Formal security analysis of authentication in an asynchronous communication model

Bsc thesis presentation

Jacob Wahlgren & Sam Hedin

June 18

KTH

Outline

Introduction

Topic

- Secure Data Sharing protocol
- Formal analysis tool Tamarin Prover
- Focus on authentication

Relevance of formal security analysis

- More complex systems
- High cost of failure
- Competitive advantage
- Authenticity is key

Objectives

Does the authentication in the Secure Data Sharing protocol satisfy its intended security goals?

Objectives

Does the authentication in the Secure Data Sharing protocol satisfy its intended security goals?

1. Identify key concerns
2. Model the protocol and formulate requirements as lemmas
3. Draw conclusions about the security

Background

Formal verification

1. Create a model of the system
2. Describe the properties to be verified
3. Execute the tool to confirm if properties hold

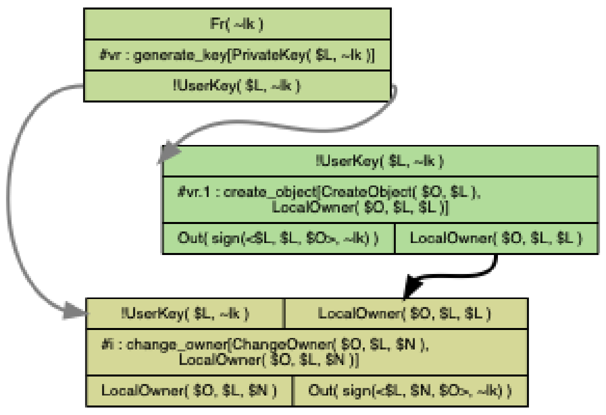
Tamarin Prover introduction

- Verification tool for security protocols
- Developed at ETH Zürich
- Inspired by ProVerif
- Used to analyse 5G authentication and TLSv1.3

Tamarin Prover modeling

- System state is a multiset of facts
- State transitions specified by rewriting system, using rules
- Properties to verify specified by lemmas using temporal logic
- Resulting traces are visualised

Tamarin Prover example



Example trace visualisation showing rules and facts.

Authenticity

Authenticity

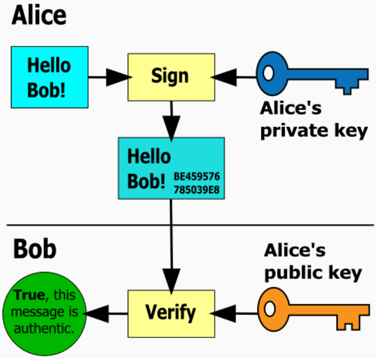
Ensuring that you can know who sent a message.

Authenticity

Ensuring that you can know who sent a message.

- Authentication = confirming authenticity
- Replay attacks

- Used for authentication
- Asymmetric cryptography
- Encrypt hash of the message



Secure Data Sharing

The SDS protocol

- Proprietary protocol, public security model
- Developed by Stockholm startup
- Not released yet

Protocol goals

- Encrypted data stored on server
- Access control per data item
- Clients participate asynchronously
- Data items kept up to date through notifications

Security requirements

In end-to-end mode:

- Only user's device is trusted
- Confidentiality, integrity, authenticity, authorization
- Untrusted server operator

In basic mode:

- Assume honest server operator

Methods

Methods

Methods

- We began by studying different tools used for formal analysis, and the theory of information security.

Methods

- We began by studying different tools used for formal analysis, and the theory of information security.
- We then followed the tutorials in the Tamarin Prover manual to learn the basics.

Methods

- We began by studying different tools used for formal analysis, and the theory of information security.
- We then followed the tutorials in the Tamarin Prover manual to learn the basics.
- As a finishing preparation, we familiarized ourselves with the SDS protocol. We worked with v 0.03 and 0.04

Results

Results

Results

- The algorithms in the system were modeled in Tamarin Prover with the use of rules.

Results

- The algorithms in the system were modeled in Tamarin Prover with the use of rules.
- Sanity checks - `exists-trace` - lemmas were written.

Results

- The algorithms in the system were modeled in Tamarin Prover with the use of rules.
- Sanity checks - `exists-trace` - lemmas were written.
- Authenticity requirements for each algorithm were identified, and expressed as lemmas.

Results

- The algorithms in the system were modeled in Tamarin Prover with the use of rules.
- Sanity checks - `exists-trace` - lemmas were written.
- Authenticity requirements for each algorithm were identified, and expressed as lemmas.
- Tamarin was used to prove or disprove the lemmas.

Owner event: Background

Owner event: Background

- The system contains data objects, which consists of fields of data.

Owner event: Background

- The system contains data objects, which consists of fields of data.
- An object can be shared between users. Different users can have different access levels.

Owner event: Background

- The system contains data objects, which consists of fields of data.
- An object can be shared between users. Different users can have different access levels.
- Each object has one owner, ownership can be transferred with the use of an `owner event`

Owner event: Background

- The system contains data objects, which consists of fields of data.
- An object can be shared between users. Different users can have different access levels.
- Each object has one owner, ownership can be transferred with the use of an `owner event`
- Owner events are authenticated with a digital signature. In the analysed version of the protocol, the signature was computed by concatenating the user ids of the current owner, the new owner and the object id.

Owner event: Model

Owner event: Model

- The creation of an object is modeled by the rule `create_object`.

Owner event: Model

- The creation of an object is modeled by the rule `create_object`.
- When an object is created, the owner event signature is sent out to the network.

Owner event: Model

- The creation of an object is modeled by the rule `create_object`.
- When an object is created, the owner event signature is sent out to the network.
- ```
  rule create_object:
  //If there is a key for a user L
  [!UserKey($L, ~lk)]
  ```
  ```

Owner event: Model

- The creation of an object is modeled by the rule `create_object`.
- When an object is created, the owner event signature is sent out to the network.
- ```
    rule create_object:
    //If there is a key for a user L
    [!UserKey($L, ~lk)]
  ```
- ```
  //It is posssible for the user to create an object
  //and become its local owner
    --[CreateObject($O, $L),
       LocalOwner($O, $L, $L)]->
  ```

Owner event: Model

- The creation of an object is modeled by the rule `create_object`.
- When an object is created, the owner event signature is sent out to the network.
- ```
  rule create_object:
  //If there is a key for a user L
  [!UserKey($L, ~lk)]
  ```
- ```
  //It is posssible for the user to create an object
  //and become its local owner
    --[CreateObject($O, $L),
       LocalOwner($O, $L, $L)]->
  ```
- ```
  //In this case, the owner event is self-signed
    [Out(sign(<$L, $L, $O>, ~lk)),
     LocalOwner($O, $L, $L)]
  ```
  ```

Owner event: Model pt. 2

Owner event: Model pt. 2

- The rule receive_owner takes in a signature and records the stated object owner.

Owner event: Model pt. 2

- The rule `receive_owner` takes in a signature and records the stated object owner.
- Finally, the `change_owner` rule allows the owner of an already existing object to be changed.

Owner event: Model pt. 2

- The rule `receive_owner` takes in a signature and records the stated object owner.
- Finally, the `change_owner` rule allows the owner of an already existing object to be changed.
- Because of how bizarre and terse Tamarin Prover code can be, I will not go into much more code in this presentation.

Owner event: Model pt. 2

- The rule `receive_owner` takes in a signature and records the stated object owner.
- Finally, the `change_owner` rule allows the owner of an already existing object to be changed.
- Because of how bizarre and terse Tamarin Prover code can be, I will not go into much more code in this presentation.
-

```
rule receive_owner:
[!UserKey($PP, ~ppk), !UserKey($P, ~pk),
  In(signature), LocalOwner($O, $L, $PP)]

--[IfTrue(verify(signature, <$PP, $P, $O>, pk(~ppk))),
  LocalOwner($O, $L, $P)]->

[LocalOwner($O, $L, $P)]

rule change_owner:
[!UserKey($L, ~lk), LocalOwner($O, $L, $L)]
--[ChangeOwner($O, $L, $N), LocalOwner($O, $L, $N)]->
[LocalOwner($O, $L, $N), Out(sign(<$L, $N, $O>, ~lk))]
```

Owner event: Lemmas

Owner event: Lemmas

- `private_key_secret`: Ensure the confidentiality of the private key used for signing the event. This proves that the adversary cannot obtain the signing key and break authenticity that way.

Owner event: Lemmas

- `private_key_secret`: Ensure the confidentiality of the private key used for signing the event. This proves that the adversary cannot obtain the signing key and break authenticity that way.
- `local_owner_is_authentic`: Show that the origin of owner events is authentic.

local_owner_is_authentic

```
lemma local_owner_is_authentic:
```

local_owner_is_authentic

```
  lemma local_owner_is_authentic:

//If a user is the owner at time i
  "All obj localuser owner #i. LocalOwner(obj, localuser, owner) @i
```

local_owner_is_authentic

```
  lemma local_owner_is_authentic:

//If a user is the owner at time i
  "All obj localuser owner #i. LocalOwner(obj, localuser, owner) @i

//there must previously have been an owner
//who created a change owner event before time i
  ==> (Ex prev_owner #j. ChangeOwner(obj, prev_owner, owner) @j
       & not(#i < #j))
```

local_owner_is_authentic

```
  lemma local_owner_is_authentic:

//If a user is the owner at time i
  "All obj localuser owner #i. LocalOwner(obj, localuser, owner) @i

//there must previously have been an owner
//who created a change owner event before time i
  ==> (Ex prev_owner #j. ChangeOwner(obj, prev_owner, owner) @j
       & not(#i < #j))

//or the user created the object themselves.
  | (Ex #j. CreateObject(obj, owner) @j & not(#i < #j))"
```

Owner event: Lemmas pt. 3

Owner event: Lemmas pt. 3

- We found that the lemma `local_owner_is_authentic` only provides non-injective agreement.

Owner event: Lemmas pt. 3

- We found that the lemma `local_owner_is_authentic` only provides non-injective agreement.
- This means that the apparent owner at some point actually was the owner, however the signature is not unique per event.

Owner event: Lemmas pt. 3

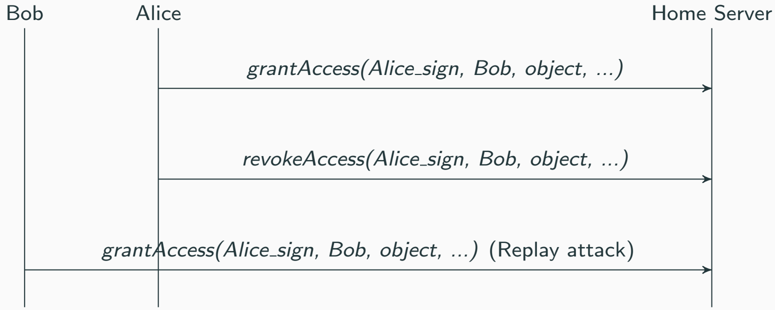
- We found that the lemma `local_owner_is_authentic` only provides non-injective agreement.
- This means that the apparent owner at some point actually was the owner, however the signature is not unique per event.
- Because the signature may be reused, it is possible to fake ownership if one was previously the owner of the object through a replay attack (explained next slide).

Owner event: Lemmas pt. 3

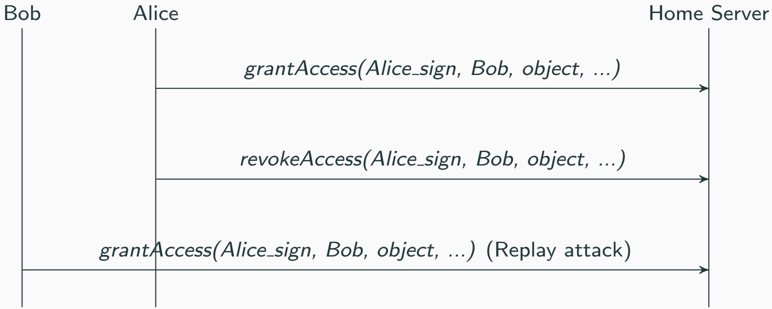
- We found that the lemma `local_owner_is_authentic` only provides non-injective agreement.
- This means that the apparent owner at some point actually was the owner, however the signature is not unique per event.
- Because the signature may be reused, it is possible to fake ownership if one was previously the owner of the object through a replay attack (explained next slide).
- A way to remedy this is to make signatures unique for each event. For instance, an incrementing serial number could be added to each signature.

Example replay attack

Example replay attack

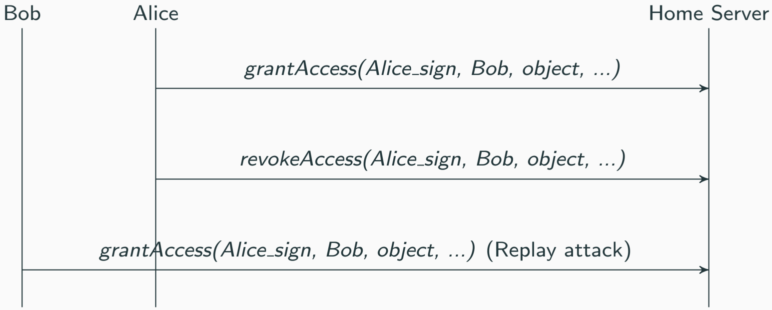


Example replay attack



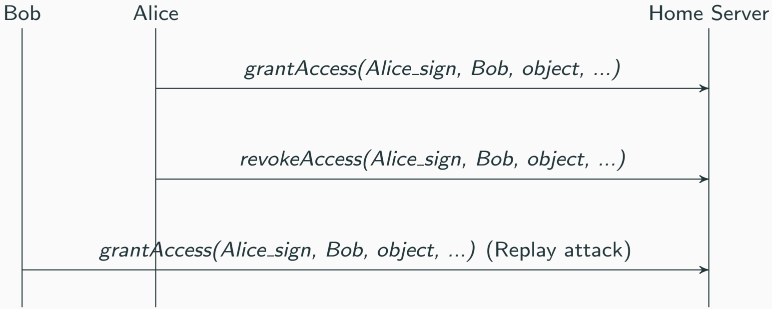
- The diagram shows an example of a possible exploit which exists when signatures can be re-used between events.

Example replay attack



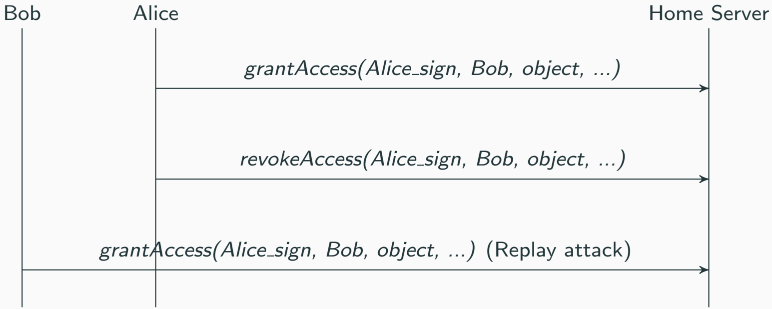
- The diagram shows an example of a possible exploit which exists when signatures can be re-used between events.
- Alice wishes to grant some level of access of an object to Bob.

Example replay attack



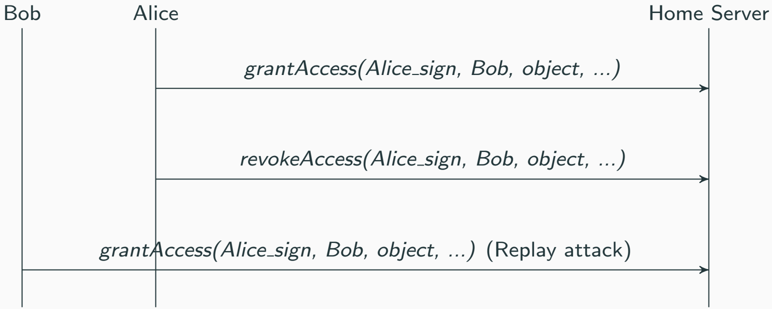
- The diagram shows an example of a possible exploit which exists when signatures can be re-used between events.
- Alice wishes to grant some level of access of an object to Bob.
- Alice does this by creating and signing the event *grantAccessEvent* with *Alice_sign*.

Example replay attack



- The diagram shows an example of a possible exploit which exists when signatures can be re-used between events.
- Alice wishes to grant some level of access of an object to Bob.
- Alice does this by creating and signing the event *grantAccessEvent* with *Alice_sign*.
- Shortly thereafter, Alice revokes the previously granted access with *revokeAccess*.

Example replay attack



- The diagram shows an example of a possible exploit which exists when signatures can be re-used between events.
- Alice wishes to grant some level of access of an object to Bob.
- Alice does this by creating and signing the event *grantAccessEvent* with *Alice_sign*.
- Shortly thereafter, Alice revokes the previously granted access with *revokeAccess*.
- However, Bob copied Alice's signature and uses it to grant himself illegitimate access to the object.

Owner event: Summary

Owner event: Summary

- The signing key cannot be obtained by an attacker.

Owner event: Summary

- The signing key cannot be obtained by an attacker.
- The owner event only provided a weaker form of authentication in the analysed version.

Owner event: Summary

- The signing key cannot be obtained by an attacker.
- The owner event only provided a weaker form of authentication in the analysed version.
- In version 0.06 of the protocol, an incrementing number was added to the signature to enforce ordering of owner events and provide stronger authenticity, based on our suggestion.

Owner event: Summary

- The signing key cannot be obtained by an attacker.
- The owner event only provided a weaker form of authentication in the analysed version.
- In version 0.06 of the protocol, an incrementing number was added to the signature to enforce ordering of owner events and provide stronger authenticity, based on our suggestion.
- Reset events had a similar issue, which was also resolved in the same manner.

Access event: Background

Access event: Background

- An access event modifies access rights to an entire object or for specific fields in that object.

Access event: Background

- An access event modifies access rights to an entire object or for specific fields in that object.
- Access events are authenticated using a digital signature.

Access event: Background

- An access event modifies access rights to an entire object or for specific fields in that object.
- Access events are authenticated using a digital signature.
- In version 0.03 the signature was computed over the concatenation of the affected label, the user id of the user making the change, the device number, a counter value, and the list of users and what access level they are granted.

Access event: Model

Access event: Model

- The model for access events builds upon the model for owner events. Different access levels are not modeled since they do not affect the authenticity of the messages.

Access event: Model

- The model for access events builds upon the model for owner events. Different access levels are not modeled since they do not affect the authenticity of the messages.
- Granting access to an object is modeled by the `send_access_event` rule. It can be executed by a user who already has access to the object.

Access event: Model

- The model for access events builds upon the model for owner events. Different access levels are not modeled since they do not affect the authenticity of the messages.
- Granting access to an object is modeled by the `send_access_event` rule. It can be executed by a user who already has access to the object.
- The `receive_access_event` rule is then used to receive and verify the event and signature.

Access event: Model

- The model for access events builds upon the model for owner events. Different access levels are not modeled since they do not affect the authenticity of the messages.
- Granting access to an object is modeled by the `send_access_event` rule. It can be executed by a user who already has access to the object.
- The `receive_access_event` rule is then used to receive and verify the event and signature.
- The rule `owner_to_access` acts as a bridge between the access and owner models, it makes sure that the owner is considered to have access as well.

Access event: Lemma

Access event: Lemma

- There is only one lemma for access events - `can_receive` - which verifies that a user with the correct access rights can grant another user similar access rights. For example, the owner of an object should be able to assign a user as an administrator of that object.

Access event: Lemma

- There is only one lemma for access events - `can_receive` - which verifies that a user with the correct access rights can grant another user similar access rights. For example, the owner of an object should be able to assign a user as an administrator of that object.
- The lemma was not originally intended to test any security properties. However, the trace it generated made us realise that the access event does not provide full agreement.

Access event: Lemma

- There is only one lemma for access events - `can_receive` - which verifies that a user with the correct access rights can grant another user similar access rights. For example, the owner of an object should be able to assign a user as an administrator of that object.
- The lemma was not originally intended to test any security properties. However, the trace it generated made us realise that the access event does not provide full agreement.
- In the generated trace, two objects were created, but the access event signature from one object was copied from the other object. Since the signature did not include the object identifier, it could be used in a replay attack to modify access rights for another object.

Access event: Lemma

- There is only one lemma for access events - `can_receive` - which verifies that a user with the correct access rights can grant another user similar access rights. For example, the owner of an object should be able to assign a user as an administrator of that object.
- The lemma was not originally intended to test any security properties. However, the trace it generated made us realise that the access event does not provide full agreement.
- In the generated trace, two objects were created, but the access event signature from one object was copied from the other object. Since the signature did not include the object identifier, it could be used in a replay attack to modify access rights for another object.
- This is an authenticity problem because the signature may be reused in the wrong context.

Access event: Lemma

- There is only one lemma for access events - `can_receive` - which verifies that a user with the correct access rights can grant another user similar access rights. For example, the owner of an object should be able to assign a user as an administrator of that object.
- The lemma was not originally intended to test any security properties. However, the trace it generated made us realise that the access event does not provide full agreement.
- In the generated trace, two objects were created, but the access event signature from one object was copied from the other object. Since the signature did not include the object identifier, it could be used in a replay attack to modify access rights for another object.
- This is an authenticity problem because the signature may be reused in the wrong context.
- A user could incorrectly be given authorization to read, modify, or even delete an object which it should not have access to.

Access event: Lemma

- There is only one lemma for access events - `can_receive` - which verifies that a user with the correct access rights can grant another user similar access rights. For example, the owner of an object should be able to assign a user as an administrator of that object.
- The lemma was not originally intended to test any security properties. However, the trace it generated made us realise that the access event does not provide full agreement.
- In the generated trace, two objects were created, but the access event signature from one object was copied from the other object. Since the signature did not include the object identifier, it could be used in a replay attack to modify access rights for another object.
- This is an authenticity problem because the signature may be reused in the wrong context.
- A user could incorrectly be given authorization to read, modify, or even delete an object which it should not have access to.
- This can be remedied by making the signature in the access event include the object identifier.

Access event: Summary

Access event: Summary

- In version 0.06 of the protocol, the object identifier was added to the signature, based on our suggestion. Therefore, the identified vulnerability no longer exists.

Basic mode: Background

Basic mode: Background

- Users can register and authenticate themselves in the basic mode or end-to-end mode.

Basic mode: Background

- Users can register and authenticate themselves in the basic mode or end-to-end mode.
- In the basic mode, users can register and authenticate with both stateless and stateful devices. Public key cryptography is not used, instead a user's identity is their email address or phone number.

Basic mode: Background

- Users can register and authenticate themselves in the basic mode or end-to-end mode.
- In the basic mode, users can register and authenticate with both stateless and stateful devices. Public key cryptography is not used, instead a user's identity is their email address or phone number.
- The purpose of the following procedure is to verify that API requests to the home server originate from the claimed user. The email channel is assumed to be secure.

Basic mode: Step-by-step

Basic mode: Step-by-step

- Registration and authentication starts with the email address being sent to the home server.

Basic mode: Step-by-step

- Registration and authentication starts with the email address being sent to the home server.
- The home server responds by sending a verification link to the specified email address. The link expires within 10 minutes.

Basic mode: Step-by-step

- Registration and authentication starts with the email address being sent to the home server.
- The home server responds by sending a verification link to the specified email address. The link expires within 10 minutes.
- When the user clicks the link, the server can match the URL to the user id, and thus conclude the request to be authentic. In that case, the server returns a session key which is used to authenticate further requests.

Registration and authentication in the basic mode: Model

Registration and authentication in the basic mode: Model

- The steps described in the specification are represented by the rules `fresh_verification_code`, `click_on_link`, `verify_user` and `create_user`.

Registration and authentication in the basic mode: Model

- The steps described in the specification are represented by the rules `fresh_verification_code`, `click_on_link`, `verify_user` and `create_user`.
- The additional rules `read_email` and `write_email` were used to test what would happen if the adversary could read or write to the email inbox.

Registration and authentication in the basic mode: Model

- The steps described in the specification are represented by the rules `fresh_verification_code`, `click_on_link`, `verify_user` and `create_user`.
- The additional rules `read_email` and `write_email` were used to test what would happen if the adversary could read or write to the email inbox.
- The rule `reveal_user_email` gives the adversary knowledge of the email address for a given user id.

Registration and authentication in the basic mode: Lemma

Registration and authentication in the basic mode: Lemma

- The lemma `can_not_compromise_session_key` states that the adversary can not obtain the session key. This assumes that the adversary does not have access to the email inbox until after the session is established.

Registration and authentication in the basic mode: Lemma

- The lemma `can_not_compromise_session_key` states that the adversary can not obtain the session key. This assumes that the adversary does not have access to the email inbox until after the session is established.
- Since the session key is used as a form of authentication in subsequent operations of the protocol, this ensures that authenticity is upheld.

Registration and authentication in the basic mode: Summary

Registration and authentication in the basic mode: Summary

- Under the assumption that the user's email is secure during the procedure, an attacker cannot gain access to the session key. Thus, authenticity is upheld. This level of security is similar to services with email password reset.

Conclusion

Conclusion

Conclusion

- Our analysis revealed two replay attacks in unpublished versions of the protocol. Therefore, the protocol's requirements on authentication were not fully satisfied.

Conclusion

- Our analysis revealed two replay attacks in unpublished versions of the protocol. Therefore, the protocol's requirements on authentication were not fully satisfied.
- The identified issues have been resolved, based on our suggestions.

Conclusion

- Our analysis revealed two replay attacks in unpublished versions of the protocol. Therefore, the protocol's requirements on authentication were not fully satisfied.
- The identified issues have been resolved, based on our suggestions.
- Our analysis of the basic mode proved the security of the authentication process, assuming that the user's email is not compromised at the time of authentication. We can thus conclude that it satisfies its intended security goals.

Conclusion, pt. 2

Conclusion, pt. 2

- We think that the presented methodology is a good way to analyse security protocols. This is supported by the fact that were able to identify real vulnerabilities in the SDS protocol.

Conclusion, pt. 2

- We think that the presented methodology is a good way to analyse security protocols. This is supported by the fact that were able to identify real vulnerabilities in the SDS protocol.
- We are not sure how well this method would scale to more complex protocols and scenarios for non-expert analysts. As our model grew larger, we had trouble making the verification of lemmas terminate.

Conclusion, pt. 2

- We think that the presented methodology is a good way to analyse security protocols. This is supported by the fact that were able to identify real vulnerabilities in the SDS protocol.
- We are not sure how well this method would scale to more complex protocols and scenarios for non-expert analysts. As our model grew larger, we had trouble making the verification of lemmas terminate.
- With detailed knowledge of the inner workings of Tamarin Prover one might be able to work around these issues, but for a beginner there is not much guidance in the manual in such scenarios.

Conclusion, pt. 2

- We think that the presented methodology is a good way to analyse security protocols. This is supported by the fact that were able to identify real vulnerabilities in the SDS protocol.
- We are not sure how well this method would scale to more complex protocols and scenarios for non-expert analysts. As our model grew larger, we had trouble making the verification of lemmas terminate.
- With detailed knowledge of the inner workings of Tamarin Prover one might be able to work around these issues, but for a beginner there is not much guidance in the manual in such scenarios.
- The automated verification may not even be the greatest value of the methodology. Since the modeling process requires such attention to detail, we could in some cases identify issues during modeling, before even running the tool.

Conclusion, pt. 2

- We think that the presented methodology is a good way to analyse security protocols. This is supported by the fact that were able to identify real vulnerabilities in the SDS protocol.
- We are not sure how well this method would scale to more complex protocols and scenarios for non-expert analysts. As our model grew larger, we had trouble making the verification of lemmas terminate.
- With detailed knowledge of the inner workings of Tamarin Prover one might be able to work around these issues, but for a beginner there is not much guidance in the manual in such scenarios.
- The automated verification may not even be the greatest value of the methodology. Since the modeling process requires such attention to detail, we could in some cases identify issues during modeling, before even running the tool.
- Without the terminology and framework of concepts such as the CIA triad and Lowe's hierarchy it is hard to do a relevant analysis.

Conclusion, pt. 2

- We think that the presented methodology is a good way to analyse security protocols. This is supported by the fact that were able to identify real vulnerabilities in the SDS protocol.
- We are not sure how well this method would scale to more complex protocols and scenarios for non-expert analysts. As our model grew larger, we had trouble making the verification of lemmas terminate.
- With detailed knowledge of the inner workings of Tamarin Prover one might be able to work around these issues, but for a beginner there is not much guidance in the manual in such scenarios.
- The automated verification may not even be the greatest value of the methodology. Since the modeling process requires such attention to detail, we could in some cases identify issues during modeling, before even running the tool.
- Without the terminology and framework of concepts such as the CIA triad and Lowe's hierarchy it is hard to do a relevant analysis.
- Analysis would be easier if all protocol specifications included detailed statements of intended security guarantees.

Future work

Future work

- Our work lead to several changes in later versions of the protocol. To ensure the new authentication is correct, another analysis like the one we performed may be carried out on the current protocol version.

Future work

- Our work lead to several changes in later versions of the protocol. To ensure the new authentication is correct, another analysis like the one we performed may be carried out on the current protocol version.
- For a more complete picture of the security of the SDS protocol, additional security aspects should be analysed. The protocol is, apart from authenticity, also intended to provide confidentiality, integrity and authorization. These can all be analysed using the same method as in this report.

Future work

- Our work lead to several changes in later versions of the protocol. To ensure the new authentication is correct, another analysis like the one we performed may be carried out on the current protocol version.
- For a more complete picture of the security of the SDS protocol, additional security aspects should be analysed. The protocol is, apart from authenticity, also intended to provide confidentiality, integrity and authorization. These can all be analysed using the same method as in this report.
- Another direction is to see if this type of analysis can be made easier. For instance, if it is possible to implement the SDS protocol using ProScript or other types of verification tools.

Thanks!

Thank you for listening! Now time for opposition and questions.

Examiner: Panos Papadimitratos

Supervisor: Mohammad Khodaei