UNIVERSITY OF MARYLAND
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENEE 457
Computer Systems Security
Instructor: Dana Dachman-Soled
Prepared by: Annette Keller

Homework 2: Security Application with Cryptographic Implementations

Out: 11/04/19 Due: 11/11/19 10:59am

Student Name:

1 Introduction

This homework is intended to be a fast and lightweight overview of some application feature and implementation references that may be helpful for a Build-It/Break-It project. Each page of this homework should only take a few minutes and it's worth half the credit of a problem-solving homework. Extra credit is available for getting a head start on your Build-It/Break-It project.

2 Example Security Application: Logging (15 points)

a. How Does SIEM Logging Work?

What is SEIM and how does it work?

b. Standards and Best Practices for SIEM Logging

Why are secure logging best practices important?

c. Application Security Threats, Attacks

What 5 application security threats or defenses against application security threats could be on a shortlist for an application that manages access to a room with a hardware device like a cipher lock?

☐	Threat Modeling	☐	Credential Stuffing
☐	Input Validation	☐	Session Management
☐	Bean Validation	☐	Buffer Overflow
☐	Reverse Engineering Tampering	☐	Parameter Manipulation
☐	Application Logging	☐	Exception Management
☐	Bad Authorization Logic	☐	Deserialization
☐	Correct Cryptographic Primitives	☐	Secure Coding Standards

You may include any defenses of your choices here (optional)

3 Cryptographic Primitives (25 points)

a. **Understanding Cryptographic Primitives**

What are cryptographic primitives?

What cryptographic primitives may be used for source authentication of a message?

What cryptographic primitives may be used for ensuring message integrity?

b. **Cryptographic primitives in blockchains**

What are the top 4 cryptographic primitives used in blockchains?

What are the usages of hash functions in blockchains? (Just list them, no description necessary)

The digital signature primitive is described as having 3 uses. What are they?

c. **ENEE457 Computer Systems Security: Fall 2019 Lecture Summaries**

What cryptographic primitives have we covered in class?

4 Secure Coding Best Practices (15 points)

a. Cryptographic Best Practices

What is a best practice for selecting a random number generator?

b. The Do’s and Don’ts of Writing Crypto Code

What is "rolling your own" in crypto coding? Is it a do or a don't?

c. SEI Cert C Secure Coding Standard

Secure applications can be compromised by memory errors. Name a memory management error that can result in the leak of cryptographic secrets even if your application is coded correctly, the system doesn't run out of resources and you don't have any software vulnerabilities?

5 Code Review of OpenSSL Implementation (20 points)

In engineering workplaces, you may get "help" on how to do something by being handed some old code to read, and then you have to infer how the process or principles were applied from your reading of the code. Below is a listing of function calls in an OpenSSL implementation of some cryptographic code that you could be given as a reference for how to add some secure features to a design project. It's in C++ but it's readable to you with your C background. The function headers are listed as top level. For extra clarity, functions they call are also listed hierarchically. The code base is in Ian Bull's github repository. Luckily, he has also written a tutorial article, Code Signing and Verification with OpenSSL, explaining the code.

```
RSA* createPrivateRSA(std::string key)
RSA* createPublicRSA(std::string key)
bool RSASign( RSA* rsa, const unsigned char* Msg, size_t MsgLen, unsigned char** EncMsg, size_t*
MsgLenEnc)
     EVP_DigestSignInit(m_RSASignCtx,NULL, EVP_sha256(), NULL,priKey)
     EVP_DigestSignUpdate(m_RSASignCtx, Msg, MsgLen)
     EVP_DigestSignFinal(m_RSASignCtx, NULL, MsgLenEnc)
     EVP_DigestSignFinal(m_RSASignCtx, *EncMsg, MsgLenEnc)
     EVP_MD_CTX_cleanup(m_RSASignCtx)
bool RSAVerifySignature( RSA* rsa,  unsigned char* MsgHash, size_t MsgHashLen, const char* Msg,
size_t MsgLen, bool* Authentic)
     EVP_DigestVerifyInit(m_RSAVerifyCtx,NULL, EVP_sha256(),NULL,pubKey)
     EVP_DigestVerifyUpdate(m_RSAVerifyCtx, Msg, MsgLen)
     EVP_DigestVerifyFinal(m_RSAVerifyCtx, MsgHash, MsgHashLen)
     EVP_MD_CTX_cleanup(m_RSAVerifyCtx)
void Base64Encode(const unsigned char* buffer,  size_t length, char** base64Text)
     BIO_new(BIO_f_base64())
     BIO_new(BIO_s_mem())
     bio = BIO_push(b64, bio)
     BIO_write(bio, buffer, length)
     BIO_flush(bio)
     BIO_get_mem_ptr(bio, &bufferPtr)
     BIO_set_close(bio, BIO_NOCLOSE)
     BIO_free_all(bio)
size_t calcDecodeLength(const char* b64input)
void Base64Decode(const char* b64message, unsigned char** buffer, size_t* length)
     BIO_new_mem_buf(b64message, -1)
     BIO_new(BIO_f_base64())
     BIO_push(b64, bio)
     BIO_read(bio, *buffer, strlen(b64message))
char* signMessage(std::string privateKey, std::string plainText)
     RSASign(privateRSA, (unsigned char*) plainText.c_str(), plainText.length(), &encMessage,
     &encMessageLength)
     Base64Encode(encMessage, encMessageLength, &base64Text)
bool verifySignature(std::string publicKey, std::string plainText, char* signatureBase64)
     Base64Decode(signatureBase64, &encMessage, &encMessageLength)
     RSAVerifySignature(publicRSA, encMessage, encMessageLength, plainText.c_str(),
     plainText.length(), &authentic)

int main() {
  std::string plainText = "My secret message.\n";
  char* signature = signMessage(privateKey, plainText);
  bool authentic = verifySignature(publicKey, "My secret message.\n", signature);
  if ( authentic ) {
    std::cout << "Authentic" << std::endl;
  } else {
    std::cout << "Not Authentic" << std::endl;
  }
}
```

a. What is an RSA object and what element in the chosen plaintext attack (CPA) model does it contain?

b. In the **RSAVerifySignature** function header, identify which parameters are inputs, which are outputs and what they are used for.

c. What is Base 64 encoding and why is it used? There was something that was encoded in Base 64 in the last project. What was that?

d. Ian Bull explains in his article how the **RSASign** function works. How do the EVP functions map to those things he describes?

6. Warm-up for a Build-it/Break-It Access Log Application (25 points)

You are asked to outline a secure logging application for the Build-It/Break-It project. The specifications can be found on the course webpage. Pseudocode is allowable. If you need extra space, please use the extra page on the end of this file. You may append any additional pages you wish to add.

Your outline should include the following:

- description of functions that you plan to create for such an implementation,
- description of program flow,
- any data structures or buffers that may be used, and
- cryptographic approaches you might take for such an application and the cryptographic primitives they might use

7 Extra credit (15 points)

Get a start on your Build-it/Break-it project header file. Except for any cryptographic methods, implement the above with

+ 5 points: function headers
+ 5 points: function comments specifying usage, functionality, inputs and outputs for the function calls