Verification^2 of ADEM

Master's Thesis Project Description

Lasse Meinen

Supervisors: Linard Arquint, Felix Linker, Prof. Dr. Peter Müller
Department of Computer Science, ETH Zurich

Start: 25th September 2023
End: 25th March 2024

1 Introduction

In times of international or domestic armed conflict the Red Cross, Crescent, and Crystal have served as a means for protected parties, such as members of medical sectors and humanitarian organizations, to physically mark their personnel, facilities, or objects as protected under International Humanitarian Law (IHL). However, in recent years cyber operations have increasingly become part of armed conflict, moving conflict to a realm in which physical markings bear no meaning. The International Committee of the Red Cross (ICRC) therefore decided to investigate the idea of a 'digital emblem' — a digital equivalent of the physical markings. [1]

The ensuing research project — in partnership with the Centre for Cyber Trust (CECYT), which is a joint endeavor of ETH Zurich and the University of Bonn — led to the development of an Authentic Digital EMblem (ADEM). [2]

A prototype of the emblem verification procedure has already been implemented in Go and the underlying protocol's security properties have in part been verified using the protocol verifier Tamarin. [3] The goal of this thesis is to verify the implementation's core components using the Gobra program verifier [4] for Go.

2 Background

In the following subsections background on the proposed digital emblem and on code-level verification with the Gobra program verifier is presented.

2.1 ADEM

ADEM proposes a digital emblem, which is a signed JSON Web Token (JWT), marking a set of assets as protected. An emblem, or, more specifically, the public key used to create the emblem's signature, can be *endorsed*. An endorsement is used by party P_1 to attest that a particular public key K belongs to party P_2 and that P_2 may signal protection under IHL. Furthermore, an endorsement may introduce constraints regarding the issued emblems. For example, an endorsement can constrain the lifetime of an issued emblem or only allow a single website to be protected.

Note that, technically, a party can endorse itself. Such *internal endorsements* could, for example, be used to maintain more fine-grained control over which keys are used to protect which assets. A chain of internal endorsements eventually ends in a public key that is not further internally endorsed. Such a public key is called a *root key*. To provide accountability, a party can commit to a root key by encoding it as a Web PKI certificate and successfully submitting the certificate to Certificate Transparency (CT) logs, which are signed by standard certificate authorities. Note that a root key itself might still be endorsed by other parties. It is expected that most verifiers will only accept emblems endorsed by parties they trust. That is, there is a valid chain of endorsements, possibly including an endorsement of the Protected Party (PP)'s root key, ending in the root key of a trusted party.

The verification of an emblem, and possibly an accompanying set of endorsements, returns a set of *security levels*. Depending on what additional information is available to a verifier, they may require varying levels of security to accept an emblem. However, it is expected that in practice most verifiers will only accept emblems with the security level *endorsed-trusted*, which is returned only if there is a valid chain of endorsements ending in the root key of a party trusted by the verifier. The chain of endorsements cannot have an arbitrary shape: An emblem lists an issuing PP whose root key was, possibly indirectly through a chain of internal endorsements, used to create the emblem's signature, and only that root key can be endorsed by other authorities.

ADEM defines the following three security requirements, the latter two of which have been verified using the protocol verifier Tamarin [3].

(1) *Covert inspection* requires that an agent who wishes to verify whether an asset is protected under IHL must be indistinguishable from agents who interact with that asset for other purposes.

(2) *Verifiable authenticity* requires that agents must be able to correctly associate emblems to the issuing PP and the respectively marked assets.

(3) *Accountability* requires that independent parties must be able to identify misbehaving parties. For example, parties might misbehave by issuing a fraudulent emblem, where a fraudulent emblem is to be understood as marking an asset not protected under IHL, akin to the illicit display of a physical red cross.

A prototype implementation of ADEM has been written in Go and can be found on GitHub [5]. It includes components for emblem and endorsement generation and verification, command line support, utilities for root key registration, and example usage snippets.

2.2 Gobra

To build further confidence in the aforementioned ADEM implementation, in particular the verification component, the implementation's core components will be formally verified. To that end, we will make use of Gobra, a modular, deductive program verifier for Go. [4] Gobra uses separation logic style field permissions [6], which specify the heap locations that a statement, an expression, or an assertion may access, and encodes annotated Go programs into the intermediate verification language Viper. [7] Viper provides two verification backends, one based on symbolic execution and one based on verification condition generation. By default, Gobra uses the former. Either way, both backends ultimately use the SMT solver Z3 [8] to discharge proof obligations. Gobra supports many of Go's language features, including interfaces, primitive datatypes, and concurrent primitives such as *goroutines* and *channels*.

Putting it all together, Gobra allows the verification of memory safety, crash safety, absence of data races, and user-provided functional properties — all of which will be considered in the verification of the ADEM codebase.

3 Goals

The goal of the thesis is to formally verify the verification component of the ADEM codebase. It is important to note that the Gobra program verifier currently does not support all features of the Go programming language. Additionally, the ADEM codebase relies on multiple external libraries, such as those for JSON Web Key (JWK) and CT log-related functionality. The analysis's scope may need to be adjusted accordingly.

C1 **Safety verification**

In the first step, we will verify the safety, as is routine, of the verification component of the ADEM Go implementation. That is, we will verify memory safety (e.g. that there are no null-pointer dereferences), crash safety (e.g. no division by zero), and data race freedom of the implementation.

Library functions will simply be annotated to specify memory permissions under the assumption that these annotations are correct based on available documentation. However, it is beyond the scope of this step to analyze or verify the library implementations themselves.

Estimated time: 6 weeks

C2 **Define security properties**

The core of the thesis will be the identification and definition of relevant security properties. Note that these are distinct from the security goals of the protocol itself, which have already been proven using the protocol verifier Tamarin. [3] In essence, the goal is to formalize the following two statements concerning the emblem verification procedure, as outlined in [2]:

> First, the procedure verifies every received token's signature and checks that it is valid with respect to its lifetime. [...]
>
> Second, the procedure determines and returns all the emblem's security levels, which indicate whether an emblem was endorsed and by whom.

Estimated time: 1 weeks

C3 **Verify security properties**

Naturally, the specified security properties will need to be verified. To that end, the previously introduced annotations will be strengthened to provide the required security guarantees.

Estimated time: 6 weeks

C4 **Evaluation**

Finally, we will evaluate the effectiveness and efficiency of our analysis. This could include collecting and evaluating verification time measurements, reasoning about the overhead in terms of lines of code introduced by formal code verification, and a depiction of our experience using Gobra.

Estimated time: 1 week

E1 **Precisely specify library interfaces**

Important parts of the verification procedure depend on external libraries. If time permits, we will identify the most critical of these external function calls and validate their specifications by analyzing the underlying implementations in detail.

For example, consider the following line of code

```
jwtT, err := jwt.Parse(rawToken, jwt.WithKeyProvider(km)),
```

where `jwt` references the imported library `github.com/lestrrat-go/jwx/v2/jwt`, which provides JWK-related functionality. The function call `jwt.Parse` parses and, more importantly, *validates* the signature of a byte-encoded JWT token. However, a JWT may omit a signature by specifying the signature algorithm 'none', as specified in section 3.6 of RFC7518 [9]. This is where the behavior of different libraries could diverge: If the JWT includes a *non-null* signature in addition to the 'none' signature algorithm, some libraries will return an error, while others will accept the token as valid.

By analyzing the library's implementation we can ensure that we have correctly specified its behavior and that the code verification procedure correctly captures edge cases as described above.

Estimated time: 3 weeks

E2 **Minimize adaptations of the codebase**

The ADEM codebase, in particular the verification component, makes frequent use of concurrency primitives, which are known to not be fully supported by Gobra. Should we encounter a concurrency pattern or other language feature that turns out to be challenging to verify, there are two options for how to proceed: (a) rewrite the code to be more easily provable, or (b) investigate more deeply how one could provide the desired properties. For this extension goal, we will aim for option (b) as much as possible. As a consequence of this extension goal, it might be necessary to enhance Gobra to better support the used language features.

Estimated time: 4 weeks

E3 **Precisely relate the Tamarin model with the defined security properties**

ADEM includes the formal definition and proof of authentication and accountability using the protocol verifier Tamarin. The formal proofs are available online. [10] In order to facilitate reasoning about the relevance of the security properties defined in C2, we will investigate how we can relate them to the existing Tamarin model and its properties.

Estimated time: 1 week

References

[1] ICRC, *Digitalizing the Red Cross, Red Crescent and Red Crystal Emblems: Benefits, Risks, and Possible Solutions.* Geneva: ICRC, 2022.

[2] F. Linker and D. Basin, "ADEM: An authentic digital emblem," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, to appear.

[3] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 696–701.

[4] F. A. Wolf, L. Arquint, M. Clochard, W. Oortwijn, J. C. Pereira, and P. Müller, "Gobra: Modular specification and verification of go programs," in *Computer Aided Verification*, A. Silva and K. R. M. Leino, Eds. Cham: Springer International Publishing, 2021, pp. 367–379.

[5] "adem-wg/adem-proto: This repository contains libraries and command line utility support for an authentic digital emblem (adem)," https://github.com/adem-wg/adem-proto, (accessed Sept. 27, 2023).

[6] P. O'Hearn, J. Reynolds, and H. Yang, "Local reasoning about programs that alter data structures," in *Computer Science Logic*, L. Fribourg, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–19.

[7] P. Müller, M. Schwerhoff, and A. J. Summers, "Viper: A verification infrastructure for permission-based reasoning," in *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, ser. LNCS, B. Jobstmann and K. R. M. Leino, Eds., vol. 9583. Springer-Verlag, 2016, pp. 41–62.

[8] "Z3prover/z3: The z3 theorem prover," https://github.com/Z3Prover/z3, (accessed Sept. 28, 2023).

[9] M. B. Jones, "JSON Web Algorithms (JWA)," RFC 7518, May 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7518

[10] "adem-wg/adem-proofs: This repository contains formal proofs of the adem design, encoded for the tamarin model checker." https://github.com/adem-wg/adem-proofs, (accessed Oct. 25, 2023).