# Improving Privacy of OpenID Cloud Identity Management Framework: Formal Analysis, Verification of Protocol

Roshni Bhandari
Department of Computer Engineering
S. S. Agrawal Inst. Of Engg. & Technology, Navsari

Dhiren Patel
Department of Computer Engineering
National Institute of Technology, Surat, India

Brijesh A. Bhandari
Department of Computer Engineering
S. S. Agrawal Inst. Of Engg. & Technology, Navsari

## ABSTRACT

Cloud computing is a new trend of computing paradigm that provides a set of scalable resources on demand. However, it also being a target of cyber attacks and creates risk for data privacy and protection. An Identity Management System (IDM) supports the management of multiple digital identities for authentication and authorization. The various identity management frameworks that help making Cloud environment more secure. OpenID 2.0 is a user-centric Web single sign-on protocol with over one billion OpenID-enabled user accounts, and thousands of supporting websites. The security of the protocol is critical.

In OpenID Identity Management Framework, User Privacy is the issue. In this paper we had introduced the results of a systematic analysis of the OpenID authentication protocol using scyther tool. Our formal analysis reveals that the protocol does not guarantee the authenticity and integrity of the authentication request, and it lacks bindings among the protocol messages and the browser. We provide a simple and scalable defense mechanism for service providers to ensure the authenticity and integrity of the protocol messages.

## General Terms

Identity Management, User Provider, Service Provider, Identity Provider, Cloud Computing

## Keywords

Single Sign-on, OpenID, Authentication, Protocol Analysis

## 1. INTRODUCTION

Cloud Computing is the use of computing resources (Hardware and Software) that are delivered as a service over an Internet. Today's Identity Management infrastructures are adhocracies, built one application or system at a time. The result is a spider web of overlapping repositories, inconsistent policy frameworks and process discontinuities. The resulting systems are error-prone, expensive to manage and riddled with security loopholes. To address this scope and complexity, a simple model is required-a framework that can be used to discuss the major business issues and how to deal with them effectively and efficiently.

A framework will serve as a basis for vital understanding between business management and technical managers on all identity management initiatives. For Cloud Computing, one has to carefully choose appropriate Identify Management framework that can support large number of users running applications in Virtualized environment. In identity management frameworks there is requirement of preserving user privacy and as well as user identity. Thus there is a need for a privacy preserving protocol that doesn't adversely affect the adjoining identity management protocols. OpenID 2.0 is a user-centric Web single sign-on protocol with over one billion OpenID-enabled user accounts, and tens of thousands of supporting websites.

OpenID 2.0 is a decentralized web Single-Sign-On solution. In the OpenID, users are free to choose their own OpenID provider and do not require any pre-registration of service provider to identity provider. While the security of the protocol is critical, so security analysis is required. Based on the analyses, three weaknesses of the OpenID protocol were identified: (i) a lack of authenticity guarantee of the authentication request, (ii) a lack of bindings between the authentication messages and the browser, and (iii) a lack of integrity protection of the authentication request the weakness of the OpenID; authentication and integrity. The solution of the said weaknesses improves the privacy of the users. These backgrounds motivated for the current research work. Some of the commonly agreed upon goals of identity management are: A formal specification and analysis of the OpenID protocol that identifies three weaknesses and correlates possible attack vectors. A protocol is verified using Scyther tool. Proposed mechanism is to prevent attacks that exploit the uncovered weaknesses in the protocol.

## 2. MATERIALS AND METHODS

In this section, a summarization of the related work on OpenID identity management framework. Several possible threats are documented in the OpenID specification itself, including (i) a phishing attack that redirects users to a malicious replica of an IdP website, (ii) the masquerade of an IdP by an Man-In-The-Middle attacker between the SP and IdP to impersonate users on the SP, (iii) a replay attack that exploits the lack of assertion nonce checking by SPs, and (iv) a denial-of-service (DoS) attack that attempts to exhaust the computational resources of SPs and IdPs.

The related work in Tsyrklevich and Tsyrklevich [1] demonstrate a series of possible attacks on the OpenID protocol: (i) a malicious user could trick an SP to perform port scans and exploit non accessible internal hosts; (ii) an MITM attacker between the SP and IdP could perform two distinct DH key exchanges with each party to sign authentication assertions on behalf of the IdP; (iii) an IdP could track all the websites a user has logged into via the return to parameter; (iv) a network attacker could sniff the wire to intercept an authentication response to log into the SP as the victim user; and (v) a Web attacker could insidiously log a user into her SP via a cross-site forged login request.

Barth et al. [2] introduce login CSRF, in which an attacker logs the victim into a site as the attacker by using the victim's browser to issue a forged cross-site login request embedded with the attacker's user name and password. The authors also illustrate how the session swapping attack works in OpenID and in PHP cookie-less authentication. In the case of OpenID session swapping, the attacker first signs into Service Provider using the attacker's identity, intercepts the authentication response, and then embeds the intercepted response in a web page that victims will visit.

Cross-site request forgery that enables an unauthorized and unrelated "third-party" website to retrieve information from or perform actions on the "first-party" website that the user has voluntarily interacted.

Security Analysis of OpenID [3] The Author examined the OpenID extension framework and found that, due to an improper verification of OpenID assertions, the extension parameter values sent within the OpenID protocol could be manipulated if the channel is not SSL-protected.

Attribute exchange security alert [4] The Author found some SP implementations do not check that the information passed through Attribute Exchange extension was signed, which allows an attacker to modify the profile attributes returned from an IdP.

## 2.1 OpenID Authentication Protocol

The main flow of OpenID is shown in Figure 1 and is detailed as follows [5]:

2.1.1 The user requests access to a service or resource at the SP site. At this moment, we assume that the user is not authenticated.

2.1.2 The SP requires the authentication of the user and asks for his OpenID identifier. In order to do so, the SP shows the user a OpenID login page, where he can supply an OpenID identifier.

2.1.3 The user provides an OpenID identifier. He may have several identifiers, and he can choose which one to use. Additionally, OpenID 2.0 allows the user to simply provide the identifier of his identity provider, enhancing this way his privacy by reducing the chances of being traced through his identifier.

2.1.4 The SP performs a discovery process using the supplied identifier to locate the IdP of the user.

2.1.5 The SP and the IdP perform an association process, that is, they generate a shared secret through a Diffie Hellman key exchange. This shared secret will be used to verify subsequent communications.

2.1.6 The SP constructs an authentication request and redirects the user to the IdP site through an HTTP redirection. We will assume that the Attribute Exchange extension is used, so the SP also includes a petition for a set of attributes into the authentication request.

2.1.7 The user gets authenticated by the IdP, for example, by providing his credentials. OpenID does not define a method of authentication, but password-based methods are the most common ones.

2.1.8 The IdP constructs an authentication response, which contains an assertion about the result of the authentication. In case the SP asks for attributes, the IdP also includes their values. Additionally, the IdP signs the request. The user is then redirected back to the SP site in order to continue with the authentication process.

2.1.9 The SP verifies the authentication response and reads the attribute values included within.

2.1.10 The user gets authenticated at the SP site and is able to access to the requested service.

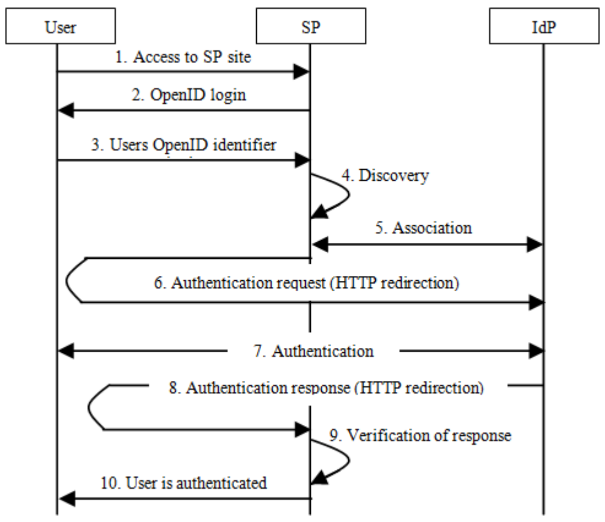

User
SP
IdP
1. Access to SP site
2. OpenID login
3. Users OpenID identifier
4. Discovery
5. Association
6. Authentication request (HTTP redirection)
7. Authentication
8. Authentication response (HTTP redirection)
9. Verification of response
10. User is authenticated

**Fig 1: OpenID Authentication Protocol [5]**

## 2.2 Protocol Formalization

UP -> SP : i , Login Request

User UP selects an IdP, or enters her OpenID identifier i (URL) into an OpenID login form on an SP. The browser B then sends i (URL) to SP "Login Request".

SP -> UP : IdP, i, h, SP, Auth Request

SP sends i (URL), h (optional), and a return URL r to IdP via B to obtain an assertion "Auth Request". The return URL r is where IdP should return the response back to SP (via B).

UP -> IdP : IdP, i, h, SP, E(na, KUI) , UP to IdP authentication

B sends i, r, and h to IdP. The user sends identity providers a nonce na with a shared key k.

IdP -> UP : E(nb , kUI), k1=H(na,nb)

IdP checks i and h against its own local storage. If h is not presented, IdP generates a new session handle h and a session key k. In addition, if a cookie that was previously set after a successful authentication with UP is present in the request, IdP could omit the next steps. IdP presents a login form to authenticate the user. IdP sends UP another nonce nb also encrypted with k. UP computes a new key k1 = H(na.nb) and sends back IdP the value of nb encrypted with k1.

UP -> IdP : E(nb , k1), IdP authenticates UP on nb

UP provides her credentials to authenticate with IdP, and then consents to the release of her profile information. IdP sends UP another nonce nb also encrypted with k.

IdP -> UP : IdP, i, h, SP, n, s , s=HMAC(IdP,i,h,SP,n,KSI)

IdP sends idp, i, h, r, n, and s to the URL specified in r via Bdan "Auth Response".

UP -> SP : IdP, i, h, SP, n, s, Assertion Validation

B redirects the authentication response to SP.SP computes s' = HMAC(idp.i.h.r.n, k) over the received idp, i, h, r, and n, and checks whether s' = s.

We have designed the security goals to analyze the OpenID protocol. These goals will ensure that whether the OpenID protocol meets its specific security requirements or not.

The security goals are specified as follows:

G1: RP authenticates IdP on the value of the signature s = HMAC(IdP.i.h.RP.n, kRI).

When an SP receives an Auth Response, the SP has to check that the Auth Response is generated by the IdP.

G2: RP authenticates IdP on the value of UB.

When an UP receives an Auth Request and Auth Response, the UP has to check that the Auth Request and Auth Response is generated by the IdP.

G3: IdP authenticates UB on the value of nb.

G4: The session key k1 = H(na.nb) should be kept secret between UB and IdP. UP has been authenticated by the IdP.

G5: IdP authenticates SP on the value of the Auth Request (IdP.i.h.SP).

When an IdP receives an Auth Request, the IdP has to make sure that the Auth Request is originated by the SP.

G6: SP authenticates UB on the value of the OpenID identifier i.

The SP needs to ensure the Login Request is initiated by the UB with the user's OpenID identifier.

## 3. THE OPENID VULNERABILITY ASSESSMENT TOOL

Scyther and its GUI were developed by Cas Cremers in 2007 [6]. Scyther is an automatic push-button tool for the verification and falsification of security protocols.

Scyther is freely available for Windows, Linux, and Max OS X platforms. It can be downloaded from [7].

Scyther tool analyzed security protocols in three different ways [6]:

1) Verification of claims: The input language of Scyther allows the specification of security properties in terms of claim events, i.e., in a role specification one can claim that a certain value is confidential (secrecy) or certain properties should hold for the communication partners (authentication). Scyther can be used to verify these properties or falsify them.
2) Automatic claims: If the protocol specification contains no security claims, Scyther can automatically generate claims. At the end of each role, authentication and secrecy claims are added. This augmented protocol description is then analyzed by Scyther as in the previous case.
3) Characterization: For protocol analysis, each protocol role can be "characterized". Scyther analyzes the protocol, and provides a finite representation of all traces that contain an execution of the protocol role. By manually inspecting these patterns, one can quickly gain insight in the potential problems with the protocol and modify it if necessary. For example, the Needham-Schroeder protocol, Scyther determines that there are only two patterns for the responder role: one is the correct behavior of the protocol, and the other is the well-known man-in-the-middle attack. Hence, there are no other possible ways of executing the responder role.

### 3.1 Defense Mechanisms

Hash-based message authentication code (H-MAC)

Data passes between a Sender and a Receiver, sometimes through one or more intermediaries. The data contained in the request message from the Sender influences the Receiver behavior. There is a risk that an attacker could manipulate messages in transit between the Sender and the Receiver to maliciously alter the behavior of the Receiver.

Message manipulation can take the form of data modification within the message, or even substitution of credentials, to change the apparent source of the request message. Working of Hash-based message authentication code involves the following steps: The sender creates a MAC uses a shared secret key and attaches it to the message. The sender sends the message and MAC to the receiver.

The receiver verifies that the MAC that was sent with the message by using the same shared secret key that was used to create the MAC [8].

UP -> SP : i, **t1, t1=HMAC(UP.i.kSP)**, Login Request

SP -> UP : IdP, i , h , SP, **t2, t2=HMAC(UP,IdP,i,h,SP, kSI)** Auth Request

UP -> IdP : IdP, i, h, SP, **t2**, E(na, kUI), UP-to-IdP authentication

IdP -> UP : E(nb , kUI), k1=H(na,nb)

UP -> IdP : E(nb , k1), IdP authenticates UP on nb

IdP -> UP : IdP, i, h, SP, **t2**, n, s , s=HMAC(IdP,i,h,SP,**t2**,n,KSI) Auth Response

UP -> SP : IdP, i, h, SP, **t2**, n, s.
When delivering a login form, SP generates token t1 and add it to the login form as a hidden form field. kSP is a session level secret key generated by SP. Token t1 is used to ensure the login request is originated from the SP itself.

On receiving a login request, SP computes t1' and checks whether t1'= t1 from the request. If it is then SP initiates an Auth Request with parameter t2 delivered to the return_to URL of the Auth Request.

On receiving an Auth Response, SP extracts t2 from the return_to URL, computes t2' and check whether t2'=t2 to the Auth response signature validation.

The lack of security guarantee in the OpenID protocol .We aimed to satisfy the following properties when designing our defense mechanisms:

Completeness: The defense mechanism must address all weaknesses uncovered from our formal model.

Compatibility: The protection mechanism must be compatible with the existing OpenID protocol and must not require modifications to IdPs and the browsers.

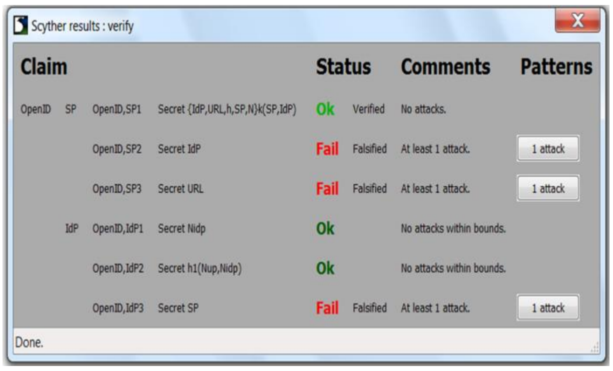
Scalability: Statelessness is a desirable property of the defense mechanism.

Simplicity: it should only use cryptographic functions (i.e., HMAC) and data that are readily accessible to SPs.

## 4. EXPERIMENTAL RESULTS

We formally model the OpenID authentication protocol in Scyther tool.

The main goal of the OpenID authentication protocol is to check to a service provider that the user owns a specific OpenID URL controlled by the identity provider.



Scyther results : verify

| Claim | | | | Status | | Comments | Patterns |
|---|---|---|---|---|---|---|---|
| OpenID | SP | OpenID,SP1 | Secret {IdP,URL,h,SP,N}k(SP,IdP) | Ok | Verified | No attacks. | |
| | | OpenID,SP2 | Secret IdP | Fail | Falsified | At least 1 attack. | 1 attack |
| | | OpenID,SP3 | Secret URL | Fail | Falsified | At least 1 attack. | 1 attack |
| | IdP | OpenID,IdP1 | Secret Nidp | Ok | | No attacks within bounds. | |
| | | OpenID,IdP2 | Secret h1(Nup,Nidp) | Ok | | No attacks within bounds. | |
| | | OpenID,IdP3 | Secret SP | Fail | Falsified | At least 1 attack. | 1 attack |

Done.

**Fig 2: Scyther output of OpenID Authentication Protocol**

Compromising G2

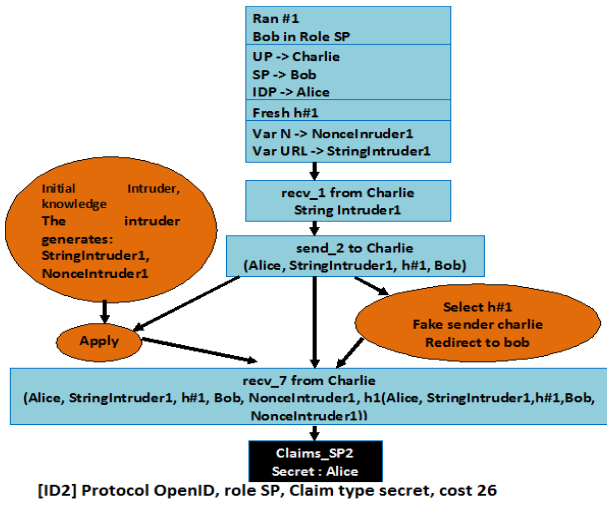

Ran #1
Bob in Role SP
UP -> Charlie
SP -> Bob
IDP -> Alice
Fresh h#1
Var N -> NonceInruder1
Var URL -> StringIntruder1
Initial Intruder, knowledge The intruder generates: StringIntruder1, NonceIntruder1
recv_1 from Charlie String Intruder1
send_2 to Charlie (Alice, StringIntruder1, h#1, Bob)
Select h#1 Fake sender charlie Redirect to bob
Apply
recv_7 from Charlie (Alice, StringIntruder1, h#1, Bob, NonceIntruder1, h1(Alice, StringIntruder1,h#1,Bob, NonceIntruder1))
Claims_SP2 Secret : Alice
[ID2] Protocol OpenID, role SP, Claim type secret, cost 26

**Fig 3: Session swapping and impersonation attack on OpenID authentication protocol**

The violation of the G2 goal reveals that the OpenID protocol lacks bindings between the Auth Request, Auth Response, and the browser. This means that when an SP receives an Auth Response, the SP cannot assert that the Auth Response is sent from the same browser through which the authentication request was issued.

The lack of contextual binding in the protocol enables many possible attacks when an Auth Response is intercepted by an intruder, such as (1) a session swapping attack that forces the user's browser to initialize a session authenticated as the attacker, (2) an impersonation attack that impersonate the user by sending the intercepted Auth Response via a browser agent controlled by the attacker.

Compromising G5

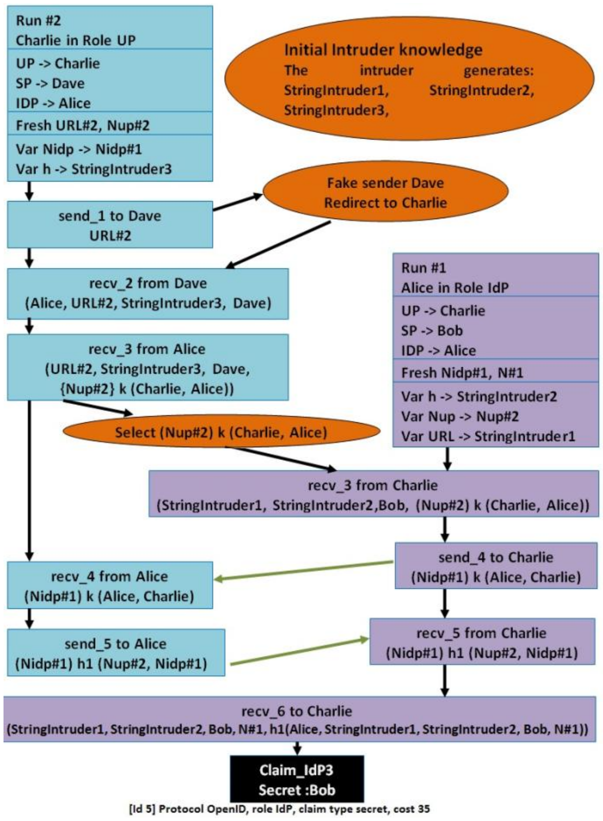

Run #2
Charlie in Role UP
UP -> Charlie
SP -> Dave
IDP -> Alice
Fresh URL#2, Nup#2
Var Nidp -> Nidp#1
Var h -> StringIntruder3
Initial Intruder knowledge The intruder generates: StringIntruder1, StringIntruder2, StringIntruder3,
Fake sender Dave Redirect to Charlie
send_1 to Dave URL#2
recv_2 from Dave (Alice, URL#2, StringIntruder3, Dave)
Run #1
Alice in Role IdP
UP -> Charlie
SP -> Bob
IDP -> Alice
Fresh Nidp#1, N#1
Var h -> StringIntruder2
Var Nup -> Nup#2
Var URL -> StringIntruder1
recv_3 from Alice (URL#2, StringIntruder3, Dave, {Nup#2} k (Charlie, Alice))
Select (Nup#2) k (Charlie, Alice)
recv_3 from Charlie (StringIntruder1, StringIntruder2,Bob, (Nup#2) k (Charlie, Alice))
send_4 to Charlie (Nidp#1) k (Alice, Charlie)
recv_4 from Alice (Nidp#1) k (Alice, Charlie)
send_5 to Alice (Nidp#1) h1 (Nup#2, Nidp#1)
recv_5 from Charlie (Nidp#1) h1 (Nup#2, Nidp#1)
recv_6 to Charlie (StringIntruder1, StringIntruder2, Bob, N#1, h1(Alice, StringIntruder1, StringIntruder2, Bob, N#1))
Claim_IdP3 Secret :Bob
[Id 5] Protocol OpenID, role IdP, claim type secret, cost 35

**Fig 4: A SSO CSRF and parameter forgery attack on OpenID authentication protocol**

The violation of the G5 goal indicates that the authenticity and integrity of the Auth Requests are not protected by the OpenID protocol. That is, an IdP might accept an Auth Request sent from the intruder or the Auth Request might be altered during the transmission. This weakness could be exploited in many ways, such as (1) a SSO CSRF attack that forces the victim to log into her RP website by sending a forged Auth Request via the victim's browser, (2) a parameter forgery attack that manipulates the victims profile attributes requested by the RP websites through a modification of the Auth Request within the protocol. Compromising G6

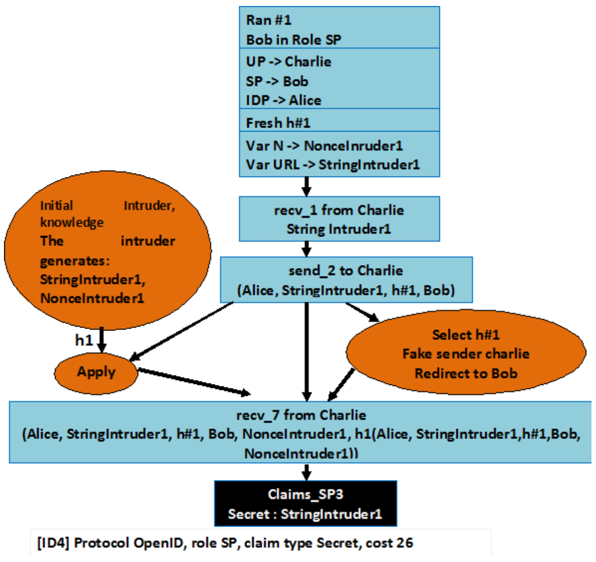
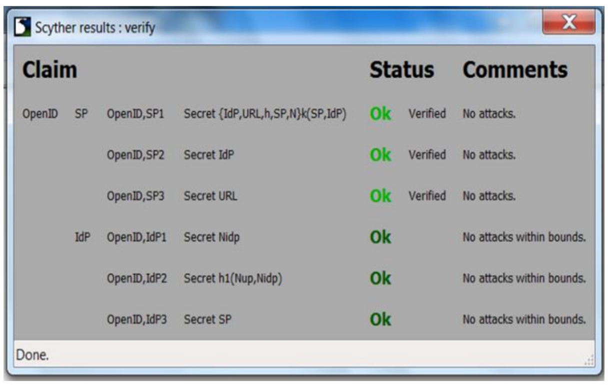


**Fig 5: A SSO CSRF Attack via Login Request on OpenID Authentication Protocol**

## 5. VERIFICATION OF MODIFIED OPENID AUTHENTICATION PROTOCOL

The defense mechanism of the OpenID Authentication Protocol is stateless, and designed to be implemented completely on the SP server-side. All required cryptographic functions (i.e., HMAC) and data (i.e., Auth Request) are readily accessible to the SP. The mitigation approach uses an HMAC function to bind the session identifier to the protocol messages in order to provide binding and ensure the integrity and authenticity of the authentication request. Using an HMAC code as a validation token avoids the exposure of the session identifier, and prevents an attacker who learned the token from inferring with the user's session identifier. For SPs that support an OpenID extension, the extension request parameters can be included in the return_to URL to be protected by the defense mechanism.

Our defense mechanism prevents SSO CSRF via Login Request attacks as an attacker is not able to compute the validation token t1 without knowing the session identifier and the SP's secret key. SSO CSRF via Auth Request and session swapping attacks are mitigated as well, because the session identifier in the attacker's browser session is different from the one in the victim's browser. The integrity of Auth Request is guaranteed as the Auth Request is accompanied by an HMAC.


Scyther results : verify

| Claim | | | | Status | | Comments |
|---|---|---|---|---|---|---|
| OpenID | SP | OpenID,SP1 | Secret {IdP,URL,h,SP,N}k(SP,IdP) | Ok | Verified | No attacks. |
| | | OpenID,SP2 | Secret IdP | Ok | Verified | No attacks. |
| | | OpenID,SP3 | Secret URL | Ok | Verified | No attacks. |
| | IdP | OpenID,IdP1 | Secret Nidp | Ok | | No attacks within bounds. |
| | | OpenID,IdP2 | Secret h1(Nup,Nidp) | Ok | | No attacks within bounds. |
| | | OpenID,IdP3 | Secret SP | Ok | | No attacks within bounds. |

Done.

**Fig 6: Scyther output of modified OpenID authentication protocol**

## 6. CONCLUSION

We conducted a formal model checking analysis of the OpenID 2.0 protocol.

From the present study following conclusions are drawn:

Our model checking analysis revealed that the OpenID protocol does not provide an authenticity or integrity guarantee for the authentication requests. We have improved it to provide a simple and scalable defense mechanism for service providers to ensure the authenticity and integrity of the protocol messages.

In future work, we plan to do analysis of other identity management frameworks.

## 7. REFERENCES

[1] Barth A, Jackson C, and Mitchell JC, Robust defenses for cross-site request forgery, In Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08), New York, USA, 2008.

[2] Sovis P, Kohlar F, and Schwenk J, Security analysis of OpenID, In Proceedings of the Securing Electronic Business Processese Highlights of the Information Security Solutions Europe 2010 Conference; October 2010.

[3] Lindholm A. Security evaluation of the OpenID protocol, Master of Science Thesis, Royal Institute of Technology, 2009.

[4] Wang R, Chen S, and Wang X, Attribute exchange security alert, http://openid.net/2011/05/05/attribute-exchange-securityalert

[5] Nunez D, Agudo I, and Lopez J, Integrating openid with proxy re-encryption to enhance privacy in cloud-based identity services, in Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on, 2012.

[6] Cremers C, Scyther tool, http://people.inf.ethz.ch/cremersc/scyther

[7] Cremers, C. Scyther - semantics and verification of security protocols, Ph.D. dissertation, Eindhoven University of Technology, 2006.