HOCHSCHULE
MITTWEIDA
University of Applied Sciences

BACHELORARBEIT

Herr
Marc Ulbricht

Untersuchungen zur zertifikatsbasierten Aktualisierung von verteilten Steuerungssystemen

Mittweida, August 2022

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Untersuchungen zur zertifikatsbasierten Aktualisierung von verteilten Steuerungssystemen

Autor:
Marc Ulbricht

Studiengang:
Angewandte Informatik

Seminargruppe:
IF17wI-B

Erstprüfer:
Prof. Dr.-Ing. Thomas Beierlein

Zweitprüfer:
Andreas Weger, M.Sc.

Einreichung:
Mittweida, 04.08.2022

Verteidigung/Bewertung:
Mittweida, 20.07.2022

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

Research on the certificate-based updating of embedded control systems

Author:
Marc Ulbricht

Course of Study:
Applied Computer Science

Seminar Group:
IF17wI-B

First Examiner:
Prof. Dr.-Ing. Thomas Beierlein

Second Examiner:
Andreas Weger, M.Sc.

Submission:
Mittweida, 04.08.2022

Defense/Evaluation:
Mittweida, 20.07.2022

Bibliografische Beschreibung:

Ulbricht, Marc:
Untersuchungen zur zertifikatsbasierten Aktualisierung von verteilten Steuerungssystemen. – 2022. – 6 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, 2022.

Referat:

Dieses Dokument soll als minimales Beispiel für eine Abschlussarbeit dienen und hat nur einen sehr begrenzten Nährwert.

Inhaltsverzeichnis

1 GPG BASH

```
# if a GPG_SIG_FILE was provided,
# check if it is a valid GPG signature
if [ -f "${ZIP_FILE_PATH}${GPG_SIG_FILE}" ] ; then

        if [ "${GPG_SIG_FILE: -4}" == ".gpg" ] ; then
            echo "pass .gpg files only as argument to the -f
            parameter"
            exit 1
        fi

        echo "checking signature..."

        ${GPG} --verify "${ZIP_FILE_PATH}${GPG_SIG_FILE}"
        "${ZIP_FILE_PATH}${ZIP_FILE}" 2>/dev/null

        IS_VALID_SIGNATURE=$?

        if [ $IS_VALID_SIGNATURE -eq 1 ] ; then
            echo "incorrect GPG signature, check the signature file
            name for errors: ${ZIP_FILE_PATH}${GPG_SIG_FILE}"
            exit 1

        elif [ $IS_VALID_SIGNATURE -eq 2 ] ; then
            echo "invalid GPG signature, check the signature file
            name for errors: ${ZIP_FILE_PATH}${GPG_SIG_FILE}"
            exit 1

        else
            echo "correct signature, resuming update..."
        fi
fi
```

2 C

```c
#include "ed25519.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
int main(int argc, char** argv) {

// public key has to be 32-Byte writable char array
// signature has to be 64-Byte writable char array

long FILE_SIZE;
char *BUFFER;
unsigned char PUBLIC_KEY[32], SIGNATURE[64];


// load public key and signature

FILE *SIGNATURE_FILE = fopen ( argv[2] , "rb" );
if ( !SIGNATURE_FILE ) {
perror(argv[2]), exit(1);
}

fseek( SIGNATURE_FILE , 0L , SEEK_END);
FILE_SIZE = ftell( SIGNATURE_FILE );
if (FILE_SIZE != sizeof(PUBLIC_KEY)+sizeof(SIGNATURE)) fclose(SIGNATURE_

rewind( SIGNATURE_FILE );

BUFFER = calloc( 1, FILE_SIZE + 1 );
if ( !BUFFER ) fclose(SIGNATURE_FILE), fputs("memory alloc fails", stder

if ( 1 != fread( BUFFER , FILE_SIZE, 1 , SIGNATURE_FILE) )
fclose(SIGNATURE_FILE), free(BUFFER), fputs("entire read fails", stderr)
fclose(SIGNATURE_FILE);

for (int i = 0; i < FILE_SIZE; ++i) {
if ( i < sizeof(PUBLIC_KEY)) {
PUBLIC_KEY[i] = ((char *)BUFFER)[i];
}
else
SIGNATURE[i-sizeof(PUBLIC_KEY)] = ((char *)BUFFER)[i];
}

free(BUFFER);
```

```
/*
// open signature, prepare and load it into a buffer
FILE *SIGNATURE_FILE = fopen ( argv[3] , "rb" );
if ( !SIGNATURE_FILE ) perror(argv[3]), exit(1);

fseek( SIGNATURE_FILE , 0L , SEEK_END);
FILE_SIZE = ftell( SIGNATURE_FILE );
rewind( SIGNATURE_FILE );

BUFFER = calloc( 1, FILE_SIZE + 1 );
if ( !BUFFER ) fclose(SIGNATURE_FILE), fputs("memory alloc fails", stderr), ex

if ( 1 != fread( BUFFER , FILE_SIZE, 1 , SIGNATURE_FILE) )
fclose(SIGNATURE_FILE), free(BUFFER), fputs("entire read fails", stderr), exit
fclose(SIGNATURE_FILE);

// write signature from buffer to char array
for (int i = 0; i < sizeof(SIGNATURE); ++i) {
SIGNATURE[i] = ((char *)BUFFER)[i];
}
free(BUFFER);
*/

// load update file

FILE *UPDATE_FILE = fopen ( argv[1], "rb" );
if ( !UPDATE_FILE ) perror(argv[1]), exit(1);

fseek( UPDATE_FILE , 0L , SEEK_END);
FILE_SIZE = ftell( UPDATE_FILE );
rewind( UPDATE_FILE );

BUFFER = calloc( 1, FILE_SIZE + 1 );
if ( !BUFFER ) fclose(UPDATE_FILE), fputs("memory alloc fails", stderr), exit(

if ( 1 != fread( BUFFER , FILE_SIZE, 1 , UPDATE_FILE) )
fclose(UPDATE_FILE), free(BUFFER), fputs("entire read fails", stderr), exit(1);
fclose(UPDATE_FILE);

unsigned char *message = malloc(FILE_SIZE + 1 );
for (int i = 0; i < FILE_SIZE; ++i) {
message[i] = ((char *)BUFFER)[i];
}
free(BUFFER);

// verify integrity of update file
```

```
if (ed25519_verify(SIGNATURE, message, FILE_SIZE, PUBLIC_KEY)) {
printf("\nvalid signature\n");
return 0;
} else {
printf("\ninvalid signature\n");
return 1;
}
free(message);

#include "ed25519.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[]) {

unsigned char seed[32], public_key[32], private_key[64], signature[64];
long FILE_SIZE;
char *buffer;


// open update file and read it into a buffer
FILE *UPDATE_FILE = fopen ( argv[1], "rb" );
if ( !UPDATE_FILE ) perror(argv[1]), exit(1);

fseek( UPDATE_FILE , 0L , SEEK_END);
FILE_SIZE = ftell( UPDATE_FILE );
rewind( UPDATE_FILE );
buffer = calloc( 1, FILE_SIZE + 1 );
if ( !buffer ) fclose(UPDATE_FILE), fputs("memory alloc fails", stderr),

if ( 1 != fread( buffer , FILE_SIZE, 1 , UPDATE_FILE) )
fclose(UPDATE_FILE), free(buffer), fputs("entire read fails", stderr), e
fclose(UPDATE_FILE);
// write update file from buffer to char array
unsigned char *message = malloc(FILE_SIZE + 1 );
for (int i = 0; i < FILE_SIZE; ++i) {
message[i] = ((char *)buffer)[i];
}

free(buffer);
if (ed25519_create_seed(seed)) {
printf("error while generating seed\n");
exit(1);
}
ed25519_create_keypair(public_key, private_key, seed);
ed25519_sign(signature, message, FILE_SIZE, public_key, private_key);
free(message);
```

```
// write signature and public to given file

FILE *KEY_SIG_FILE = fopen(argv[2], "w+");
if (KEY_SIG_FILE == NULL)
{
printf("error opening file\n");
exit(1);
}
for (int i= 0; i < sizeof(public_key); i++) {
fputc(public_key[i], KEY_SIG_FILE);
// Failed to write do error code here.
}
for (int i= 0; i < sizeof(signature); i++){
fputc(signature[i], KEY_SIG_FILE);
}

fclose(KEY_SIG_FILE);
/*
FILE *sig_file = fopen("/home/mulbric9/BA/ed25519/src/signature.txt", "w+");
if (sig_file == NULL)
{
printf("error opening file\n");
exit(1);
}
for (int i = 0; i < sizeof(signature); i++) {
fputc(signature[i], sig_file);
// Failed to write do error code here.
}
// Failed to write do error code here.

fclose(sig_file);


if (ed25519_verify(signature, message, FILE_SIZE, public_key)) {
printf("valid signature\n");
} else {
printf("invalid signature\n");
}*/

return 0;
}


}
```

Anhang A: UML-Diagramme

...

Eidesstattliche Erklärung

Hiermit versichere ich – Marc Ulbricht – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Mittweida, 20. Juli 2022	
Ort, Datum	Marc Ulbricht