Article Type: Description (see below for more detail)

Enhancing Security Testing for Identity Management Implementations: Introducing MIG-L and MIG-T

Andrea Bisegna, *Fondazione Bruno Kessler, Trento, TN, 38123, Italy*

Matteo Bitussi, *Fondazione Bruno Kessler, Trento, TN, 38123, Italy*

Roberto Carbone, *Fondazione Bruno Kessler, Trento, TN, 38123, Italy*

Silvio Ranise, *Fondazione Bruno Kessler and Department Mathematics - UniTN, Trento, TN, 38123, Italy*

Abstract—We introduce MIG-L, a declarative language for the specification of security tests, and MIG-T, a testing tool, for Identity Management solutions based on SAML and OAuth/OIDC by verifying compliance with Best Current Practices, detecting known vulnerabilities, and providing suggestions for fixes. Experiments demonstrate the flexibility and effectiveness of our approach.

Index Terms: *Testing tools, testing strategies, authentication.*

Multi-Party Web Applications (MPWA) plays a crucial role in building trust in digital ecosystems by adopting Identity Management (IdM) protocols to secure their implementations. IdM protocols involve three entities: the User (typically interacting through a web browser), the web application (playing the role of a Client), and an Identity Provider (IdP) acting as a trusted third party. Standards for IdM protocols are available (e.g., the Security Assertion Markup Language 2.0 (SAML),[1] OIDC,[2] and OAuth[3]) that describe how Clients can request and consume assertions from IdPs to authenticate users via Single Sign-On (SSO) procedures. This allows users to access multiple applications or services using a single set of credentials while providing a streamlined user experience and increasing security by reducing password fatigue. Despite the advantages, several vulnerabilities have been and are still being found, exposing potential errors and security risks inherent in design and implementation [1], [6], [7]. The complexity of protocols and reliance on third-party entities further exacerbate the risks. Privacy and data protection concerns necessitate meticulous consideration during deployment. Fragmentation of information sources and a dearth of comprehensive remediation strategies contribute to the complexity of securing these systems. While security testing tools are available, they often exhibit a narrow focus on specific vulnerabilities, potentially overlooking broader security concerns [8], [9], [10], [11]. Moreover, administrators may lack the expertise required to address vulnerabilities effectively, particularly for average IT professionals. To alleviate these problems, we propose a declarative language for security testing, providing a straightforward method to define and execute security test cases in the context of IdM deployments by making the following main contributions:

- We consider an existing threat model [4] based on security controls, threats, vulnerabilities, and risks, and propose an extended version. The extended version allows not only to perform security tests to identify known vulnerabilities but also to check compliance with Best Current Practices (BCPs) put forward by standardization efforts (e.g., OIDC) to avoid recurring vulnerabilities with high risk.
- We introduce a novel approach for the specification of the test cases leveraging a new declarative language named MIG-L. The language is tailored specifically for IdM protocols and is built upon the extended threat model previously defined.

XXXX-XXX © 2024 IEEE
Digital Object Identifier 10.1109/XXX.0000.0000000

[1]https://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html

[2]https://openid.net/connect

[3]https://tools.ietf.org/html/rfc6749

- We automate our approach by integrating MIG-L into MIG-T[4], a security testing tool included in Micro-Id-Gym [5].
- For validation, we conducted experiments with MIG-T in three scenarios involving IdM protocols. In an OIDC Federation, we identified three vulnerabilities. In an OAuth implementation for a PSD2 compliant payment service, we detected two vulnerabilities and a misconfiguration in the OAuth protocol. Among 48 pairs of service and identity providers supporting SSO- based Linking Account, we identified 21 pairs vulnerable to CSRF.

We make available the security tests specified in MIG-L and the code of MIG-T at the following links https://github.com/stfbk/mig/tree/master/testplans/spid-cie-oidc/implementations/spid-cie-oidc-django/input/mig-t and https://github.com/stfbk/mig-t, respectively.

Plan of the paper.
In Section OUR THREAT MODEL we propose an extended threat model based on an existing one. Then, in Section OUR APPROACH, we present our approach to support security testing. In Section TEST SPECIFICATION, we delve into the specification to define test cases. In Section USE CASES, we describe the experimental analysis on three different scenarios. In Section COMPARISON WITH AVAILABLE TOOLS we collect state-of-the-art tools covering our research. We conclude our work in Section CONCLUSIONS.

OUR THREAT MODEL

The threat model we propose supports the automation of security tests, specified in a high-level language, for both verifying the proper implementation of BCPs and performing attacks on deployments based on IdM protocols. Additionally, the proposed threat model supports by automation the use of structured data security assessments and facilitates informed decision-making for cybersecurity risk management. By leveraging automation, we can expedite the analysis process and enhance result accuracy, ensuring the delivery of up-to-date and comprehensive assessments.

By returning actionable hints for fixing vulnerabilities, we assist IT professionists to increase the security of IdM deployments. In Figure 1, the threat model derived from [4] is shown in the dashed box whereas outside are the concepts of the extended threat model that are discussed in Section OUR APPROACH. It consists in identifying potential threats and vulnerabilities as well as determining the appropriate security controls to mitigate these risks. Security controls encompass a range of countermeasures implemented to secure against intentional and unintentional threats. Vulnerabilities represent implementation and configuration flaws that could be exploited by threats. Understanding the likelihood and impact of these threats is essential for effective risk management. Cyber Threat Intelligence (CTI) plays a crucial role in helping organizations identify and prioritize potential risks. Moreover, CTI aids in the development of targeted risk management strategies tailored to specific vulnerabilities, thus improving overall security posture. We then introduce an extended threat model for IdM implementations, depicted in Figure 1. Our extended threat model incorporates known attacks and BCPs outlined in the IdM standards, ensuring a comprehensive coverage of security issues.

From our threat model, we derive two types of test cases. The former assess the proper implementation of security controls outlined in standards like OIDC or OAuth, facilitating automated compliance testing. The latter perform attacks to exploit known vulnerabilities, enabling automated security testing. By leveraging these test cases, it is possible to identify recurring vulnerabilities and evaluate the security posture of the system through targeted, (known) attacks. The purpose of BCPs, implemented through Security Controls, is to identify key vulnerabilities and provide structured mitigations, Security testers design test cases that assess these measures and make sure they are implemented correctly. Let us consider a test *(Test 1, T1)* aiming at checking whether the adoption of Proof Key for Code Exchange (PKCE) is in place for an OAuth/OIDC deployment. PKCE mitigates the possibility of unauthorized access to protected resources.[5] Security Testers also play the role of attackers to assess the impact of vulnerabilities by performing known attacks. By employing techniques such as Common Vulnerabilities and Exposures (CVE) analysis, these methods offer insights into associated risks, assisting in the decision-making process for incorporating mitigation strategies. For example, in a “Code replay" attack, the attacker intercepts and reuses the authorization response to gain access to a user’s resources without their knowledge *(Test 2, T2)* [3]. The extended threat model considers the nature of BCPs and attacks, which can change over time, offering the possibility of easily expressing

[4]https://github.com/stfbk/mig-t

[5]https://tools.ietf.org/html/rfc7636

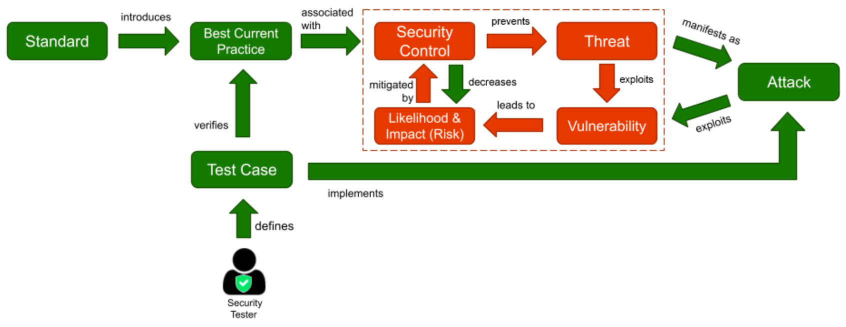

FIGURE 1: Our extended threat model.

and incorporating new test cases in response to a rapidly changing threat landscape. Security Testers have two options according to the type of test case they execute. The first capability pertains to verifying BCPs, by assuming network or web attacker capabilities, depending on the specific BCP being tested. For example, BCPs addressing redirect Universal Resource Identifier (URI) attacks in OAuth require the capabilities of a web attacker, while others, like securing the OAuth token exchange process, require the capabilities of a (more powerful) network attacker. The second option relates to executing attacks, where Security Testers have the capabilities of a web attacker. For instance, in the authorization code interception attack, Security Testers intercept the authorization code exchanged between the OAuth client and authorization server. This attack can be executed through phishing or exploiting vulnerabilities in the OAuth client application, allowing attackers to obtain access tokens and access protected resources.

OUR APPROACH

According to the threat model in Figure 1, a Security Tester is responsible for creating two types of test cases: one to verify the compliance with the BCPs, and the other to perform attacks. For each test case, a Security Tester defines both a Session and a Test as reported in Figure 2, which provides an overview of our approach. The Session is similar to a UI integration test typically employed by testers of web applications.[6] Indeed, Session encompasses a series of user actions executed by a browser to trigger the HTTP messages required for a Test, enabling our approach to simulate user interactions and gather information on the web application's response. The Test, written in MIG-L, a formal and concise declarative language for security testing, consists of a sequence of operations aimed at managing and manipulating HTTP messages and Session, as well as integrating an oracle automating the criterion against which the outcome of a test can be evaluated. MIG-L is designed to be versatile and adaptable across all web digital identity protocols, as its underlying principles and structure are sufficiently generic to support them. To ensure this flexibility, MIG-L enables security testers to define tests for any web digital identity protocol by specifying relevant parameters and interactions. A Test can be passive or active. The former involves analyzing the intercepted HTTP messages statically, without any interaction (saving a value of a parameter) or modification of the HTTP messages during the execution of the Session. An active test allows for interactions during the execution of the Session. By automating the execution of the test case, a Security Tester can rapidly and precisely evaluate the security of the IdM implementation.

As depicted in Figure 2, the Test Handler is responsible for interpreting and executing the test specified in MIG-L, and it triggers, when needed, the Session Handler, which processes the Session and replicates the user actions within a browser. All HTTP messages pass through (and may be intercepted by) a Proxy. The Proxy Interaction component manages the communication with the Proxy to enforce the conditions specified in the Test. Finally, the Reporting component provides output detailing identified vulnerabilities and suggestions for appropriate security controls, aiding Security Testers and stakeholders in understanding and addressing the results of the tests, thus paving the way to cybersecurity risk management.

To define an architecture for the execution of security tests specified in MIG-L, we define three distinct APIs used by components independently of the underlying technology. These APIs—the Browser API, Proxy API, and Session API—support the automated execution of user actions, interception and manipula-

[6]https://www.divami.com/blog/selenium-guide-to-automated-ui-testing/

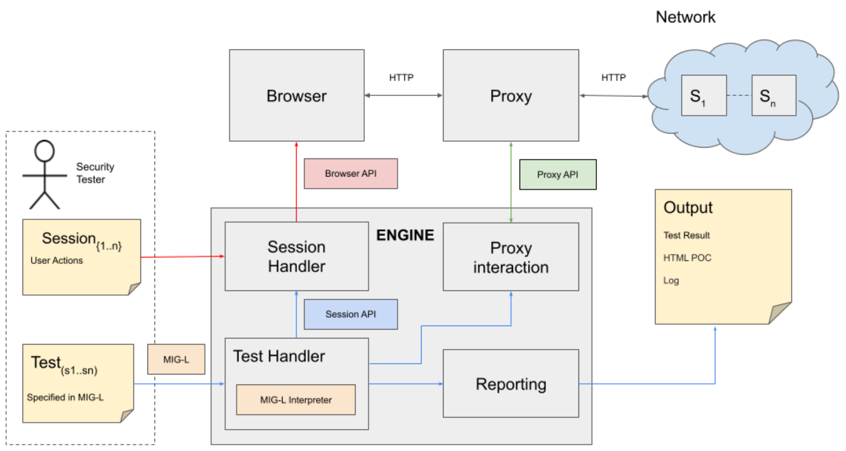

FIGURE 2: High level view of our approach.

tion of messages, and management of test session executions, respectively. They streamline the testing process and enhance the flexibility and usability of our approach.

TEST SPECIFICATION

MIG-L is a declarative language for security testing, encompassing many command combinations for manipulating HTTP messages or interacting with the Session. One of the key strengths of MIG-L is its user-friendly nature. The language is designed to be highly expressive yet intuitive, making it accessible to security testers with limited programming experience. The declarative nature of MIG-L allows testers to specify test scenarios in a concise manner without delving into complex scripting or programming. To further simplify the process, we provide a library of pre-defined test templates for common test patterns, which can be easily adapted to specific protocols and implementations. Additionally, MIG-L can be used to generate tests for other types of testing, including triggering additional security tests for the implementation, such as fuzzing, though this is not considered in this work.

To illustrate the flexibility of this language we present two test cases, T1 and T2. These examples correspond to the BCP and attack cases described in Section OUR THREAT MODEL. Our approach, as defined in Section OUR APPROACH, requires two inputs: the Test specified in MIG-L and the Session.

The Session depicted in Listing 1 and required for both tests (T1 and T2) is a Selenium script that executes the OIDC flow in the spid-cie-oidc-django[7] implementation based on the SPID/CIE OIDC Federation. The commands inherited by the Selenium script are highlighted in red, while comments are in blue.

```
open | http://relying-party.org:8001/
  oidc/rp/landing | // access SP webpage
click | xpath=/html/body/div[2]/div/span
  [2]/a | // press login button
click | xpath=/html/body/div[2]/ul/li
  [2]/a | // choose IdP
type | id=username | user // insert user
type | id=password | oidcuser // insert
  password
click | xpath=/html/body/div[2]/div[3]/
  button/span[2] | // press login button
click | id=agree | // provide consent
```

Listing 1: Session used in T1 and T2.

The first test we analyze, T1, concerns compliance with the BCP in OAuth, specifically focusing on the adoption of PKCE and show in Listing 2. This test intercepts the authorization request and verifies the presence of `code_challenge` and `code_verifier` in the URL.
The commands `start_test` and `end_test` mark the beginning (Line 1) and end (Line 11) of the test, respectively. The `start_test` command requires the test name and description, the test type (either passive or active), the set of (references to) Sessions (only `s1` for *T1*).
The command `start` (at Line 2) indicates that the Session s1 is executed.
The commands `start_msg_operation` and `end_msg_operation` mark the beginning (Line

[7]https://github.com/italia/spid-cie-oidc-django/

3) and the end (Line 10) of the HTTP message operations to be executed within the test, respectively. The commands start_checks and end_checks identify the beginning (Line 4) and end (Line 6) of an operation in charge of verifying the presence of a parameter, respectively. In this case we detect the presence of the code_challenge parameter in the URL of the authorization request. The authorization_request is defined as the HTTP request where the URL contains the response_type and client_id parameters.
Similarly to Lines 4-6, the commands in Lines 7-9 verify the presence of the parameter code_verifier in the URL of the token request. The token_request is defined as the HTTP request where the URL contains grant_type, code, redirect_uri, and client_id.

```
1  start_test(Verify presence of PKCE, The
       test verify the presence of
       code_challenge and code_verifier,
       passive, {s1})  // passive test
2   start(s1) // run s1
3   start_msg_operation()  // start of
       message operation
4    start_checks(authorization_request) //
       start checks in authorization_request
5     check(code_challenge, is present, url)
       // verify the presence of
       code_challenge in url
6    end_checks()
7    start_checks(token_request) // start
       checks in token_request
8     check(code_verifier, is present, url)
       // verify the presence of
       code_verifier in url
9    end_checks()
10  end_msg_operation()  // end of message
       operation
11 end_test() // end of passive test
```

Listing 2: MIG-L test for T1 (passive).

The second test T2 we show performs the code replay attack in [3]. The steps of this attack involve intercepting the code parameter during the execution of the OIDC flow and then reusing it in a new flow.

As said above, Session (s1) for T2 is identical to the one used in T1. Another Session (s1_copy) is responsible for reusing the code parameter in a new flow. This Session (s1_copy) is automatically generated and duplicated by the MIG-L test for T2.

Listing 3 contains the specification for T2.
The commands at Line 1 and 15 indicate the start and finish of the test, respectively. Unlike the test in Listing 2, we have defined an active test and provided two Sessions (s1 and s1_copy). For active tests, another information must be included, namely the expected test result (in this case, it is incorrect flow s1_copy, meaning that the test is passed if the flow of s1_copy is incorrect, i.e. the execution fails. Indeed, if all the user actions in s1_copy are successfully executed, it means that the attack is successful and thus the test is considered failed).
The command save (at Line 2) indicates that all the user actions reported from the command track[M0,ML], where M0 represents the first user action and ML represents the last user action reported in Session s1, are saved in s1_copy.
The command start (at Line 3) indicates that the Session s1 is executed.
The commands start_test_operation and end_test_operation (at lines 4 and 8, respectively) define which HTTP message to intercept and the action to take once intercepted. All the operations between lines 5 and 7 will be applied to the HTTP message. In this case, the HTTP message to intercept is the authorization_response in s1, and once intercepted, it must be dropped. This means inducing an error in the execution and thus halting the execution of s1. The authorization_response is defined as the HTTP response with code and state in the URL.
The commands start_msg_operation and end_msg_operation identify the beginning (Line 5) and end (Line 7) of an operation in charge of saving the value of a parameter, respectively. In this case, we identify the value of the parameter code in the URL and save it to a new variable called saved_code.
The command start (at Line 9) indicates that the Session s1_copy is executed.
Similarly to Lines 4 and 8, with the command start_test_operation and end_test_operation (at Lines 10 and 14, respectively) case, we want to intercept the authorization_response message from s1_copy without dropping the HTTP message, allowing the execution of the remaining user actions specified in s1_copy.
The commands start_msg_operation and end_msg_operation (at Lines 11 and 13, respectively) replace the value of the code parameter in the URL with that in saved_code.

```
1  start_test(Code replay attack, The test
     intercepts the code in the
     authentication response and replay it
     in another authentication response,
     active, {s1, s1_copy}, incorrect flow
     s1_copy)  // active test
2  save(s1_copy, track[M0,ML],  s1) // copy
      all the user actions from s1 to
     s1_copy
```

```
start(s1) // run s1
start_test_operation(oauth_response, s1,
   drop) // drop oauth_response message
  generated in s1
start_msg_operation() // begin message
  operation
save_parameter(code, saved_code, url) //
   save parameter code from url and
  store it as saved_code
end_msg_operation() // end message
  operation
end_test_operation() // end test
  operation
start(s1_copy) // run s1_copy
start_test_operation(oauth_response,
  s1_copy, intercept) // intercept
  oauth_response message from s1_copy
start_msg_operation() // begin message
  operation
edit_parameter(code, saved_code, url) //
   edit code parameter from url with
  saved_code
end_msg_operation() // end message
  operation
end_test_operation() // end test
  operation
end_test() // end of active test
```

Listing 3: MIG-L test for T2 (active).

With our language, it is also possible to execute multiple tests simultaneously using the same Session. For this, we have introduced the ability to define a test suite, which consists of an object comprising a collection of tests, using the command `define_suite`. This command specifies *(i)* name - the label assigned to the suite, and *(ii)* description - the descriptive annotation allocated to it. Once the test suite is defined, tests can be defined.

Additionally, MIG-L can be used to generate tests for detecting other types of vulnerabilities (e.g., those detectable by fuzzing). Although we do not further elaborate on this point, we mention that it is not difficult to specify a test case in MIG-L to detect a recently disclosed vulnerability in the OAuth standard that, paired with cross-site scripting (XSS) flaws in the two sites, enables account takeovers in more than a million websites.[8]

IMPLEMENTATION

We have implemented our approach in Micro-Id-Gym [5] by extending MIG-T, a plugin to support pentesting activities in IdM implementations.

Based on the Burp web proxy,[9] MIG-T offers a comprehensive set of APIs to interact with, including those defined in Section OUR APPROACH. The three APIs available in MIG-T are the Browser API, Proxy API, and Session API and are instances of those depicted in Figure 2. The Browser API allows for the automated execution of user actions defined 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. For instance, the `HttpRequestResponse.getRequest()` method can retrieve intercepted messages. Finally, the Session API manages Session executions. For example, the `start(`Session`)` method creates a new thread object to run a specified Session.

USE CASES

We illustrate three applications of MIG-T that highlight its versatility and effectiveness by conducting the security analysis in (S1) the SPID/CIE OIDC deployment in Developers Italia, (S2) an OAuth deployment for a PSD2 service, and (S3) 48 pairs of SP-IdP supporting SSO-based Account Linking.

For all the experiments reported in the following, we use JSON as a concrete syntax of MIG-L as described in Section TEST SPECIFICATION. The reasons underlying this choice are practical, namely to simplify the development of the parser and interpreter of the language. We will soon implement a parser from the syntax presented in this work to the JSON syntax.

S1. SPID/CIE OIDC for Developers Italia Deployment

In the context of a long term collaboration with the Italian National Mint and Printing Office (Istituto Poligrafico e Zecca dello Stato, IPZS), we contributed to the development of an extensive corpus of compliance and security tests for deployments of the national digital identity solutions based on the Italian electronic identity card (Carta d'Identità Elettronica, CIE) which use the OIDC standard. The collection of the test cases has been made available on the Github web site (at https://github.com/stfbk/mig/tree/master/testplans/spid-cie-oidc/implementations/spid-cie-oidc-django/input/mig-t) which offers resources to the community of developers who design and develop code for Italian digital public services.

[8] https://salt.security/blog/over-1-million-websites-are-at-risk-of-sensitive-information-leakage—xss-is-dead-long-live-xss

[9] https://portswigger.net/burp/documentation/desktop/tools/proxy

The security analysis consisted of a set of tests, with a particular focus on OIDC, aimed at conducting a comprehensive security evaluation of both the OpenID Provider (OP) and Relying Party (RP) (available at https://github.com/italia/spid-cie-oidc-django) implementations using MIG-T. The objective was to confirm their adherence to the SPID/CIE OIDC standards.

After running MIG-T against the OP, two issues were identified in the Authentication Request:

- The `client_id` and `response_type` parameters could be edited and removed, contrary to the requirement that they should be present both as query parameter and inside the JWT `request` object.
- The presence and correctness of the `scope` parameter in the URL were not properly validated, despite the requirement that it must be sent as query parameter and inside the JWT `request` object, with both values being the same. This mismatch occurred because the JWT `request` object suppressed the URL values.

Regarding the RP test results, one issue was identified during the analysis with MIG-T:

- The `iss` parameter was removed from the Authentication Response without triggering any error at the RP. Despite editing the `iss` parameter with an arbitrary value, the RP failed to validate it correctly. The lack of this validation leads to a Mix-Up Attack,[10] where an attacker can exploits a malicious Authorization Server (AS) to steal authorization codes or access tokens. Subsequently, the attacker can access the victim's resources. This issue was promptly addressed by Developers Italia in release version 0.7.1.[11]

By using MIG-T, the security posture of the reference implementations has been significantly enhanced. This provides a considerable advantage, as all those who reuse them to operate services for the Italian public administration will benefit. Similarly, all Italian citizens who utilize these services will be able to access them with greater security.

S2. OAuth for PSD2 Deployment

In 2020, during the activities of a European project, we considered assessing a PSD2 (the revised Payment Services Directive) service operated by a prominent Italian IdP.[12] The PSD2 regulation requires that banks operating across the EU expose standard open application programming interfaces (APIs) that enable their customers to securely share their account data with other banks and third-party providers after giving explicit consent. The ultimate goal is to provide customers with more choice, increase competition and stimulate innovation. OAuth seems to be the obvious choice for providing customers with a secure mechanism for delegating scoped access to third party services since it has emerged as the de facto standard for API secure interoperability.

This approach has been proposed by various entities, including the Berlin group. Adoption of OAuth could potentially revolutionize the payments industry, impacting everything from online payment methods to the information displayed during transactions. Ensuring the security of electronic payments is paramount for safeguarding users, and all electronically offered payment services should be implemented securely, utilizing technologies capable of guaranteeing trustworthy user authentication.

We conducted a security analysis by executing a defined set of tests focusing on OAuth, with the aim of evaluating the IdP deployment and ensuring compliance with the OAuth standard. Employing MIG-T, we performed the security assessment, through which we successfully identified two vulnerabilities and a misconfiguration of the OAuth protocol.

About compliance with BCPs, certain required parameters according to the OAuth standard were absent from the deployed protocol, while some optional parameters were included. Specifically, MIG-T identified issues with the `redirect_uri` and `state` parameters in the Authentication Request, which were not set properly and not verified by the IdP.

As regards the vulnerability in the `redirect_uri` parameter, it is possible that the AS, authorization endpoint, and client redirection endpoint can be improperly configured and operate as open redirects.[13] An open redirect occurs when an endpoint automatically redirects a user-agent to a specified location without validation, potentially leading to phishing attacks or malicious site visits. Additionally, if the AS only allows the client to register part of the redirection URI, an attacker could exploit an open redirect to send the authorization code or access token to an endpoint

[10] https://danielfett.de/2020/05/04/mix-up-revisited/

[11] https://github.com/italia/spid-cie-oidc-django/releases/tag/v0.7.1

[12] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32015L2366

[13] https://sec.okta.com/articles/2021/02/stealing-oauth-tokens-open-redirects

controlled by the attacker.

As for the `state` parameter, it is recommended in OAuth as an opaque value used by the Client to maintain state between the request and callback. If not adequately protected, it may be vulnerable to CSRF attacks, allowing attackers to associate their protected resources with a target user's session.[14] This vulnerability could be exploited by causing the target browser to send the target site a request containing the attacker's authorization code or access token, potentially leading to undesirable outcomes such as saving sensitive user data to an attacker-controlled location.

Phishing attacks, malicious site visits, and CSRF attacks may have potentially dramatic consequences including identity thefts, unauthorized transfer of funds, compromise of sensitive data or the user machine, and even the complete compromise of services and applications in case of victims with administrative privileges. It is thus crucial to adopt the appropriate mitigations, as suggested by OAuth standard documents, and improve the security posture of deployments. For this, we have shared our findings with the IdP and helped them to incorporate the mitigations suggested by MIG-T, including implementing strict uri validation to avoid the issues with the `redirect_uri` and employing robust cryptographic techniques to verify the integrity of the `state` parameter.

S3. SSO-based Linking Account Deployments

The SSO-based linking account process (SSOLinking) enables users to connect their accounts on SP websites to their IdP accounts.

In [1] we focus on a significant (yet often overlooked) attack, known as Account Hijack targeting the SSOLinking, that leverages two CSRF vulnerabilities—one affecting the IdP and the other the SP. The former vulnerability is an Authentication CSRF (also referred to as Login CSRF), while the latter is a CSRF vulnerability on the button triggering the SSOLinking. We conducted experimental analyses by using MIG-T on a selection of popular SPs offering the SSOLinking with major IdPs. The results of our experiments–performed in a responsible and non-intrusive manner by ensuring that the testing activities do not disrupt the normal operation of the systems under test and do not compromise user data or system integrity–are concerning: out of the 609 websites considered, 48 were eligible for our experiments, and 21 of these were found to be vulnerable to SSOLinking vulnerabilities (43.7%).

We contacted the vendors of vulnerable websites using either bug bounty platforms like HackerOne and Bugcrowd or direct contact details from their websites. For sites with established reporting protocols, we filed detailed reports. Communicating the complexities of the attack was challenging due to its dual vulnerabilities, but the responses were generally positive. Some vendors acknowledged the issues and provided rewards.

COMPARISON WITH AVAILABLE TOOLS

We compare MIG-T to available tools with capabilities to perform activities that are relevant to the security testing of web applications and IdM protocols in a semi automated way. As a result of an extensive survey, we consider 25 tools grouped in 4 main categories, namely (i) General purpose for tools covering security issues in web applications requiring user authentication (the tools in this category have not been tuned to detect vulnerabilities specific to IdM protocols), (ii) OAuth/OIDC and (iii) SAML for tools aimed to discover vulnerabilities in OAuth/OIDC and SAML deployments, respectively, (iv) OAuth/OIDC and SAML for tools targeting vulnerabilities in both OAuth/OIDC and SAML deployments. By considering the design of MIG and after conducting a careful analysis of the documentation of each tool and, when possible, their use, we consider the following 7 features to structure the comparison, namely (i) Automation (Auto) about the degree of automation to execute tests, (ii) Web Application Security (WAS) about the capability of identifying known vulnerabilities commonly found in web applications, (iii) Compliance (Compl) for verifying compliance with protocol standards, (iv) IdM for tools targeting vulnerabilities that are specific to IdM protocol deployments, (v) Active and (vi) Passive about the capability of executing active and passive tests (as defined in Section TEST SPECIFICATION), and (vii) Declarative for the availability of a language to specify security tests. Table 1 shows which features are provided (✓sign) and which are not (- sign) by each one of the selected 25 tools and the last row shows MIG-T.

From a cursory look at the table, two observations are immediate. First, MIG-T has all the 7 features considered. This is not an accident but it has been done on purpose to explain how our approach advances the state of the art. Second, the first and last columns are particular since all the tools feature automation and none, except MIG-T, supports a

[14] https://auth0.com/docs/secure/attack-protection/state-parameters

declarative language. Concerning the former, we emphasize that each tool offers a different degree of automation for different activities related to security testing. For instance, AUTHSCAN [10] analyzes HTTP traces and Javascript to abstract protocol specifications for the automated identification of vulnerabilities but it requires manual intervention to resolve inconsistencies and potential false positives. MIG, instead, automates all the activities related to the execution and evaluation of security tests once user actions have been specified in sessions and the actions to identify vulnerabilities or check the application of BCPs have been specified. In other words, a detailed comparison about the degree of automation offered turns out to be quite a challenging task given the heterogeneity of the approaches on which the tools are based. However, a fundamental limitation common to all the tools considered here, including MIG-T, is related to the adoption of automation-prevention techniques by top web applications to protect, in particular, registration and login procedures. Indeed, in the use of MIG-T, we encountered these techniques in the form of captchas and bot-detection libraries; for example, the latter can identify the use of Selenium libraries to automate browser actions and block access. How to overcome this kind of limitation and achieve a higher degree of automation in MIG-T is left to future work. Concerning the last column, we observe that the availability of a declarative language, MIG-L, to specify executable security tests is a distinctive feature of MIG-T. There are two main advantages in using a declarative language. On the one hand, it allows security testers to focus on few fundamental constructs to write compact specifications of complex test cases and simplifies the understanding of the test cases and the results by IT professionals who use them. On the other hand, it paves the way to the use of generative AI techniques to assist in the production of security tests, making available automated checks to identify vulnerabilities in rapidly evolving threat landscapes. We plan to investigate the use of AI techniques to synthesize security tests in MIG-L from semi structured descriptions of security tests in natural language (such as those made available at https://github.com/stfbk/mig/blob/master/testplans/spid-cie-oidc/testplan.csv for the Italian national digital identity system). For the remaining columns, the situation is less polarized. For WAS, the CSRF-checker [13] in the General purpose category is limited to detecting one type of attacks with potentially dramatic impact and listed in the CWE Top 25, all the 3 tools for SAML and 9 out of 14 for OAuth/OIDC are not able to identify vulnerabilities commonly found in web applications. All the other tools, except MIG, cannot identify this kind of vulnerability. Given the availability of MIG-L, it is possible to specify tests for a wide range of vulnerabilities and attacks, exceeding the scope of those in the IdM protocols. This is particularly interesting since it is possible to define tests that can convincingly demonstrate the possibility of articulated attacks that exploit multiple vulnerabilities associated with different parts of a web application such as the, previously discussed (see Section TEST SPECIFICATION) recent composite attack made possible by a flaw in the OAuth protocol and the presence of XSS flaws.

TABLE 1: Comparison between MIG-T and the other state-of-the-art tools.

	Auto	WAS	Compl	IdM	Active	Passive	Declarative
General purpose							
AUTHSCAN [10]	✓	✓	-	✓	✓	-	-
BLOCK	✓	✓	-	-	-	✓	-
CSRF-checker [13]	✓	-	-	-	✓	-	-
InteGuard	✓	✓	-	-	-	✓	-
OAuth/OIDC							
FAPI conformance suite	✓	-	✓	-	-	✓	-
Impersonator	✓	-	-	✓	✓	-	-
MoScan	✓	✓	✓	-	✓	-	-
OAuch [11]	✓	-	✓	-	✓	✓	-
OAuth Detector	✓	-	-	✓	-	✓	-
OAuthGuard	✓	-	-	✓	-	✓	-
OAuthShield	✓	✓	✓	✓	✓	-	-
OAuthTester [14]	✓	✓	-	-	-	✓	-
OCSRF	✓	✓	✓	✓	✓	-	-
OpenID attacker	✓	-	-	✓	✓	-	-
OVERSCAN	✓	-	-	✓	-	✓	-
PrOfESSOS	✓	-	-	✓	✓	-	-
S3kVetter	✓	✓	✓	✓	✓	-	-
SSOScan	✓	-	-	✓	✓	✓	-
SAML							
SAMLyze	✓	-	-	✓	✓	-	-
SAML Raider [9]	✓	-	-	✓	✓	-	-
WS-Attacker [8]	✓	-	-	✓	-	-	-
OAuth/OIDC and SAML							
BLAST	✓	✓	-	✓	✓	-	-
BRM Analyzer	-	-	-	-	-	-	-
EsPReSSO	✓	-	-	✓	✓	-	-
WPSE	✓	✓	✓	-	-	✓	-
MIG-T	✓	✓	✓	✓	✓	✓	✓

The column Compl is meaningless for the tools in the General purpose category and is only relevant for the remaining ones: in OAuth/OIDC, half of the tools are not able to check compliance with respect to the standard, none of the tools in SAML can perform compliance checking, and only 1 tool in OAuth/OIDC and SAML can do. Because of the extended threat model adopted in MIG-T (recall Figure 1, it can perform compliance checks with respect to both OIDC and SAML standards since it considers BCPs and assists users to apply them to mitigate potential vulnerabilities that can be mitigated by their adoption when returning reports with actionable information. In other words, this is another distinctive feature of MIG-T. The column IdM is not very relevant to general purpose tools and becomes crucial for all those in the remaining categories. By contrasting columns IdM with column Compl, it becomes clear that only 3 out of 20 tools (i.e. not considering the 5 in General purpose) can verify compliance and checks for IdM vulnerabilities (the OauthTester [14] cannot perform either activities as it is only able to flag protocol executions that deviate from those specified in the OAuth RFC document). This means that most tools are focused on just one activity between vulnerability detection and compliance checking, contrary to MIG-T which naturally supports both again because of its extended threat model.

The columns Active and Passive clearly show that almost all tools support just one type of tests with two notable exceptions, namely the BRM Analyzer [15] (classified as OAuth/OIDC and SAML) that supports none, as it requires manual inspections of the reports produced from SSO protocol traces to detect vulnerabilities, and OAuch [11] that supports both. As discussed above, MIG-T supports both types of tests not only for OAuth/OIDC as OAuch but also for SAML.

CONCLUSIONS

We presented a declarative and automated approach to the security of IdM protocols, which are essential building blocks for securing online services and whose deployments are plagued by a consistent number of vulnerabilities despite various standardization efforts including SAML, OAuth, and OIDC. Experiments on real deployments (including a national digital identity system, a PSD2 payment service, and a number of SSOLinking solutions in web applications) confirm that automation is crucial for scalability and declarativity for considering new vulnerabilities in a rapidly evolving threat landscape.

REFERENCES

1. A. Bisegna, M. Bitussi, R. Carbone, L. Compagna, A. Sudhodanan, and S. Ranise, “CSRF-ing the SSO waves: security testing of SSO-based account linking process", In Proceedings 2024 IEEE European symposium on security and privacy (EuroS&P), 2024.
2. A. Bisegna, R. Carbone, G. Pellizzari, and S. Ranise, “Micro-Id-Gym: A Flexible Tool for Pentesting Identity Management Protocols in the Wild and in the Laboratory", In International Workshop on Emerging Technologies for Authorization and Authentication (pp. 71-89). Cham: Springer International Publishing, 2020.
3. F. Yang, and S. Manoharan, “A security analysis of the OAuth protocol", 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), 2013.
4. National Institute of Standards and Technology (NIST), “Risk Management Guide for Information Technology Systems", 2012.
5. A. Bisegna, R. Carbone, I. Martini, V. Odorizzi, G. Pellizzari, and S. Ranise, “Micro-Id-Gym: Identity Management Workouts with Container-Based Microservices",International Journal of Information Security and Cybercrime 8, pp. 45–50, 2019.
6. A. Carmel, “Traveling with OAuth - Account Takeover on Booking.com", 2023, https://salt.security/blog/traveling-with-oauth-account-takeover-on-booking-com.
7. N. Engelbertz, N. Erinola, D. Herring, J. Somorovsky, V. Mladenov, and J. Schwenk, “Security Analysis of eIDAS–the Cross-Country Authentication Scheme in Europe", 12th USENIX Workshop on Offensive Technologies (WOOT 18), 2018.
8. C. Mainka, J. Somorovsky, and J. Schwenk, “Penetration testing tool for web services security", 2012 IEEE Eighth World Congress on Services, 2012.
9. V. Prasad, and S. Shukla, “SAML Raider: A Burp Suite Extension for SAML Security Testing", 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering, 2018.
10. G. Bai, J. Lei, G. Meng, S. Sathyanarayan Venkatraman, P. Saxena, J. Sun, Y. Liu, and J. Song Dong, “AUTHSCAN: Automatic Extraction of Web Authentication Protocols from Implementations", Proceedings of the 20th Annual Network and Distributed System Security Symposium, 2013.
11. P. Philippaerts, D. Preuveneers, and W. Joosen, “OAuch: Exploring Security Compliance in the OAuth 2.0 Ecosystem", Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses, 2022.
12. A. Bisegna, R. Carbone, and S. Ranise, “Micro-Id-Gym: A Flexible Tool for Pentesting Identity Management Protocols in the Wild and in the Laboratory", In

International Workshop on Emerging Technologies for Authorization and Authentication, 2021.

13. A. Sudhodanan, R. Carbone, L. Compagna, N. Dolgin, A. Armando, and U. Morelli, “Large-scale analysis & detection of authentication cross-site request forgeries", 2017 IEEE European symposium on security and privacy (EuroS&P), 2017.
14. R. Yang, G. Li, W. Cheong Lau, K. Zhang and P. Hu, “Model-based security testing: An empirical study on OAuth 2.0 implementations", Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, 2016.
15. R. Wang, S. Chen and X. Wang, “Signing me onto your accounts through Facebook and Google: A traffic-guided security study of commercially deployed Single-Sign-On web services", 2012 IEEE Symposium on Security and Privacy, 2012.

Andrea Bisegna is a researcher in the Security & Trust Research Unit of the Center for Cybersecurity at Fondazione Bruno Kessler, Italy. His primarily research revolves around digital identity management, as well as the implementation and analysis of security protocols. He received his Ph.D. degree in Secure and Reliable Systems from the University of Genova (Italy). He is the creator and main contributor of Micro-Id-Gym. Contact him at a.bisegna@fbk.eu.

Matteo Bitussi is a MA student in Computer Science at the University of Trento (Italy) and he is currently a research assistant in the Center of Cybersecurity at Fondazione Bruno Kessler (Italy). He designed and developed MIG-T and contributed to the development of MIG-L as part of his Bachelor’s Thesis and ongoing research. Contact him at mbitussi@fbk.eu

Roberto Carbone is the head of the Security & Trust Research Unit of the Center for Cybersecurity at Fondazione Bruno Kessler. He received his Ph.D. in Electronic and Computer Engineering and Telecommunications from the University of Genova (Italy) in 2009. His previous appointments include a period as visiting scholar in the Department of Computer Science at the University of Pittsburgh (Pennsylvania, US). He has been involved in several international and national research projects and industrial collaborations. His research focuses on digital identity management and the (formal) analysis of security protocols and services. Contact him at carbone@fbk.eu.

Silvio Ranise is the director of the Center for Cybersecurity at Fondazione Bruno Kessler and a full professor of Computer Science at the University of Trento Department of Mathematics. He holds a PhD in Computer Engineering from the University of Genoa (Italy) and the Henri Poincaré University (Nancy, France). He has been a researcher at the INRIA-National Institute for Research in Digital Science and Technology, visiting professor at the Computer Science Department of the University of Milan and senior researcher at Fondazione Bruno Kessler. Contact him at ranise@fbk.eu.