gym	■	■	■	■	■	■	■	■	■	■

■ = provided feature; - = feature not provided

From the table's results, it is clear that there is no tool that offers all the features that MIG provides. To ensure compatibility with future works, MIG was designed and developed with three concepts in mind: *modularity*, *flexibility*, and *usability*. Modularity ensures each component of MIG is independent, so components can be added or removed without affecting others. Flexibility allows MIG components to adapt to different environments and run on both Windows and Unix systems. Usability focuses on making MIG easy to use with straightforward configuration, automatic deployment and execution, and user-friendly interaction. In the subsequent chapters, MIG's features are described in detail.

Chapter 8

Conclusions and Future Work

Ensuring the security of an organization's digital assets has become increasingly important in today's technology-driven world, with the proliferation of cyber threats. To secure against data breaches, loss of confidential information, financial damage, and reputational harm, organizations must implement effective cybersecurity measures. Regular security testing is one such measure that can help assess the effectiveness of security controls, identify vulnerabilities, and make recommendations for improvement. By conducting security tests on a regular basis, organizations can proactively address potential threats and strengthen their overall cybersecurity posture. In addition to regular security testing, digital IdM is also crucial in protecting against identity theft, fraud, and other security breaches. MPWA are commonly used in digital identity systems and play a crucial role in building trust in current and future digital ecosystems. IdM protocols such as SAML, OIDC, and OAuth provide policies for securing assertions and ensuring their integrity during transfer, while also eliminating the need to store authentication information within services. This solution can help public and private organizations prevent the misuse or abuse of login credentials and reduce the risk of data breaches.

However, implementing IdM protocols can be challenging due to the complexity involved in designing, implementing, and deploying cryptographic mechanisms. Automated security testing is crucial in mitigating security issues, especially as the functional details of implementations and deployments add to the complexity of IdM solutions. Compliance with security standards is also critical for robust security in IdM implementation, as it is related not only to security policies but also to legal aspects related to national-level infrastructure. The implementation of IdM protocols which include standards requires robust security testing to ensure their proper functioning and protect against potential threats. However, this process is not without its challenges. The first challenge is the scattered nature of security information, which makes it difficult for Security Testers to access and gather all relevant information required for testing. The second challenge is the fast-evolving threat landscape, which requires continuous updates to security testing techniques and approaches to keep pace with the latest threats. Finally, the need for a controlled

testing environment that guarantees traceability, reproducibility, and business continuity further adds to the complexity of security testing. To address these challenges, organizations need to establish a robust and agile security testing strategy, supported by suitable infrastructure and testing procedures, that can ensure the security and resilience of their IdM implementations.

For this, we proposed Micro-Id-Gym (MIG), offering on the one hand (in the laboratory) an easy way to configure and reproduce the production environment in a sandbox. This can be done by choosing instances built in the framework or allowing users to upload new ones. Security Testers can then develop hands-on experiences on how IdM solutions work and also use a set of security testing tools to perform attacks with high impact and better understand the underlying security issues. On the other hand (in the wild), MIG provides a set of security testing tools for the automatic security analysis of IdM protocols as reported in Chapters 4 and 5. These tools include: *(i)* MIG-T, which is designed to automatically execute test cases specified in MIG-L (see Chapter 3), *(ii)* MIG-D, which allows users to easily visualize and analyze the results of security testing activities in the form of graphical representations, and *(iii)* MIG-S, which facilitates the creation and sharing of cyber threat intelligence information. As reported in Chapter 6, we first evaluated the effectiveness of MIG by conducting an experiment to demonstrate its accuracy in assisting with the detection of relevant vulnerabilities in the implementation of IdM protocols. Secondly, we utilized MIG and MIG-T to conduct security analyses across various corporate scenarios and projects, identifying security concerns in multiple areas such as the implementation of IdM protocol in the Italian digital identity system supporting OIDC and SAML, a PSD2 service provided by a prominent Italian IdP based on OAuth, and the SSO-linking. During our analysis we identified that 19 out of 40 Clients analyzed were vulnerable to a CSRF on the action initiating the SSO-Linking process at the Client where the attacker will be able to authenticate to the victim's Client account by executing the SSO login at the Client through the attacker's IdP account. As part of our commitment to security, we participated in various bug bounty programs to responsibly disclose the vulnerabilities we identified and collaborating with the security researchers to further enhance their systems. Our findings were recognized by the providers, who awarded us both monetary compensation and public recognition.

As future work, we plan to design and integrate in MIG a security chatbot designed to assist in the implementation of IdM protocols in complex and dynamic environments, providing real-time feedback on security threats and vulnerabilities. Our aim is to create a user-friendly, effective, and reliable security chatbot that will improve the security posture of organizations, reduce the workload of Security Testers, enable developers to easily integrate IdM protocols and make a significant contribution to the field of IdM.

Appendix A

Survey Questionnaires' Content

Table A.1 shows the content of the post-experiment survey questionnaire mentioned in Section 6.1.2. It deals with object clarity of the tasks, cognitive effects of the treatments on the behavior of the subjects and perceived usefulness of MIG-D. The first set of questions (Q1-Q6) needs to be answered twice (one answer for each performed lab) while the remaining set only needs to be answered once as it refers to the overall session.

Table A.1: Post-experiment survey questionnaire.

ID	Applies to	Question
Q1	Each lab	I had enough time to perform the tasks. (1-5).
Q2	Each lab	I experienced no difficulty in detecting the vulnerability. (1-5).
Q3	Each lab	Which operations (e.g., mouse over steps, open tab, search, . . .) did you perform to understand whether the protocol was vulnerable to the mentioned vulnerability?
Q4	Each lab	Did you consult internet to find help to answer the questionnaire? If yes, which online queries did you search(e.g., keywords used)? Which content was helpful?
Q5	Overall	Which tool did you find more useful to answer the questions? (Report Lab 1-2).
Q6	Overall	Which tool did you find more intuitive? For which tool was more difficult to find the proper information about the protocol in order to answer the questions? (Report of Lab 1-2).
Q7	Overall	Which tool would you use for your work? Motivate your answer (to the previous question). (open question).
Q8	Overall	Do you know any tool that performs similar tasks? (open question).
Q9	Overall	Do you have any suggestion related to the tool usage? (open question).
Q10	Overall	What do you think is the main advantage using MIG-D? Would you add more information to the MIG-D? (open question).

Appendix B

Mapping between MIG-L and Machine Readable Specification

The mapping between the operations in MIG-L and their corresponding translations in the machine readable JSON specification is presented in Table B.1.

Table B.1: Mapping between MIG-L and JSON.

MIG-L	JSON
Testsuite definition	
<code>define_suite(\$name, \$description, \$filter) <i>list of tests</i></code>	<pre>1 { 2 "test suite":{ 3 "name": "\$name", 4 "description": "\$description", 5 "filter messages": "\$filter" }, 6 "tests": <i>list of tests</i> 7 }</pre>
Table Continues →	

List of Tests

Passive Test definition

<code>start_test (\$name, \$description, passive) <i>list of operations</i> (See List of Operations for Passive Test) end_test ()</code>	<pre>1 { 2 "test": { 3 "name": "\$name", 4 "description": "\$description", 5 "type": "passive", 6 "operations": 7 <i>list of operations</i> 8 } }</pre>
--	---

Active Test definition

<code>start_test (\$name, \$description, active, \$sx, \$result) <i>list of operations</i> (See List of Operations for Active Test) end_test ()</code>	<pre>1 { 2 "test": { 3 "name": "\$name", 4 "description": "\$description", 5 "type": "active", 6 "sessions": ["\$sx"], 7 "operations": <i>list of</i> 8 <i>operations</i> 9 "result": "\$result" 10 } }</pre>
--	--

List of Operations for Passive Test

<code>regex(\$regexToApply, \$message_sec, \$message_type)</code>	<pre>1 { 2 "message type": "\$message_type", 3 "regex": "(\$regexToApply)", 4 "message section": "\$message_sec" 5 }</pre>
<code>start_checks(\$message_type) <i>list of checks</i> end_checks ()</code>	<pre>1 { 2 "message type": "message_type", 3 "checks": <i>list of checks</i> 4 }</pre>

Table Continues →

List of Checks

check(\$string, \$action, \$in)	<pre>1 { 2 "in": "\$in", 3 "check": "\$string", 4 "\$action": "\$action" 5 }</pre>
check_parameter(\$param, \$action, \$in, \$value)	<pre>1 { 2 "in": "\$in", 3 "check parameter": "\$param", 4 "\$action": "\$value" 5 }</pre>

List of Operations for Active Test

start(\$session_name)	<pre>1 { 2 "session": "\$session_name", 3 "action": "start" 4 }</pre>
stop(\$session_name)	<pre>1 { 2 "session": "\$session_name", 3 "action": "stop" 4 }</pre>
pause(\$session_name)	<pre>1 { 2 "session": "\$session_name", 3 "action": "pause" 4 }</pre>
resume(\$session_name)	<pre>1 { 2 "session": "\$session_name", 3 "action": "resume" 4 }</pre>
clear_cookies(\$session_name)	<pre>1 { 2 "session": "\$session_name", 3 "action": "clear cookies" 4 }</pre>
start_validate() <i>list of checks</i> end_validate()	<pre>1 { 2 "action": "validate", 3 "checks": <i>list of checks</i> 4 }</pre>

Table Continues →

<pre> start_intercept (\$message_type, \$from_session, \$then) start_preconditions <i>list of preconditions</i> (see List of Operations for Passive Test) end_preconditions start_msg_operations <i>list of message operations</i> end_msg_operations start_session_operations <i>list of session operations</i> end_session_operations end_intercept() </pre>	<pre> 1 { 2 "action": "intercept", 3 "then": "\$then", 4 "message type": "\$message_type", 5 "from session": "\$session_name", 6 "preconditions": <i>list of</i> 7 <i>preconditions</i>, 8 "message operations": <i>list of</i> 9 <i>message operations</i>, "session operations": <i>list of</i> <i>session operations</i>, } </pre>
--	---

<pre> save_message(\$variable) </pre>	<pre> 1 { 2 "save": "\$variable" 3 } </pre>
---------------------------------------	---

<pre> replace(\$saved_variable, \$is_request) </pre>	<pre> 1 { 2 "\$is_request": "\$saved_variable" 3 } </pre>
--	---

List of Session Operations

<pre> mark(\$marker_name, \$action, \$session_name) </pre>	<pre> 1 { 2 "session": "\$session_name", 3 "mark": "\$action", 4 "name": "\$marker_name" 5 } </pre>
--	---

<pre> save(\$variable_name, \$action, \$session_name) </pre>	<pre> 1 { 2 "session": "\$session_name", 3 "save": "\$action", 4 "as": "\$variable_name" 5 } </pre>
--	---

<pre> add(\$insert, \$marker_name, \$session_name) </pre>	<pre> 1 { 2 "session": "\$session_name", 3 "insert": "\$insert", 4 "at": "\$marker_name" 5 } </pre>
---	---

Table Continues →

remove(\$marker_name, \$session_name)	<pre> 1 { 2 "session": "\$session_name", 3 "remove": "\$marker_name" 4 } </pre>
--	---

List of Message Operations

gen_poc(\$pattern, \$name)	<pre> 1 { 2 "pattern": "\$pattern", 3 "name": "\$name" 4 } </pre>
----------------------------	---

add_parameter(\$param_name, \$param_value, \$from)	<pre> 1 { 2 "from": "\$from", 3 "add": "\$param_name", 4 "this": "\$param_value" 5 } </pre>
---	---

edit_parameter(\$param_name, \$new_value, \$from)	<pre> 1 { 2 "from": "\$from", 3 "edit": "\$param_name", 4 "in": "\$new_value" 5 } </pre>
--	--

edit_regex(\$regex, \$new_value, \$from)	<pre> 1 { 2 "from": "\$from", 3 "edit regex": "\$regex", 4 "in": "\$new_value" 5 } </pre>
---	---

remove_parameter (\$param_name, \$from)	<pre> 1 { 2 "from": "\$from", 3 "remove parameter": "\$param_name" 4 } </pre>
--	---

remove_match_word(\$word, \$from)	<pre> 1 { 2 "from": "\$from", 3 "remove match word": "\$word" 4 } </pre>
--------------------------------------	--

save_parameter(\$param_name, \$variable, \$from)	<pre> 1 { 2 "from": "\$from", 3 "save": "\$param_name" 4 "as": "\$variable" 5 } </pre>
---	--

Table Continues →

save_match(\$string, \$variable, \$from)	<pre> 1 { 2 "from": "\$from", 3 "save matches": "\$string" 4 "as": "\$variable" 5 } </pre>
decode_param(\$param_name, \$from, \$encoding, \$type, \$remove_sign, \$self_sign)	<pre> 1 { 2 "from": "\$from", 3 "decode param": "\$param_name", 4 "encodings": \$encoding, 5 "type": "\$type", 6 "remove-sign": "\$remove_sign" 7 "self-sign": "\$self_sign" 8 } </pre>
remove_tag(\$tag_name)	<pre> 1 { 2 "type": "xml", 3 "remove tag": "\$tag_name" 4 } </pre>
remove_attribute(\$attribute, \$tag_name)	<pre> 1 { 2 "type": "xml", 3 "remove attribute": "\$attribute" 4 "xml tag": "\$tag_name" 5 } </pre>
edit_tag(\$tag_name, \$new_value)	<pre> 1 { 2 "type": "xml", 3 "edit tag": "\$tag_name" 4 "value": "\$new_value" 5 } </pre>
edit_attribute(\$attribute, \$tag_name, \$new_value)	<pre> 1 { 2 "type": "xml", 3 "edit attribute": "\$attribute", 4 "xml-tag": "\$tag_name", 5 "value": "\$new_value" 6 } </pre>
add_tag(\$new_tag, \$parent_tag, \$new_value)	<pre> 1 { 2 "type": "xml", 3 "add tag": "\$new_tag", 4 "xml-tag": "\$parent_tag", 5 "value": "\$new_value" 6 } </pre>

Table Continues →

add_attribute(\$new_attr, \$tag_name, \$new_value)	<pre> 1 { 2 "type": "xml", 3 "add attribute": "\$new_attr", 4 "xml-tag": "\$tag_name", 5 "value": "\$new_value" 6 } </pre>
save_tag(\$tag_name)	<pre> 1 { 2 "type": "xml", 3 "save-tag": "\$tag_name" 4 } </pre>
save_attribute (\$attr_name, \$tag_name)	<pre> 1 { 2 "type": "xml", 3 "save-attribute": "\$attr_name", 4 "xml-tag": "\$tag_name" 5 } </pre>
txt_remove(\$value)	<pre> 1 { 2 "type": "txt", 3 "txt remove": "\$value" 4 } </pre>
txt_edit(\$text, \$new_text)	<pre> 1 { 2 "type": "txt", 3 "txt edit": "\$text", 4 "value": "\$new_text" 5 } </pre>
txt_add(\$text, \$new_text)	<pre> 1 { 2 "type": "txt", 3 "txt add": "\$text", 4 "value": "\$new_text" 5 } </pre>
txt_save(\$text, \$new_var)	<pre> 1 { 2 "type": "txt", 3 "txt save": "\$text", 4 "as": "\$new_var" 5 } </pre>
jwt_remove(\$from, \$parameter)	<pre> 1 { 2 "type": "jwt", 3 "from": "\$from", 4 "jwt remove": "\$parameter" 5 } </pre>

Table Continues →

<code>jwt_edit(\$from, \$parameter, \$new_value)</code>	<pre>1 { 2 "type": "jwt", 3 "from": "\$from", 4 "jwt edit": "\$parameter", 5 "value": "\$new_value" 6 }</pre>
<code>jwt_add(\$from, \$parameter, \$new_value)</code>	<pre>1 { 2 "type": "jwt", 3 "from": "\$from", 4 "jwt add": "\$parameter", 5 "value": "\$new_value" 6 }</pre>
<code>jwt_save(\$from, \$parameter, \$new_value)</code>	<pre>1 { 2 "type": "jwt", 3 "from": "\$from", 4 "jwt save": "\$parameter", 5 "as": "\$new_value" 6 }</pre>
<code>jwt_sign</code>	<pre>1 { 2 "type": "jwt", 3 "from": "\$from", 4 "jwt sign": "true" 5 }</pre>

Bibliography

- [Ada08] Adam Barth, Collin Jackson, and John C. Mitchell. Robust Defenses for Cross-Site Request Forgery. In *Proceedings of the 15th ACM Conference on Computer and Communications Security, CCS '08*, pages 75–88, New York, NY, USA, 2008. Association for Computing Machinery.
- [Age19] Agenda Digitale. Sicurezza PA, tempo di responsible disclosure e bounty program anche in Italia. <https://www.agendadigitale.eu/sicurezza/sicurezza-pa-tempo-di-responsible-disclosure-e-bounty-program-anche-in-italia/>, 2019.
- [AGI] AGID and Team Digitale. Art. 64-bis. Accesso telematico ai servizi della pubblica amministrazione. https://docs.italia.it/italia/piano-triennale-ict/codice-amministrazione-digitale-docs/it/v2017-12-13/_rst/capo5_sezione3_art64-bis.html.
- [ale] The top 500 sites on the web. <https://www.alexa.com/topsites>. Last access on May 21, 2023.
- [Ale07] Alessandro Armando, Roberto Carbone and Luca Compagna. LTL Model Checking for Security Protocols. In *20th IEEE Computer Security Foundations Symposium (CSF'07)*, pages 385–396, 2007.
- [Ale08] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar and Llanos 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, FMSE '08*, pages 1–10, New York, NY, USA, 2008. Association for Computing Machinery.
- [Ale13] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, Giancarlo Pellegrino and Alessandro Sorniotti. An authentication flaw in browser-based single sign-on protocols: Impact and remediations. *Computers & Security*, 33:41 – 58, 2013.
- [Anda] Andrea Bisegna and Matteo Bitussi. MIG-L Tests. https://drive.google.com/file/d/1d0HHjOyYL0L3SqahTgd3AGNqYntZ4GZC/view?usp=share_link.

- [Andb] Andrea Bisegna and Matteo Bitussi. SSOLinking MIG-L Tests. https://drive.google.com/file/d/1-iFMb0a1-ieQWXfH1A8nMCXffKxxhdOZ/view?usp=share_link.
- [And19] Andrea Bisegna, Roberto Carbone, Ivan Martini, Valentina Odorizzi, Giulio Pellizzari and Silvio Ranise. Micro-Id-Gym: Identity Management Workouts with Container-Based Microservices. *Int. J. Inf. Secur. Cybercrime*, 8(1):45–50, 2019.
- [And20] Andrea Bisegna, Roberto Carbone, Giulio Pellizzari and Silvio Ranise. Micro-Id-Gym: a flexible tool for pentesting identity management protocols in the wild and in the laboratory. In *Emerging Technologies for Authorization and Authentication: Third International Workshop, ETAA 2020, Guildford, UK, September 18, 2020, Proceedings 3*, pages 71–89. Springer, 2020.
- [And22] Andrea Bisegna, Roberto Carbone and Silvio Ranise. Integrating a Pentesting Tool for IdM Protocols in a Continuous Delivery Pipeline. In *Emerging Technologies for Authorization and Authentication: 4th International Workshop, ETAA 2021, Darmstadt, Germany, October 8, 2021, Revised Selected Papers*, pages 94–110. Springer, 2022.
- [Avi16] Avinash Sudhodanan, Alessandro Armando, Roberto Carbone and Luca Compagna. Attack Patterns for Black-Box Security Testing of Multi-Party Web Applications. In *NDSS*, 2016.
- [Avi17] Avinash Sudhodanan, Roberto Carbone, Luca Compagna, Nicolas Dolgin, Alessandro Armando and Umberto Morelli. Large-scale analysis & detection of authentication Cross-Site Request Forgeries. In *2017 IEEE European symposium on security and privacy (EuroS&P)*, pages 350–365. IEEE, 2017.
- [BCC⁺20] Andrea Bisegna, Roberto Carbone, Mariano Ceccato, Salvatore Manfredi, Silvio Ranise, Giada Sciarretta, Alessandro Tomasi, and Emanuele Vigliani. 6. Automated Assistance to the Security Assessment of API for Financial Services. In *Cyber-Physical Threat Intelligence for Critical Infrastructures Security: A Guide to Integrated Cyber-Physical Protection of Modern Critical Infrastructures*. Now Publishers, 2020.
- [BLM⁺13] Guangdong Bai, Jike Lei, Guozhu Meng, Sai Sathyanarayan Venkatraman, Prateek Saxena, Jun Sun, Yang Liu, and Jin Song Dong. AUTHSCAN: Automatic Extraction of Web Authentication Protocols from Implementations, 2013.
- [Blo] The Duo Blog. Duo Finds SAML Vulnerabilities Affecting Multiple Implementations. <https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations>. Last access on May 21, 2023.
- [Bra05] Brad Arkin, Scott Stender and Gary McGraw. Software penetration testing. *IEEE Security & Privacy*, 3(1):84–87, 2005.
- [Bra19] John Bradley. PKCE: The OAuth 2.0 Extension That Solves Mobile and Native App Authorization. *OAuth*, 2019(1):8–15, 2019.
- [bur] Burp Suite. <https://portswigger.net/burp>. Last access on May 21, 2023.

- [Bur05] Steven F. Burns. Threat modeling: A process to ensure application security. *GIAC security essentials certification (GSEC) practical assignment*, 2005.
- [BV16] Giovanni Bajo and Gianluca Varisco. Perché la sicurezza informatica non è una questione di bianco e nero. <https://medium.com/team-per-la-trasformazione-digitale/sicurezza-informatica-policy-responsible-disclosure-hacker-etici-52a174d44c49>, 2016.
- [Car23] Aviad Carmel. Traveling with OAuth - Account Takeover on Booking.com. <https://salt.security/blog/traveling-with-oauth-account-takeover-on-booking-com>, 2023.
- [Chr12] Christian Mainka, Juraj Somorovsky and Jörg Schwenk. Penetration testing tool for web services security. In *2012 IEEE Eighth World Congress on Services*, pages 163–170. IEEE, 2012.
- [Chr16a] Christian Mainka, Vladislav Mladenov and Jörg Schwenk. Do not trust me: Using malicious IdPs for analyzing and attacking Single Sign-On. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 321–336. IEEE, 2016.
- [Chr16b] Christopher Späth, Christian Mainka, Vladislav Mladenov and Jörg Schwenk. SoK: XML Parser Vulnerabilities. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX, August 2016. USENIX Association.
- [Cla01] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell and Anders Wesslén. Experimentation in Software Engineering. *Softw. Test., Verif. Reliab.*, 2001.
- [CMP17] Stefano Calzavara, Andrea Marin, and Alessandro Palù. Network attacker capabilities modeling in security risk assessment. In *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2017.
- [CMP18] Stefano Calzavara, Andrea Marin, and Alessandro Palù. Web attacker capabilities modeling in security risk assessment. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, 2018.
- [Con08] OASIS Consortium. SAML V2.0 Technical Overview. <http://wiki.oasis-open.org/security/Saml2TechOverview>, March 2008.
- [Dan16] Danfeng Yao, Songqing Lian and Xiao Feng Wang. How to Break XML-based Security in Real World: A Study of XML Signature Wrapping Attacks in SAML Single Sign-on Systems. *IEEE Transactions on Dependable and Secure Computing*, 13(1):1–14, 2016.
- [Dan17] Daniel Fett, Ralf Küsters and Guido Schmitz. The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 189–202, 2017.
- [Dev] Developers Italia. SPID/CIE OIDC Federation SDK. <https://github.com/italia/spid-cie-oidc-django>.
- [Dev11] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*, 2011.

- [Dev23] Developers Italia. SPID/CIE OpenID Connect, 2023. Commit 372212cde768ae344e3cb3187d5c11fc329e6598 in <https://github.com/italia/spid-cie-oidc-docs>.
- [Dig] Agenzia Per L'Italia Digitale. SPID - Regole Tecniche. <https://docs.italia.it/italia/spid/spid-regole-tecniche/it/stabile/index.html>.
- [Dim19] Dimitrina Hristova, Michael Backes, Sascha Fahl, Thomas Meyer and Oscar Nierstrasz. Breaking and Fixing SAML: An Evaluation of the OpenSAML Library. *ACM Transactions on Privacy and Security (TOPS)*, 22(3):1–38, 2019.
- [doc] Docker: Empowering App Development for Developers. <https://www.docker.com/>. Last access on May 21, 2023.
- [Eli19] Elisa Bertino, Md. Nazrul Islam and Shaghayegh Vakiliinia. Security Analysis of OAuth 2.0 Authorization Framework. In *2019 IEEE 5th World Conference on Information Systems and Technologies (WorldCIST)*, pages 313–322, 2019.
- [Eri14] Eric Y. Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher and Patrick Tague. OAuth demystified for mobile application developers. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 892–903, 2014.
- [Eth15] Ethan Shernan, Henry Carter, Dave Tian, Patrick Traynor and Kevin Butler. More guidelines than rules: CSRF vulnerabilities from noncompliant OAuth 2.0 implementations. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 239–260. Springer, 2015.
- [fac] Facebook Whitehat. <https://www.facebook.com/whitehat>. Last access on May 21, 2023.
- [fapa] Financial-grade API (FAPI). <https://fapi.openid.net/>. Last access on May 21, 2023.
- [fapb] OpenID FAPI conformance suite. <https://gitlab.com/openid/conformance-suite>. Last access on May 21, 2023.
- [FBK] FBK. Fondazione Bruno Kessler. <https://www.fbk.eu/en/>. Last access on May 21, 2023.
- [Fre04] Frederick Hirsch, Rob Philpott and Eve Maler. Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0. *Committee Draft*, 1, 2004.
- [Gar13] Gary Nilson, Kent Wills, Jeffrey Stuckman and James Purtilo. BugBox: A Vulnerability Corpus for PHP Web Applications. In *6th Workshop on Cyber Security Experimentation and Test (CSET 13)*, 2013.
- [git] GitHub - The world's leading software development platform. <https://github.com/>. Last access on May 21, 2023.

- [gooa] Google OAuth 2.0 Playground. <https://developers.google.com/oauthplayground/>. Last access on May 21, 2023.
- [goob] Google Scholar. <https://scholar.google.com/>. Last access on May 21, 2023.
- [gooc] Google Vulnerability Reward Program (VRP). <https://www.google.com/about/appsecurity/reward-program/>. Last access on May 21, 2023.
- [GTB⁺20] Sérgio Manuel Nóbrega Gonçalves, Alessandro Tomasi, Andrea Bisegna, Giulio Pellizzari, and Silvio Ranise. Verifiable contracting-a use case for onboarding and contract offering in financial services with eidas and verifiable credentials. In *DETIP-S/DeSECSys/MPS/SPOSE@ ESORICS*, pages 133–144, 2020.
- [Han21] Hanlin Wei, Behnaz Hassanshahi, Guangdong Bai, Padmanabhan Krishnan and Kostyantyn Vorobyov. MoScan: a model-based vulnerability scanner for web Single Sign-On services. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 678–681, 2021.
- [Har] Dick Hardt. The OAuth 2.0 Authorization Framework. <https://tools.ietf.org/html/rfc6749>. Last access on May 21, 2023.
- [Har18] Dick Hardt. The OAuth 2.0 Authorization Framework: Improving Security and Usability. *IEEE Security & Privacy*, 16(4):46–53, 2018.
- [HM05] John Hughes and Eve Maler. Security assertion markup language (SAML) v2.0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, pages 29–38, 2005.
- [HPM] Frederick Hirsch, Rob Philpott, and Eve Maler. Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0. <https://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf>. Last access on May 21, 2023.
- [IET12] IETF. The OAuth 2.0 Authorization Protocol: Bearer Tokens. Technical report, RFC 6750, 2012.
- [IET14] IETF. The OAuth 2.0 Threat Model and Security Considerations. Technical report, RFC 6819, 2014.
- [Jen17] Jennifer Jurreit, Patrik Fehrenbach and Friedbert Kaspar. Analysis of Security Vulnerabilities in Microsoft Office 365 in Regard to SAML. *informatikJournal*, page 127, 2017.
- [Jia19] Jiawei Zhang, Xinyu Wang, Qiaoyan Wen and Jun Gao. Security Analysis of Redirect URI in OAuth 2.0. In *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 375–381. IEEE, 2019.
- [Jon] Jon Barber. SAMLyze. <https://www.blackhat.com/us-15/arsenal.html#jon-barber>. Last access on May 21, 2023.

- [Jos21] Josep Alonso, Frederic Aymerich, Jordi Herrera-Joancomartí and Xavier Franch. A Systematic Literature Review of JSON Web Token (JWT) Security. *Journal of Information Security and Applications*, 58:102859, 2021.
- [JPD] Bret Jordan, Rich Piazza, and Trey Darley. STIX Version 2.1 - Committee Specification. <https://docs.oasis-open.org/cti/stix/v2.1/cs01/stix-v2.1-cs01.html>. Last access on May 21, 2023.
- [Kar19] Karin Sumongkayothin, Pakpoom Rachtrachoo, Arnuphap Yupuech and Kasidit Siriporn. OVERSCAN: OAuth 2.0 Scanner for Missing Parameters. In *International Conference on Network and System Security*, pages 221–233. Springer, 2019.
- [KM06] Muhammad N. Khan and Qutaibah Malluhi. Information security risk assessment: A practical approach. *Journal of Information Privacy & Security*, 2(2):35–54, 2006.
- [LBLF] Torsten Lodderstedt, John Bradley, Andrey Labunets, and Daniel Fett. Security Considerations OAuth (accessed june 23, 2020). <https://tools.ietf.org/id/draft-bradley-oauth-jwt-encoded-state-08.html#rfc.section.6>.
- [Lun] Rich Lundeen. The Deputies are Still Confused. https://www.youtube.com/watch?v=_LEvuF_105A.
- [Luy13] Luyi Xing, Yangyi Chen, Xiaofeng Wang and Shuo Chen. InteGuard: Toward Automatic Protection of Third-Party Web Service Integrations. In *NDSS*, 2013.
- [Mar00] Martin Höst, Björn Regnell and Claes Wohlin. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, 5(3):201–214, 2000.
- [Mar14] Mariano Ceccato, Massimiliano Di Penta, Paolo Falcarin, Filippo Ricca, Marco Torchiano and Paolo Tonella. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. *Empirical Software Engineering*, 19(4):1040–1074, 2014.
- [mic] Microsoft Bug Bounty Program. <https://www.microsoft.com/en-us/msrc/bounty>. Last access on May 21, 2023.
- [Mic15] Michael B. Jones, John Bradley and Nat Sakimura. JSON web token (JWT). RFC 7519. *Internet Engineering Task Force*, 2015.
- [Mic16] Michael Felderer, Matthias Büchler, Martin Johns, Achim D Brucker, Ruth Breu and Alexander Pretschner. Security testing: A survey. In *Advances in Computers*, volume 101, pages 1–51. Elsevier, 2016.
- [Mic21] Michele Benolli, Seyed Ali Mirheidari, Elham Arshad and Bruno Crispo. The Full Gamut of an Attack: An Empirical Analysis of OAuth CSRF in the Wild. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 21–41. Springer, 2021.

- [Mik08] Mikael Svahnberg, Aybüke Aurum and Claes Wohlin. Redirect URI attack. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 288–290, 2008.
- [Min] Ministero Dell’Interno. Carta di identità elettronica. <https://www.cartaidentita.interno.gov.it>.
- [MMGS15] Christian Mainka, Vladislav Mladenov, Tim Guenther, and Jörg Schwenk. Automatic Recognition, Processing and Attacking of Single Sign-On Protocols with Burp Suite. In *Open Identity Summit 2015*. Gesellschaft für Informatik eV, 2015.
- [Moh18] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich and Jason Polakis. O Single Sign-Off, Where Art Thou? An Empirical Analysis of Single Sign-On Account Hijacking and Session Management on the Web. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1475–1492, 2018.
- [MSP18] Shreya Malviya, Naveen Saxena, and Dongwon Park. Exploiting oauth 2.0 implicit grant: A case study. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 79–84. IEEE, 2018.
- [Nat14] Nat Sakimura, John Bradley, Mike Jones, Breno De Medeiros and Chuck Mortimore. OpenID Connect Core 1.0 incorporating errata set 1. *The OpenID Foundation, specification*, 335, 2014.
- [Nil18] Nils Engelbertz, Nurullah Erinola, David Herring, Juraj Somorovsky, Vladislav Mladenov and Jörg Schwenk. Security Analysis of eIDAS—the Cross-Country Authentication Scheme in Europe. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018.
- [OAS08] OASIS. SAML V2.0 Tech. Overview. <http://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf>, March 2008.
- [oaua] OAuth 2.0 Playground. <https://www.oauth.com/playground/>. Last access on May 21, 2023.
- [oaub] OAuth 2.0 Security Best Current Practice. <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics-08>. Last access on May 21, 2023.
- [oauc] OAuth Security Workshop. <https://oauth.secworkshop.events/>. Last access on May 21, 2023.
- [oED13] Italian Ministry of Economic Development. Decreto del presidente del consiglio dei ministri 3 aprile 2013, 2013.
- [Off15] Official Journal of the European Union. PSD2. <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32015L2366>, December 2015. Last access on May 21, 2023.
- [Ope] OpenID (OIDF). OpenID foundation. <https://openid.net/foundation>. Last access on May 21, 2023.

- [Ope14a] OpenID Foundation. OpenID Connect Core 1.0, 2014.
- [Ope14b] OpenID Foundation. OpenID Connect Discovery 1.0, 2014.
- [oST12] National Institute of Standards and Technology. *Risk Management Guide for Information Technology Systems*. U.S. Department of Commerce, 2012.
- [oST20] National Institute of Standards and Technology. *Security and Privacy Controls for Federal Information Systems and Organizations*. NIST Special Publication 800-53, Revision 5, 2020.
- [Pau17] Paul Koppen, Martin Stepanek, Tomas Rybar and Dusan Klinec. Your Single Sign-On Service Might Be Leaving You Logged In To Sites You Never Knew You Had An Account With. In *Proceedings of the 26th USENIX Security Symposium*, pages 1023–1038, 2017.
- [PB14] Giancarlo Pellegrino and Davide Balzarotti. Toward black-box detection of logic flaws in web applications. In *NDSS*, volume 14, pages 23–26, 2014.
- [Pie] Pieter Philippaerts. OAuch: Analyzing the Security Best Practices in the OAuth 2.0 Ecosystem. <https://www.youtube.com/watch?v=eaFQFm1K5yI/>. Last access on May 21, 2023.
- [Pol] Poligrafico e Zecca dello Stato Italiano. <https://www.ipzs.it>.
- [PPJ22] Davy Preuveneers Pieter Philippaerts and Wouter Joosen. OAuch: Exploring Security Compliance in the OAuth 2.0 Ecosystem. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 460–481, 2022.
- [PS18] Vikrant Prasad and Sachin Shukla. SAML Raider: A Burp Suite Extension for SAML Security Testing. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1151–1156. IEEE, 2018.
- [Ric15] Justin Richer. OAuth 2.0 Token Introspection. Technical report, Internet Engineering Task Force (IETF), 2015.
- [Rob10] Roberto Ortiz, Santiago Moral-García, Santiago Moral-Rubio, Belén Vela, Javier Garzás and Eduardo Fernández-Medina. Applicability of security patterns. In *On the Move to Meaningful Internet Systems: OTM 2010: Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Hersonissos, Crete, Greece, October 25-29, 2010, Proceedings, Part I*, pages 672–684. Springer, 2010.
- [Rob20] Robert Krauthgamer, Jonathan Shapiro and Ariel Stein. OAuth 2.0: An Inside Look at Missing Best Practices. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1075–1091. IEEE, 2020.
- [Ron16] Ronghai Yang, Guanchen Li, Wing Cheong Lau, Kehuan Zhang and Pili Hu. Model-based security testing: An empirical study on OAuth 2.0 implementations. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 651–662, 2016.

- [Ron18] Ronghai Yang, Wing Cheong Lau, Jiongyi Chen and Kehuan Zhang. Vetting Single Sign-On SDK Implementations via Symbolic Reasoning. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1459–1474, 2018.
- [Rui12] Rui Wang, Shuo Chen and XiaoFeng Wang. Signing me onto your accounts through Facebook and Google: A traffic-guided security study of commercially deployed Single-Sign-On web services. In *2012 IEEE Symposium on Security and Privacy*, pages 365–379. IEEE, 2012.
- [Rya14] Ryan Holmes, Jörg Schwenk, Nitesh Kumar, Neeraj S. Jain and Jack Loftus. Security analysis of the OAuth 2.0 framework. In *2014 IEEE 27th Computer Security Foundations Symposium*, pages 280–294. IEEE, 2014.
- [Ryu16] Ryuichi Hasegawa, Takuya Shimada and Yoshio Takatori. Exploiting OAuth 2.0 to perform authorization code interception attacks. In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 139–147. IEEE, 2016.
- [Sah16] Sahil Garg, Prithvi Bisht, Sruthi Venkataraman and Pradeep Mohapatra. Countermeasures against Token Replay Attacks in OAuth 2.0. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, pages 1733–1744, 2016.
- [San12] San-Tsai Sun and Konstantin Beznosov. The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 378–390, 2012.
- [SAW08] Mikael Svahnberg, Aybüke Aurum, and Claes Wohlin. Using students as subjects—an empirical evaluation. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 288–290, 2008.
- [sel] Selenium WebDriver. <https://www.selenium.dev/>. Last access on May 21, 2023.
- [Sho14] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.
- [SMJ15] Iftaah Salman, Ayse Tosun Misirli, and Natalia Juristo. Are students representatives of professionals in software engineering experiments? In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 666–676. IEEE, 2015.
- [Sof] Skybound Software. GeckoFX. <https://github.com/priyank/GeckoFX>. Last access on May 21, 2023.
- [Sta14] Stanislav Dashevskiy, Daniel Ricardo Dos Santos, Fabio Massacci and Antonino Sabetta. TESTREX: a Testbed for Repeatable Exploits. In *7th Workshop on Cyber Security Experimentation and Test (CSET 14)*, 2014.
- [Ste17] Stefano Calzavara, Mauro Conti, Silvia Giordano and Michele Pagano. A Survey of Network Anomaly Detection Techniques. *IEEE Communications Surveys & Tutorials*, 19(4):2393–2420, 2017.

- [Ste18a] Stefano Calzavara, Mauro Conti and Nicola Dragoni. Surviving the Web: A Journey into Web Session Security. In *International Conference on Availability, Reliability, and Security*, pages 1–12. Springer, 2018.
- [Ste18b] Stefano Calzavara, Riccardo Focardi, Matteo Maffei, Clara Schneidewind, Marco Squarcina and Mauro Tempesta. WPSE: Fortifying Web Protocols via Browser-Side Security Monitoring. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1493–1510, 2018.
- [Suh11] Suhas Pai, Yash Sharma, Sunil Kumar, Radhika M. Pai and Sanjay Singh. Formal Verification of OAuth 2.0 Using Alloy Framework. In *Proceedings of the 2011 International Conference on Communication Systems and Network Technologies*, CSNT '11, pages 655–659, USA, 2011. IEEE Computer Society.
- [Sur11] Suresh Chari, Charanjit Jutla and Arnab Roy. Universally Composable Security Analysis of OAuth v2.0. *IACR Cryptology ePrint Archive*, 2011:526, 01 2011.
- [Tam21] Tamjid Al Rahat, Yu Feng and Yuan Tian. OAuthShield: Efficient Security Checking for OAuth Service Provider Implementations. *arXiv preprint arXiv:2110.01005*, 2021.
- [Tre17] Trevor J. Hastie and Daryl Pregibon. Generalized linear models. In *Statistical models in S*, pages 195–247. Routledge, 2017.
- [Uni14] European Union. Regulation (eu) no 910/2014 of the european parliament and of the council of 23 july 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing directive 1999/93/ec, 2014.
- [Vic18] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. *arXiv preprint arXiv:1806.01156*, 2018.
- [Vla] Vladislav Mladenov, Christian Mainka and Florian Feldmann. Verification of SAML Tokens - Traps and Pitfalls. <https://web-in-security.blogspot.com/2014/10/verification-of-saml-tokens-traps-and.html>.
- [Wan14] Wanpeng Li and Chris J. Mitchell. Security issues in OAuth 2.0 SSO implementations. In *International Conference on Information Security*, pages 529–541. Springer, 2014.
- [Wan19] Wanpeng Li, Chris J. Mitchell and Thomas Chen. OAuthGuard: Protecting User Security and Privacy with OAuth 2.0 and OpenID Connect. In *Proceedings of the 5th ACM Workshop on Security Standardisation Research Workshop*, pages 35–44, 2019.
- [web] WebGoat. <https://owasp.org/www-project-webgoat/>. Last access on May 21, 2023.
- [wsd] Web Services Description Language (WSDL) 1.1. <https://www.w3.org/TR/wsdl.html>. Last access on May 21, 2023.
- [Xia11] Xiaowei Li and Yuan Xue. BLOCK: a black-box approach for detection of state violation attacks towards web applications. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 247–256, 2011.

- [Yan16] Yanbin Wang, Xiaogang Wang and Pengwei Guo. Web Application Security Analysis of OAuth 2.0 Framework. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 377–384. IEEE, 2016.
- [Yue19] Yue Wang, Pengfei Xie, Wenjuan Liu, Xiaofeng Zhang and Kai Lu. State Parameter Pollution Vulnerability in OAuth 2.0. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2061–2072. IEEE, 2019.
- [zap] OWASP Zed Attack Proxy. <https://www.zaproxy.org/>. Last access on May 21, 2023.
- [ZE14] Yuchen Zhou and David Evans. SSOScan: Automated testing of web applications for Single Sign-On vulnerabilities. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 495–510, 2014.