Formal Analysis of BrowserID/Mozilla Persona

CS259 Project Report

Connor Gilbert
Stanford University
Dept. of Computer Science
connorg@cs.stanford.edu

Laza Upatising
Stanford University
Dept. of Computer Science
lazau@cs.stanford.edu

ABSTRACT

Federated identity protocols are growing in popularity as the Internet community attempts to reduce the number of passwords users must remember. Such protocols allow users to maintain a shared secret with a single provider who certifies the user's identity to other services. This paper presents the first known formal analysis of one of the newest federated identity protocols, *BrowserID*, which is more commonly known by the name of its flagship implementation, *Mozilla Persona*. We model the protocol in the formal verification tool Murϕ and report the results of this formal verification under various adversarial capabilities and security requirements. Our analysis shows that, while BrowserID is secure under most conditions, adversaries with the ability to block connections to SSL endpoints or change the contents of a special file on an HTTPS web server can break the protocol's security guarantees. We demonstrate one of these attacks in a laboratory environment.

1. INTRODUCTION

Federated identity protocols present a compelling alternative to the standard model of one account per user per site, which is vulnerable to password reuse, compromise at any one of many sites holding a user's secret, and the simple inability of the user to remember strong, unrelated passwords for every website he or she uses. But, a website relying on another party to authenticate users must be able to trust the mechanism by which such authentication is completed; bugs in the design or implementation of a federated protocol have the potential to cause problems for many otherwise independent websites.

The BrowserID protocol was developed by the Mozilla Identity Team and first released in July 2011 [12]. The full protocol standard is available online but it is not the product of a standards body [10]. An early discussion of the design principles and goals of BrowserID (then called the Verified Email Protocol) was presented at the 2011 W3C Workshop on Identity in the Browser before BrowserID was formally announced [6]. The protocol is a compelling case for study because it is still relatively new, and because its attractive design and Mozilla's support make it a promising candidate for widespread use.

We first discuss related work in Section 2. Then, we describe the BrowserID system, identify our adversarial models, and define desired security properties in Section 3. Next, we describe our formal model's implementation in Section 4 before presenting the results of our verification in Section 5.

2. RELATED WORK

BrowserID has been the subject of some study in the literature. Hackett and Hawkey present an analysis of security concerns with the design of BrowserID, but do not assess direct vulnerabilities to the authentication guarantees in the message protocol itself [5]. Rather, their work concentrates on the interaction of BrowserID with the email system, the potential for key loss or phishing, privacy breaches, and other concerns. Bai et al. employ an automated process to discover flaws in the implementation of BrowserID on popular websites, but not in the design of the protocol [2].

BrowserID is one of a number of federated identity protocols in use on the web. It is similar in design to OpenID, which has been heavily analyzed, but relies on email addresses rather than URIs to identify users; in addition, BrowserID gives important cryptographic responsibilities to the browser. Sovis, Kohlar, and Schwenk present a number of attacks on the design and implementation of OpenID assertions and authentication messages [15]. Van Delft and Oostdijk present a comprehensive summary of potential and observed security issues in OpenID, though most do not address the authentication guarantee offered by the protocol and focus on denial of service and other vulnerabilities [17]. Sun, Hawkey, and Beznosov present an analysis of OpenID which employs formal model checking and empirical evaluation, discovering and developing a mitigation for a common OpenID vulnerability [16]. Stanford students have formally verified – and broken – the security assumptions for various parts of OpenID [9, 14].

The Information Card Profile specification, like BrowserID, allows Relying Parties to verify cryptographic assertions from trusted Identity Providers; it is formally verified against web adversaries by Bhargavan, Fournet, Gordon, and Swamy [3]. Similarly, Facebook Connect federated authentication is model-checked by Miculan and Urban, who discover at least one unknown vulnerability [8]. Alloy models were employed by Akhawe, Barth, Lam, Mitchell, and Song to define a foundational web security model and discover known and novel attacks on protocols including WebAuth, which is similar in spirit to BrowserID, though more commonly used with a single identity provider, like a university [1].

3. SYSTEM, ADVERSARIES, AND SECURITY PROPERTIES

3.1 System Description

BrowserID allows users to prove their identity to remote services using public-key cryptographic assertions in a chain of trust originating at their domain. An "Active Identity" – the basic identifier for a user – is of the familiar form *user@domain*; it commonly indicates an email address but does not need to be SMTP-routable.

The browser is in many ways a critical root of trust for the system – it safeguards the user's certificate obtained from the IdP, generates keypairs, signs Identity Assertions (see below), performs other cryptographic operations, and handles the user's identity and password.

The main parties in a BrowserID transaction are:

- **User** – The person proving their identity. (Actions taken by the browser are at the request of the user, so the browser and user can be combined conceptually.)
- **Identity Provider (IdP)** – The service at *domain* verifying *user*'s control over the Active Identity *user@domain*.
- **Relying Party (RP)** – A website or other service which is relying on BrowserID to establish the identity of its users.

The process by which a user's identity is certified to a Relying Party consists of three main steps:

1. **User Certificate Provisioning** – The browser obtains a signed *user certificate* from an IdP. (If the browser still has a valid certificate from a previous protocol run, it can reuse that certificate.)
   (a) *Finding an IdP* – Given a user@domain Active Identity specified by the user, the browser fetches a well-known file over SSL from the domain. The domain can respond by (1) providing a public key and relative URLs at which to complete the authentication and provisioning processes; (2) delegating authority to another domain, in which case the browser repeats the process with that domain; or (3) responding with 404 or an equivalent error, or an explicit `disabled` message[1], in which case the browser will "fall back" to a Mozilla-operated IdP at `login.persona.org`. This fallback IdP sends an email to the user's address with a verification token, allowing the user to prove control of the address and establish a password; thereafter, the fallback IdP can issue certificates for the user.
   (b) *Obtaining a certificate* – The user inputs their Active Identity and authenticates with their IdP over SSL. The browser generates a keypair and sends a public key to the IdP, receiving a certificate in return. The certificate includes the Active Identity, generated public key, IdP name, and issue and expiration times.
2. **Assertion Generation** – The browser generates an *identity assertion* in response to a specific login request, simply signing a message that includes the domain of the RP and an expiration time (recommended to be about 5 minutes in the future).
3. **Assertion Verification** – The user certificate and identity assertion are sent together to the RP. The RP verifies that the assertion is directed to it, that neither item is expired, and that the messages are both signed by valid certificates for the correct entities.

[1]This is not documented in the protocol but is supported by Persona.

3.2 Threat Models/Adversaries

We model two main types of adversaries: network adversaries and server adversaries.

We model the **network adversary** after the Dolev-Yao model [4]. The adversary has control of the network and can intercept, modify, and duplicate all messages in the network, including all communication between a user and an honest IdP/RP. The adversary can also sign messages with his own keypair and any keypair he compromises. A network adversary can, in addition, block connections to arbitrary servers (for instance, by sitting on a user's local network or by executing a DoS attack). Given that control of email is critical to the security of the fallback IdP, which validates control of email addresses for domains without their own IdPs, we allow the network adversary to intercept email. This is not a trivial capability since much of today's client email access traffic is encrypted, but most emails are still passed unencrypted between mail servers, so a well-placed adversary could intercept messages. This capability also will demonstrate the places where email control is critical to the protocol's security.

A **server adversary** has control of a domain's `/.well-known/browserid` file (hereafter "the BrowserID file") and may also control a malicious IdP server. We believe this adversary capability is reasonable: well-known sites have had content injected as a result of web software vulnerabilities, suggesting that giving a critical authentication role to the web server is risky (e.g. `DHS.gov` served malware in 2010 [13] – a subtle change to a BrowserID file might be harder to notice).

We specifically **leave out browser-based adversaries** since the browser is a critical root of trust for the protocol. A compromised browser could easily obtain the {user@domain, BrowserID password} pair and is in control of critical cryptographic operations. An adversary in the browser or the host operating system can therefore trivially undermine the security of the entire protocol.

3.3 System properties

3.3.1 Authentication of parties

When the protocol completes:

- A user is authenticated to the RP he wishes to log in to.
- The RP authenticates the user intending to log in.
- A {User Certificate, Identity Assertion} pair cannot be reused to sign on to a different RP[2].

3.3.2 Secrecy

We hold that the following protocol secrets are never compromised:

- User's credentials (user@domain and password). This includes the property that a malicious IdP cannot obtain any {user@domain, password} pair that does not belong to its domain.
- User and IdP secret keys.

[2]We limit this assurance to RPs that follow the protocol – non-compliant RPs should get no assurance of user authentication!

3.3.3 *Fallback*

Mozilla also supports a fallback IdP as a solution to the scaling issues involved with deploying Persona as a widely accepted standard. Trust for the fallback IdP, `login.persona.org`, is built into the BrowserID protocol.

One specific property that must hold for proper security is thus:

- Mozilla's fallback IdP should never certify a user for an IdP that is currently active. For example, if `stanford.edu` operates its own IdP, the fallback IdP should not issue certificates for any user whose domain is stanford.edu[3].

4. IMPLEMENTATION

We use Murϕ [7] to implement a model of the BrowserID protocol, adversaries, and security properties. Our analysis does not hinge on the specifics of cryptography – we assume, with the protocol authors, that RSA, DSA, SSL, and other supported cryptographic schemes are secure – so we can accept the black-box cryptography model allowed by Murϕ analysis. We do not attempt to model the entire universe of threats against BrowserID – including those from a more standard web attacker – and instead focus on the logical parties involved and the messages transmitted between them.

4.1 Actors

The main logical actors in the protocol are Users, Identity Providers (IdPs), and Relying Parties (RPs). We add a Domain type to model the way the protocol consults a domain over SSL to determine its support for BrowserID. Additionally, the Fallback IdP keeps track of users that it has certified so that security properties can be verified. We also add adversaries, as described in Section 4.4.

The agents maintain the following state:

```
User : record
  state:   UserStates;
  rp:      RPId;
end;

RP : record
  state:   RPStates;
  user:    UserId;
  userCertificateSigner: IdPId;
end;

Domain : record
  isIdP: boolean;  -- domain has an IdP
  IdP:   IdPId;    -- that IdP
  isDel: boolean;  -- domain delegates
  Del:   DomainId; -- delegate domain

-- Users the Fallback IdP has verified
Fallback : record
  pairs: multiset[MaxFBCerts]
              of FallbackPair;
end;
```

A FallbackPair is defined as follows, with the `certifiedFor` member indicating which agent responded with the verification token and `user` indicating the user who was certified:

[3]This is allowed by the validation standards in the protocol. However, in the open Issue filed against the BrowserID project at `https://github.com/mozilla/browserid/issues/1501`, the protocol designers recognize this is a problem for IdPs concerned with more secure authentication.

```
-- Pairs of AgentId, AgentId that
 -- represent the Fallback's verified agents
FallbackPair : record
  certifiedFor:  AgentId;
  user:          AgentId;
end;
```

This information, not specifically tracked in the actually protocol, allows us to detect when an adversary has replied to a token.

IdPs do not require state, since we don't model attempted authentication using incorrect credentials – IdPs just respond to authentication requests by automatically generating any requested certificate. This is certainly not a realistic real-world condition, but allows us to concentrate on runs of the protocol which will not abort due to incorrect credentials.

We maintain the following variables, keeping track of all agents and a limited-size network. Note that although fallback IdPs are tracked as an array, we only test with one fallback IdP in the model:

```
var
  net:   multiset[NetworkSize] of Message;
  user:  array[UserId]   of User;
  idp:   array[IdPId]    of IdP;
  fbIdP: array[FBIdPId]  of Fallback;
  rp:    array[RPId]     of RP;
  dom:   array[DomainId] of Domain;
  nAdv:  array[NetAdvId] of NetAdv;
  wAdv:  array[WebAdvId] of WebAdv;
  mDom:  array[DomainId] of Domain; --adv.
  mIdP:  array[IdPId]    of IdP;    --adv.
```

4.2 Messages

For simplicity in the implementation of message sending and adversary behaviors, we modeled messages as a single record type encompassing all possible messages in the protocol. Only the relevant fields are defined for each message.

Some messages in the protocol are sent over SSL. The protocol requires authentication and provisioning messages to the IdP to be sent over SSL. The transport security for other communications is not specified. In cases where SSL is required, we set a message variable `ssl` and prevent any agent other than the specified destination from reading those messages. In addition, we prevent replay of such messages by the network adversary because SSL is secure against replay attacks, so replaying them will not have any effect.

In order to limit the number of active states in our model, each protocol step is fired only when the correct type of message is in the network. Therefore each protocol step can be mapped to an incoming message type and an outgoing message type.

```
MessageType : enum {
  M_AskUserToAuth,
  -- RP asks user to authenticate
  M_GetBrowserIdFile,
  -- User gets /.well-know/browserid
  M_BrowserIdFile,
  -- Domain's response
  M_UserAuth,
  -- User asks IdP for User Cert
  M_UserAuthFallback,
  -- User must authenticate with
    -- fallback IdP
  M_FallbackRequestVerify,
  -- Fallback IdP requests user to
```

```
    -- verify that it owns user@domain
  M_FallbackReplyVerify,
  -- User certifies it owns user@domain
  M_UserCert,
  -- IdP'sResponse with signed cert
  M_UserCertAssertion,
  -- Message to RP with signed
    -- user cert and identity assertion
};
```

The message format is:

```
Message : record
  mType:    MessageType;
  source:   AgentId;
  dest:     AgentId;
  ssl:      boolean; -- SSL to dest

  -- BrowserId file Response Fields
  auth: AgentId; -- delegate to domain

  -- User Certificate Req/Resp Field
  credentials: UserId;
  genKey:   UserId; -- User-gen. key

  -- User Certificate Fields
  user:        UserId;
  domain:      AgentId;
  idpName:     IdPId; -- issuing IdP
  sigUserCert: IdPId; -- signing IdP

  -- Identity Assertion Fields
  rp: RPId; -- object of assertion
  sigIdAssert: UserId; -- user sig.
end;
```

4.3 Protocol Steps

We model each step in the protocol as a separate rule. Each rule either initiates the protocol (the RP asking the user to authenticate) or responds to a message (each other rule).

"Epsilon" Rules: We implement "epsilon rules" that nondeterministically assign various combinations of parameters, including domain delegation and IdP selection and lack of BrowserID support, before they are used in the protocol. For example, we define these rules to initialize the delegation state of the domains.

```
ruleset d: DomainId do
  ruleset i: IdPId do
    rule 20 "Assign an IdP to a Domain"
      dom[d].init = False
    ==>
    begin
      dom[d].isIdP := True;
      dom[d].IdP   := i;
      dom[d].init  := True;
    end;
  end;
end;

ruleset d: DomainId do
  rule 20 "Domain does not support BrowserID"
    dom[d].init = False
  ==>
  begin
    dom[d].init := True;
  end;
end;

ruleset d1: DomainId do
  ruleset d2: DomainId do
    rule 20 "Delegate domain d1 to d2"
      dom[d1].init   = False &
      dom[d2].isIdP = True
    ==>
    begin
      dom[d1].isDel := True;
      dom[d1].Del   := d2;
      dom[d1].init  := True;
    end;
  end;
end;
```

States: We define the following states that map with the action the user or RP is carrying out or waiting for:

```
UserStates : enum {
  U_SLEEP,   -- nothing's happening
  U_FINDIDP, -- finding IdP
  U_FINDIDPFOLLOWDEL,
   -- finding IdP, following delegation
  U_VERIFY, -- wait for fallback email
  U_AUTH,   -- authed to IdP/has cert
  U_ASSERT  -- cert+assertion sent; done.
};

RPStates : enum {
  R_SLEEP,  -- nothing's happening
  R_WAIT,   -- requested user to auth
  R_CHECK,  -- received user cert+assert
  R_DONE    -- all done
};
```

4.4 Adversaries

Network Adversary: We implemented a standard Dolev-Yao network adversary with the ability to intercept messages (either removing them or injecting them later) and generate new messages with information it has captured [4].

The network adversary maintains knowledge of messages and credentials it has compromised.

```
NetAdv : record
  msgs: multiset[MaxKnowledge] of Message;
  creds: multiset[MaxKnowledge] of UserId;
end;
```

The network adversary only intercepts messages that are unencrypted or sent encrypted to it:

```
if (isundefined(net[m].ssl) |
  !net[m].ssl | net[m].dest = n) then
  -- [Continue with capture]
end;
```

And, the network adversary pays particular attention to credentials in intercepted messages:

```
if !isundefined(msg.credentials) then
  -- learn credentials
  multisetadd (msg.credentials,
               nAdv[n].creds)
end;
```

In addition, if a message is not in the adversary's current set of known messages, the adversary will record that message. Code is omitted for brevity.

In accordance with the Dolev-Yao model, the adversary can tamper with the user certificate in messages from a user to a relying party, but cannot forge signatures.

Server Adversary: We model the server adversary as a domain with a compromised BrowserID file. The file is responsible for key distribution, authentication and provision flow; delegation of authority; and indication of no support for BrowserID, as described in Section 3.1.

Since it controls the contents of the BrowserID file, the server adversary can therefore generate a response to the user's and RP's request for the IdP's BrowserID file using whatever parameters it chooses. We present one variant, in which neither the `idpName` nor the `auth` fields are defined, indicating a response analogous to a 404 error or the explicit "disabled" message:

```
-- Respond BrowserId not supported
-- (Will cause fallback to Mozilla)
ruleset w: WebAdvId do
  choose m: net do
    rule 20 "Web adv. forces fallback"
      net[m].mType = M_GetBrowserIdFile
    ==>
    var
      outM: Message;
      inM:  Message;
    begin
      inM := net[m];
      multisetremove (m,net);
      undefine outM;
      outM.mType   := M_BrowserIdFile;
      outM.source  := inM.dest;
      outM.dest    := inM.source;
      multisetadd (outM,net);
    end;
  end;
end;
```

This adversary can also send a message that indicates another domain will handle authentication for this domain; we direct the requestor to a malicious domain which can then steal credentials or carry out other undesirable behavior. Malicious domains and IdPs maintain the same state as normal domains and IdPs but must never be contacted in a secure protocol run.

4.5 Security Properties

Authentication: We define standard authentication properties: the RP and the user must mutually believe they are talking with each other. This follows the standard form:

```
invariant "User correctly authenticated"
  forall r: RPId do
    rp[r].state = R_DONE
    ->
    user[rp[r].user].rp = r
  end;

invariant "RP correctly authenticated"
  forall u: UserId do
    user[u].state = U_ASSERT
    ->
    rp[user[u].rp].user = u
  end;
```

Secrecy: We hold that the user's IdP credentials must not be known to any adversary at any time.

```
invariant "Network adversary does not have
           user's credentials"
 forall n: NetAdvId do
   true
   ->
   multisetcount(m: nAdv[n].messages,
    (isundefined(nAdv[n].messages[m].ssl) |
     nAdv[n].messages[m].ssl = false) &
    (!isundefined(
       nAdv[n].messages[m].credentials))) = 0
 end;
```

As discussed in the server adversary section, a user should never be forced or confused into contacting a malicious IdP, who might steal their legitimate credentials or commit other undesired actions.

```
invariant "Never contact malicious IdP"
  true
  ->
  multisetcount(m:net,
    (ismember(net[m].dest,mIdP) &
     net[m].mType = M_UserAuth) = 0
```

By establishing a shared secret with the FallbackIdP based on a user's Active Identity *user@domain*, an adversary could obtain a user certificate even though it never learns the secret *user* shares with the *domain* IdP. Thus we modeled an invariant that ensures that a FallbackIdP never establishes a shared secret with an adversary to certify a valid user's Active Identity.

```
invariant "Adversary never establishes
        shared secret with Fallback IdP"
  forall f: FBIdPId do
    true
    ->
    multisetcount (c:fbIdP[f].pairs,
      fbIdP[f].pairs[c].certifiedFor !=
        fbIdP[f].pairs[c].user) = 0
  end;
```

Temporal Validity: Due to the difficulty of expressing these concepts in Murϕ, we do not model the concept of time or the temporal validity of certificates and assertions. This is a reasonable concession – the protocol clearly requires that RPs verify the freshness of user certificates and identity assertions, so the expiration of either will prevent the protocol from successfully completing. However, we explore a potential vulnerability based on this security property in Section 5.2.

5. EVALUATION

5.1 Security Assurances

The BrowserID protocol authors identify the protocol's fundamental security assumptions: "the user's private key remaining secret, the domain's private key remaining secret, and the SSL connection to the IdP being secure" [11]. We determine that, when all of these assumptions hold **and** SSL connections are guaranteed to complete **and** the HTTPS web server is uncompromised, the protocol is secure against our network and server adversaries. The security assurances provided under these conditions are:

- The user is properly authenticated to the Relying Party (RP).
- The user is only logged in to RPs with which it is interacting. (A network message cannot be forged which causes the user to be logged into an unrelated RP.)

- The credentials a user uses to log in to an IdP are not compromised by a network adversary.
- Secret keys are not revealed[4].
- The fallback IdP cannot be induced to issue certificates for a user whose domain has an IdP.

(This list encompasses all of the properties described in Section 3.3.)

5.2 Security Flaws

Network Adversary:
When we allow a network adversary to block connections to an HTTPS web server and read the contents of unencrypted email, we have the following security flaw:

- If the network adversary blocks the HTTPS connection from the browser to the domain's SSL web server, for instance through local-network-level blocking or through a Denial of Service attack. The browser then "falls back" to the `login.persona.org` fallback IdP. If the user does not yet have a shared secret with the fallback IdP, an email is sent to the user to prove that the user controls that account. The network adversary can intercept this message and capture its secret token or link, "proving" he is the user and establishing a shared secret with the fallback IdP[5]. According to the protocol, RPs "SHOULD" accept assertions from the fallback IdP even if an IdP is running on a domain, so the adversary can log in under the user's identity.

  A countermeasure – which appears to be implemented in the Mozilla fallback IdP – is for the fallback IdP to refuse to certify for a user whose domain has operated an IdP recently. Then, the adversary would be forced to keep the IdP down for a longer period in order to compromise credentials.

Server Adversary:
Next, we allow a server adversary to modify the contents of the `/.well-known/browserid` file. This is a reasonable adversary capability to consider because the BrowserID protocol places significant trust in the HTTPS web server, giving it responsibility for IdP key distribution and potential delegation of authority to other domains. Less secure domains may be compromised through file upload bugs in Content Management Systems or other software and thus experience these potential security flaws. This leaves any user authenticating using an Active Identity under a compromised domain vulnerable to the following attacks:

- An adversary who can compromise the BrowserID file can delegate authority and redirect the user to a malicious domain and malicious IdP. This malicious IdP can capture user credentials and issue signed user certificates on behalf of the domain. Any RP verifying a user certificate for this domain will also experience this delegation when retrieving the domain's public key and therefore will consider such certificates valid.

[4]This is not verified explicitly by our model, which has no concept of private keys because they are never sent by any party.

[5]An especially clever adversary might even send the password he has established to the user, trying to convince them that Persona has helpfully generated a password for them.

- A malicious IdP can act as a silent Man-in-the-Middle (MitM) during the authentication; the IdP can record the user's credentials, then use those credentials to obtain a certificate for the user from the true IdP provisioning system, and finally return that truly valid certificate to the user unmodified.

  This requires the adversary to compromise the SSL web server and place a new BrowserID file that delegates to a malicious IdP. Then, this malicious IdP must return the compromised domain's correct BrowserID public key still in use at the real IdP. RPs, when verifying the user certificate, should accept it because the Issuer Name matches the email domain and because the public key the RP retrieves for that domain will verify the certificate's signature.[6]

  Presumably, a domain that supports BrowserID with its own IdP will be more vigilant of the contents of the BrowserID file, so this attack may not be wholly realistic. Nevertheless, the potential impact is substantial – a file-upload vulnerability on a domain's web server can lead to the compromise of the plaintext credentials for all BrowserID users on that domain.

  (This vulnerability is not explicitly found by our Murϕ model due to complexity; rather, it is a combination of the above vulnerability, found by Murϕ, and a critical reading of the protocol specification.)

  We have demonstrated this vulnerability on the most recent version of the Persona verifier, the flagship implementation of BrowserID. For details, see Appendix A.

Temporal Validity:
We note, despite not modeling this possibility in Murϕ for concerns over simplicity and correctness, that sessions can be extended past the expiration of the assertions backing them. The user certificate from an IdP is valid for a maximum of 24 hours, and browsers may reuse the same certificate until it is no longer valid. The RP, however, is not required to end the user's session when the user certificate expires. For long-term identities like persistent email addresses, this is probably acceptable; however, if an IdP wishes to implement other authentication guarantees – for instance, a limited-time assertion – or to implement a security policy requiring their users to reauthenticate more often than the longest RP's session length, they will not be able to do so.

5.3 Runtimes, States, and Rules

The protocol is relatively complex: it requires a large number of messages to be exchanged between a number of parties – Users, IdPs, Domains, and RPs, plus malicious variants of these actors. We only model a one each of the network and server adversaries; since adversaries have no means to collude, increasing their number would provide no benefit. We find that it is difficult for Murϕ to complete its analysis on parameter values larger than 1, but judge that the protocol is unlikely to change in fundamental correctness as more actors are involved. Detailed statistics of states explored, rules fired, and time are located in Table 1.

[6]The protocol specifies that the Issuer Name is "either the domain of the certified email address in the last certificate, or the issuer listed in the first certificate if the email-address domain does not support BrowserID." Even if RPs notice that the Issuer is the bare, undelegated domain while the BrowserID file indicates delegation – despite the public key being the same – the adversary could react by returning the delegation only to targeted users and returning the previous BrowserID file, bearing just the public key and no delegation, to others, including RPs.

Table 1: States, rules, and runtime for various runs of the protocol

Users	IdPs	RPs	Domains	Properties Tested*	Adversaries	States	Rules	Time	Result
1	1	1	1	valid, IdP, fallback	net, server	388	827	.10s	Fail (IdP)
1	1	1	1	valid, fallback	net, server	1869	4741	.10s	Fail (fallback)
1	1	1	1	valid	net, server	1,102,049	3,178,443	55.43s	Success
1	2	1	2	valid	net, server	9,988,876	31,738,384	1031.37s	Success

* Security properties are as described in Sections 3.3 and 4.5.
"Valid" indicates the mutual authentication and network adversary secrecy properties.
"IdP" indicates the property that malicious IdPs are not contacted and that they never learn credentials.
"Fallback" indicates that an adversary cannot establish a shared secret with the fallback IdP.

6. CONCLUSIONS

Our analysis is constrained by the realities of modeling a complex, multiparty protocol in Murϕ while ensuring correctness and termination. This analysis is consciously limited to the messages sent and state maintained by parties in the protocol; we do not attempt to analyze the full range of threats present on the web, including those presented by more standard web attackers. This is an important area for further exploration.

Our work indicates that BrowserID is secure under the exact conditions under which the protocol designers were operating. Nevertheless, slight, reasonable increases in adversary power lead to the failure of the protocol's fundamental security guarantees, even in our constrained model.

APPENDIX

A. DEMONSTRATED ATTACK DETAILS

We have demonstrated the second Server Adversary attack described in Section 5.2.

The experimental setup was a Mac OS X 10.7 laptop running the most recent version of the Persona verifier, available from `https://github.com/mozilla/browserid`. A remote server owned by one of the authors was configured to serve from two domains, which we will denote as *compromised.com* and *evil.com*. Each was configured with a valid SSL certificate.

compromised.com was configured to delegate its authority to *evil.com*. This simulates the server adversary compromising the BrowserID file at *compromised.com* and redirecting it to the malicious *evil.com*. The BrowserID files for each follow.

`https://compromised.com/.well-known/browserid:`
```
{
  "authority": "evil.com"
}
```

`https://evil.com/.well-known/browserid:`
```
{
  "authentication": "/browserid/sign_in.html",
  "provisioning": "/browserid/provision.html",
  "public-key": {excised for brevity}
}
```

The public key served by *evil.com* is the one used by the IdP key signer at *compromised.com* to sign certificates issued by *compromised.com*. This can easily be determined by the attacker: it is simply the public key served by *compromised.com* before it was compromised. The IdP at *evil.com* would simply forward the request on to the real IdP for *compromised.com*, which for the purposes of our test accepted any credentials.

We supplied an Active Identity *user@compromised.com* to the sign-in website included in the Mozilla source code. The sign-in succeeded, even though the certificate was listed as "issued by *compromised.com*" yet returned from the *evil.com* delegated IdP. This matches the protocol, which states that a certificate can always be accepted from the domain of the user's Active Identity, in this case *user@compromised.com*. We can rule out the verifier having ever cached the public key for *compromised.com* because it never served a public key to the verifier used in the experiment. The verifier checked the returned user certificate using the key from the delegated IdP but did not assure that the certificate had actually been issued by that IdP.

This demonstrates the potential for a server adversary to carry out a silent Man-in-the-Middle attack on Persona/BrowserID authentication process. The malicious IdP could capture the user's credentials, forward them to the true IdP, and simply pass the returned certificate back to the user. No error is detected by the verifier and no notification is made to the user. And, critically, a file upload vulnerability's effects have been amplified into a significant compromise of user credentials.

B. REFERENCES

[1] D. Akhawe, A. Barth, P. Lam, J. Mitchell, and D. Song. Towards a formal foundation of web security. In *Computer Security Foundations Symposium (CSF), 2010 23rd IEEE*, pages 290 –304, July 2010.

[2] G. Bai, J. Lei, G. Meng, S. Venkatraman, P. Saxena, J. Sun, Y. Liu, and J. Dong. AUTHSCAN: Automatic extraction of web authentication protocols from implementations. In *Proceeding of the Network and Distributed System Security Symposium (NDSS)*, 2013.

[3] K. Bhargavan, C. Fournet, A. Gordon, and N. Swamy. Verified implementations of the information card federated identity-management protocol. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 125–135. ACM, 2008.

[4] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[5] M. Hackett and K. Hawkey. Security, privacy and usability requirements for federated identity. In *Web 2.0 Security and Privacy (IEEE Symposium on Security and Privacy)*, 2012.

[6] M. Hanson, D. Mills, and B. Adida. Federated browser-based identity using email addresses. In *W3C Workshop on Identity*

in the Browser, 2011.

[7] A. J. Hu, D. L. Dill, A. J. Drexler, and C. H. Yang. Higher-level specification and verification with BDDs. In *Workshop on Computer-Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 82–95, 1992. Montreal, Quebec, June 29 – July 2, 1992.

[8] M. Miculan and C. Urban. Formal analysis of Facebook Connect single sign-on authentication protocol. In *SOFSEM*, volume 11, pages 22–28, 2011.

[9] S. Mittal and B. Phillips. CS259 project report: OpenID phishing analysis. `http://www.stanford.edu/class/cs259/WWW11/`.

[10] Mozilla Identity Team. BrowserID protocol. `https://github.com/mozilla/id-specs/blob/prod/browserid/index.md`.

[11] Mozilla Identity Team. Cryptography. `https://developer.mozilla.org/en-US/docs/persona/Crypto`.

[12] Mozilla Identity Team. Mozilla Identity Team Blog. `http://identity.mozilla.com`.

[13] L. Seitzer. If the DHS is serving malware, should it be our internet cop? *PC Magazine, Security Watch*, June 28, 2010.

[14] B. Shapero and S. Iyer. CS259 project report: OpenID attribute exchange. `http://www.stanford.edu/class/cs259/WWW11/`.

[15] P. Sovis, F. Kohlar, and J. Schwenk. Security analysis of OpenID. *Sicherheit'10*, 170:329–340, 2010.

[16] S. Sun, K. Hawkey, and K. Beznosov. Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. *Computers & Security*, 2012.

[17] B. van Delft and M. Oostdijk. A security analysis of OpenID. *Policies and Research in Identity Management*, pages 73–84, 2010.