A Security Study for Non-Internet Connected Managed Software

TOMMASO GALASSI DE ORCHI

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

A Security Study for Non-Internet Connected Managed Software

2012/06/30

Tommaso Galassi De Orchi
tgdo@kth.se

This master's thesis project was carried out at SCANIA AB.

Supervisors:

SamuelssonPeter
peter.samuelsson@scania.com
Fransson Jonas
jonas.fransson@scania.com
Jarngren Fredrik
fredrik.jarngren@scania.com

Examiner:

Professor Gerald Q. Maguire Jr
maguire@kth.se

Abstract

This master thesis project aims to improve the security of managed software developed at SCANIA's research and development group NEVE. The thesis will present several security schemes that can be effective against tampering, theft, and reverse engineering of application. The schemes presented were selected to ensure confidentiality, integrity, authenticity, and authentication of applications. NEVE's software will be analyzed and compared against state of the art solutions. A theoretical threat analysis will be presented, corroborated by empirical reverse engineering attacks. The final part of this thesis introduces a new security scheme for C# .NET programs operating without requiring an internet connection.

Sammanfattning

Denna uppsats försöker förbättra säkerheten av [managed software] utvecklad hos SCANIAS forsknings- och utvecklingsgrupp NEVE. Den visar på flera säkerhetslösningar som kan vara effektiva mot manipulation, stöld och omvänd ingenjörskonst av applikationer. De säkerhetslösningar som presenteras valdes för att säkerställa sekretess, integritet, äkthet och autentisering hos applikationer. NEVEs mjukvara kommer att analyseras och ställs mot de allra senaste lösningarna. En teoretisk hotanalys kommer att presenteras, förstärkt med attacker baserat empiriskt omvänd ingenjörskonst. Den sista delen av denna uppsats introducerar en ny säkerhetslösning. Den riktar sig mot program skrivna i C# .NET som inte kräver en uppkoppling mot internet.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Acronym	Explanation
AES	Advanced Standard Encryption
CDB	Credential Database
CIA	Confidentiality Integrity Availability
CIL	Common Intermediate Language
CLR	Common Language Runtime
DB	Database
DES	Data Encryption Standard
ECU	Electronic Control Unit
FCL	Framework Class Library
GSM	Global System for Mobile Communications
HMAC	Keyed-Hashed Message Authentication Code
HOTP	Hash-Based One Time Password
JIT	Just In Time Compilation
MAC	Message Authentication Code
MD5	Message Digest (Version 5)
MSIL	Microsoft Intermediate Language
NEVE	ECU Tools and System Test Transmission
NEVS	System Test Engine
NIST	National Institute of Standards and Technology
OATH	Open Authentication
OTP	One Time Password
PBKDF2	Password-Based Key Derivation Function
PE	Portable Executable
R&D	Research and Development
RAM	Random Access Memory
RE	Reverse Engineering
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
SVM	Secure Virtual Machine
TF-A	Two-Factor Authentication
TOTP	Time-Based One Time Password
XML	Extensible Markup Language

1. Introduction

All around the developed world, governments, finance, defense, and telecommunications industries are major subjects of cyber-attacks from criminals or nation-states seeking advantage in military or economic power. The attacks are so numerous and sophisticated that is hard to determine which threats or vulnerabilities pose the higher risk.

In the recent years the numbers of vulnerabilities discovered in software applications exceeded the numbers of vulnerabilities discovered in operating systems. As a consequence more exploitation attempts are occurring against application programs. Target of the attacks are usually widely used applications. These application are especially vulnerable targets if the producer is unable to release effective patches for the vulnerability.

A new trend is converting trusted web sites into malicious servers in order to target the client side applications and browsers that are commonly present in every computer.

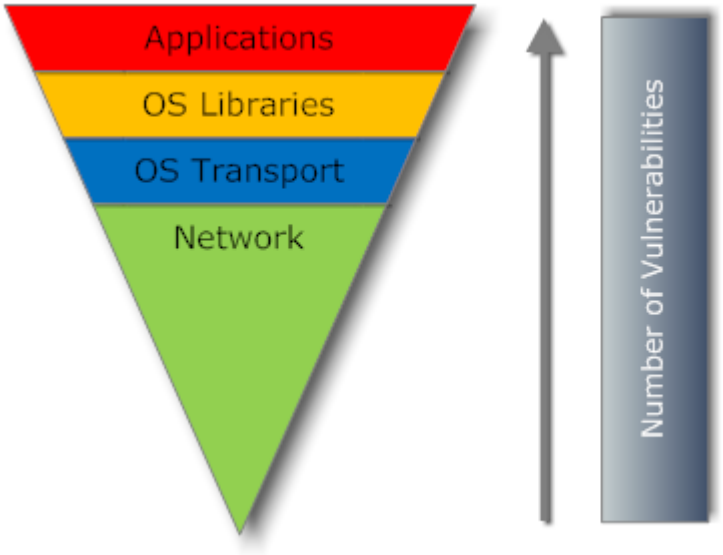


Figure 1 - Number of Vulnerabilities in Network, OS and Applications[1] [24].

A branch of these applications exploits involves software copyright and intellectual property protection. Software intellectual propriety is an extensively discussed topic which involves both technical and ethical issues. One approach to this topic is the production and development of so called *open source.* Open source development is a pragmatic methodology (and philosophy), that promotes free redistribution and access to an end-product's design and implementation details. Open source software is developed and released with the application's source code. This source code allows users, developers, and others to fully understand how the application works in detail. Moreover, open source distribution allows for improvements and sharing of knowledge by a larger community, to accelerating the progress in all fields of science. This methodology/philosophy collides when there is the need to protect intellectual propriety.

[1] The author would like to thank the SANS institute for allowing the use of the image in this master thesis project.

Commercial companies often need to protect their industrial secrets, hence they cannot openly release the source code of their software. The need to release binary versions of the software, while protecting the intellectual property embodied in the source code leads to the need to implement software security mechanisms in order to prevent leakage of industrial secretes, cryptographic keys or proprietary algorithms.

1.1 Background

As the September 2009 report, "The Top Cyber Security Risks", by the SANS Institute faculty analysis reveals, software vulnerabilities are the main threat to IT infrastructures' security. Recent studies have shown that hacking attacks have been successful not only against computer networks, but also against medical equipment (such as peacemakers), vehicles, and electronic voting equipment [19] [31] [32].

A company's research and development division (R&D) is where most of the "know how" and IT assets are developed and stored in the majority of the companies. Failing to protect this information from IT threats, may lead to economic loss, credibility loss, or it can even drive the company to a complete failure. For example, DigiNotar's loss of credibility after the Iranian hackers attack of 19th July 2011, lead the company to bankruptcy in two months and by September of the same year the company was shutdown[2].

SCANIA with a 2011 operating income of SEK 12,398 billion and 37,496 employees[3] has a huge IT infrastructure that permeates almost every aspect of the company.

The NEVE[4] R&D group at SCANIA is currently undertaking a study to evaluate the security of their development tools in order to identify possible threats and to improve the security mechanisms that are currently in place. The team develops with Microsoft .NET a managed code programming framework which could be vulnerable to reverse engineering attacks.

1.1 Overview of the NEVE group at SCANIA

SCANIA's R&D group "ECU Tools and System Test Transmission" (NEVE), develops and tests software tools used to configure trucks' embedded electronics. This electronics controls different engine parameters and, the gearbox, based upon a variety of sensors such as temperature, pressure, *etc*.

This software represents valuable assets for the company and must not fall in unauthorized hands. If some of those tools would leak to the public, anyone would be able to modify engine parameter settings, such as the "speed limit" for the vehicle. Moreover a malicious workshop

[2] The official statement by VASCO of DigiNotar bankruptcy two months later after the attacks, *Source*: http://www.vasco.com/company/press_room/news_archive/2011/news_vasco_announces_bankruptcy_filing_by_diginotar_bv.aspx
[3] Scania end of year report 2011, *Source*: http://www.scania.com/Images/wkr0001_293506.pdf
[4] NEVE is the Electronic Control Unit Tools and System Test Transmission R&D group.

could modify trucks voiding the warranty, and infringing the agreements other companies have made with SCANIA without the risk of being discovered.

The software tools on which the research will focus are developed by the NEVE research group.

There are three key tools and the number of users of these tools is listed in Table 1. Note that for the purpose of this thesis these tools are only identified as Tool-A, Tool-B, and Tool-X. These three tools are the most important software assets of this effort, hence they are the focus of the research in this thesis project.

These software tools are also tightly coupled because they all are used in the development process that leads to the testing phase of the ECU software.

Table 1 – SCANIA's R&D NEVE software assets. Source: SCANIA R&D

Tool	Users
Tool-A	566
Tool-B	321
Tool-X	305
Total Users	**1192**

1.2 Problem Statement

The focus of this thesis project concerns the following question: is it possible to achieve connectionless managed software security?

Despite any security mechanism in place, software ultimately has to be executed on a given computer architectures and their instructions must be understood by the computer's CPU at execution time. This fundamental requirement leaves the above question without a definitive answer.

Knowledge is the parameter that voids any classic computing[5] software protection mechanism. Once we have a working model of how the CPU operates we are able to understand what are the instructions coded inside a software instance.

This problem might find a solution in the quantum computation model where data are represented by physical properties of matter and computation is done over physical

[5] By classic computation the author defines traditional computation in distinction with the quantum computation model.

characteristics of the information represented such as entanglement or laws of motion. Although quantum computation seems very promising field long time will pass before it will be part of humans' everyday life and the need for an immediate working solution emerges.

One approach to address the question raised above is to reduce the knowledge that an attacker has of the software or the computer that the software is to be run on. However, as the computer that the software to be run on is going to be in a manufactured product which we assume the attacker can get physical access to, it is only a matter of time and money to reverse engineer the computer, hence giving the attacker very detailed knowledge of the computer hardware that the software is to be run on. Additionally, the use of non-standard hardware will increase the costs to the company of developing both the hardware and software, and would significantly increase the cost of the hardware platform. For this reason we will not consider this approach further in this thesis.

This leads us to explore the approach of reducing the knowledge of the attacker based upon the software itself. For this reason, the following hypothesis is made: if software protection depends on the knowledge of the coded operations, then adding layers of complexity may slow down the understanding process of an attacker. If the added complexity will slow down the process of understanding, then the following statement is deducted: adding sufficient complexity can secure the software for a relevant amount of time.

This research aims to determine if sufficient complexity can be added in a .NET application to slow down the process of reverse engineering, and thus making it secure for a relevant[6] amount of time.

1.3 Purpose of This Master Thesis

The main task of this thesis project is to identify security threats in proprietary .NET software tools developed at NEVE and to analyze the security of a credential database (the relevance of this database is described in section 5.1.4). The threats will be analyzed and a solution will be presented to address the findings.

The ultimate goal of this master's thesis is to address the following practical question: does adding extra layers of complexity to managed software, delay the understanding of the encoded information in the applications so that by the time they will be decoded, the intellectual property becomes outdated and thus worthless? This thesis project aims to achieve this goal, within the limitations described in section 1.5.

1.4 Methodology

[6] Where "*relevant*" cannot be defined *a priori* but must be defined case by case. For SCANIA major software release are every 6-12 months therefore, we want to secure the applications for a minimum of 12 months.

This research began with an extensive literature study of cryptographic concepts, hash functions, random number generators, and authentication mechanisms. This literature study included .NET concepts and obfuscation techniques.

The knowledge acquired the literature study served as a base for a theoretical analysis of .NET software theoretical analysis, in which software tools were examined in a search for potential threats against known vulnerabilities.

Following this, the project attempted to corroborate the theoretical results with actual software reverse engineering attacks. The empirical findings of this procedure give a deeper understanding of the theoretical analysis, confirming and extending previous results. The threats identified through the theoretical and empirical analysis were evaluated and a new security scheme will be proposed to address the existing issues. The core of this new scheme was developed on Microsoft .Net C#, while other parts consist of security policies, security procedures, and third party security tools. The new scheme was evaluated and compared with previous solutions to illustrate the benefits introduced.

This approach is restricted due to the limitations described in section 1.5.

1.5 Limitations

This Master Thesis report is subject to the following limitations.

Time Limit → The time period for this project was limited to twenty weeks, thus the results of this were expected to be only partially complete. With additional resources better solutions might be achieved.

Project specification → In this project the following requirements had to be met:

1. User friendly solution. Only minor additional work from the users can be required in order to allow psychological acceptability,
2. No connectivity. The solution must not rely on the use of Internet connection or any other private network.
3. No substantial modification to the tools. The development tools should only be slightly affected by the proposed solution.
4. Expense aware solution. The solution provided must not introduce significant additional costs.

Non-disclosure agreement → Please refer to Appendix I – Non Disclosure Agreements. Because this thesis project involves software assets that the company views as having significant value and the thesis project may identify attacks against the company's current software the author has agreed not to disclose certain matters as described in Appendix I.

1.6 Before reading this report

The author of this thesis is subject to a non-disclosure agreement, please refer to *Appendix I – Non Disclosure Agreement,* to learn more about getting access to the complete results of this thesis project (if you are eligible to do so).

In this document the following conventions are in place:

- Alice & Bob are two fictitious characters used to identify two entities that communicate or interact between themselves. They are used for explanatory purposes, i.e. "Alice sends an encrypted message to Bob".
- Tool-A, Tool-B, Tool-X, and Credential Database (CDB) are pseudonyms used in the public version of the report.

1.7 Structure of the Thesis

This section describes the thesis structure. Chapter 1 is an introduction that describes the background and the aims of this project. Chapter 2, 3, and 4 are part of the literature study where security concepts, .Net features, and reverse engineering are described. Chapter 5 introduces SCANIA and the applications subject of this project. Chapter 6 contains a theoretical threat analysis of the .Net software developed by the NEVE group followed by chapter 7, which illustrates empirical reverse engineering attacks performed against the mentioned tools. Finally chapter 9 will summarize the results, which are discussed in details in chapter 10. Conclusions and future works close this document.

2. Security

Security is a chain. It is only as strong as its weakest link [7].

The statement above might sound proverbial, but it succinctly delineates the essential truth about computer security which most companies (and individuals) fail to understand. It does not matter if your company invests one million dollars a year on IT security when someone can just call your IT-helpdesk, say a name and get a valid username and password [14]. A hacker will always exploit the weakest link to maximize their success with the minimal effort. Therefore, the big picture of a system must always be kept in mind when managing security, leaving no minor gaps and considering the human factor (often the weakest link).

Security rests on confidentiality, integrity and availability (CIA). Confidentiality is the concealment of resources and information, integrity refers to the trustworthiness of data or resources and includes origin integrity (sometimes called authentication). Availability refers to the ability to use the desired resource.

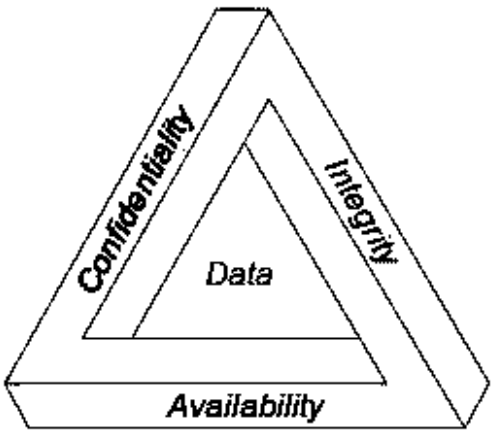


Figure 2 - Confidentiality, Integrity and Availability model (CIA model).

The interpretation of these three aspects of security can vary in relation to the environment, but to ensure that they are correctly implemented security policies and security mechanism must be implemented. It is crucial to understand the difference between a security policy and a security mechanism:

1. A *security policy* is a statement of what is, and what is not, allowed [4].
2. A *security mechanism* is a method, tool, or procedure for enforcing a security policy [4].

Matt Bishop states in his book on computer security: "Given a security policy's specification of secure and non-secure actions, a security mechanism can prevent the attack, detect the attack, or recover from the attack. The strategies may be used together or separately" [4].

[7] The original idiom is "a chain is as strong as its weakest link". The origin is unclear but it is often attributed to Thomas Reid when in 1786 in his "Essays on the Intellectual Powers of Man" he stated: *"In every chain of reasoning, the evidence of the last conclusion can be no greater than that of the weakest link of the chain, whatever may be the strength of the rest."*

This chapter will review the main characteristics of some standard security mechanisms. These protocols have been selected because they are potential candidates to address the problems that might affect NEVE's IT assets. At this stage it is not possible to delimit the whole spectrum of problems, but given the initial conditions of the environment it is possible to pin-point problems that already have a solution in current literature. To better organize the reading, the protocols will be first described in terms of their general characteristics, while chapter 8 will explain how each protocol can solve a problem or part of a problem for NEVE.

2.1 Symmetric and Asymmetric Cryptography

The word cryptography comes from the combination of the two Greek words: κρυπτός (kryptós) which means "hidden", and γραφία (graphía) which means "writing". Cryptography therefore is that branch of mathematics that deals with the methods to conceal information making them meaningless for an unauthorized person.

Cryptanalysis is the opposite of cryptology, from the Greek words: kryptós, "hidden", and analýein, "dissolving" it is the study of methods to understand what it is hiding within a cryptographic message.

One of the techniques of cryptography is symmetric key cryptography. This is a very old scheme[8] and it has been used for thousands of years. This scheme is simple and relies on two operations encryption and decryption (*Figure 3*).

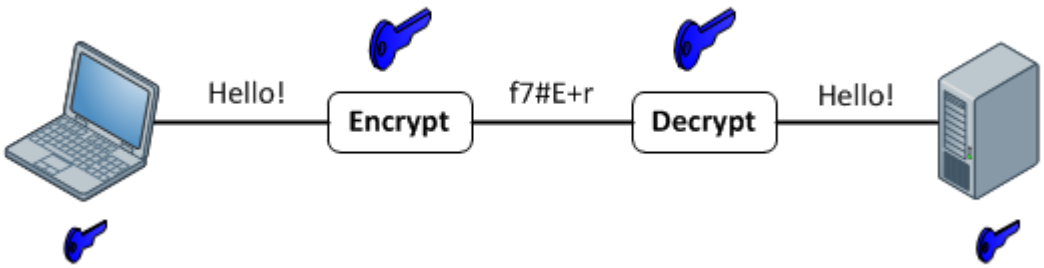


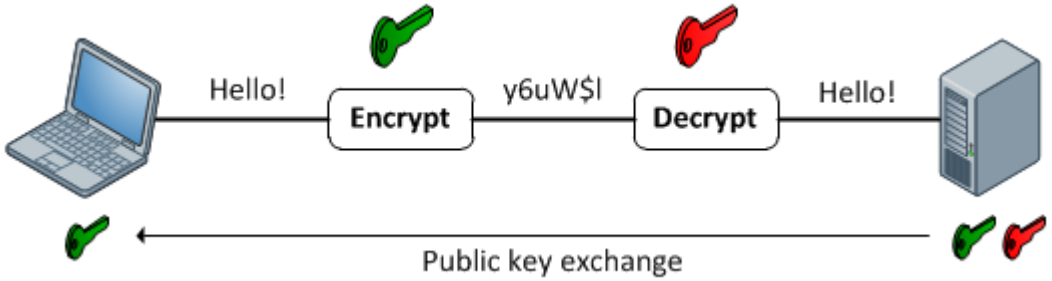
Figure 3 - Symmetric encryption scheme. The blue key represents the secret key.

A message *m* is encrypted with the algorithm *a* and the secret key *k.* To retrieve the original message, the enciphered message *c* is decrypted with the algorithm *a* and the secret key *k.* The strength of this scheme relies on the secrecy of the key used by the two communicating parties, the dimension of the key space used and the choice of the algorithm. The main issue with this kind of cryptographic system is the key exchange phase, where unwanted listeners may eavesdrop and steal the secret. Symmetric key algorithm are usually fast, thus are used in all applications that require high performances. Some examples of the symmetric key scheme are the Caesar Cipher, Data Encryption Standard (DES), and the Advanced Encryption Standard (AES).

[8] The Caesar cipher is named after Julius Caesar, who, according to Suetonius, used it to protect messages of military significance already in 50 b.c. is an example of old symmetric cryptography scheme. *Source:* www.bu.edu/lernet/artemis/years/2011/slides/crypto.pdf

Asymmetric cryptography also known as public key cryptography is a modern cryptographic scheme which involves two keys: a public key and a private key. The main advantage introduced with this scheme is that the two communicating parties do not have to solve the problem of keeping the key secret while they exchange it. This new age of cryptography began in 1976 with the scheme published by Whitfield Diffie and Martin Hellman generally referred as Diffie-Hellman (*Figure 4*).

Figure 4 - Asymmetric encryption model. The green key represents the public key, the red key the private key.



2.2 One Time Password – OTP

"Let M be a random variable that takes values from the set of messages $m_1..m_n$. The cipher C = E(M) achieves perfect secrecy if H(M | C) = H(M)."

M. Bishop, *Computer Security: Art and Science [4]*.

A One Time Password (OTP) is often referred as "*perfect secrecy*". It is a very old cryptosystem[9] which under the right circumstances, produces random outputs that bears no statistical relationship to the input. This property makes any cryptanalysis attempts useless because there is no way to distinguish the real message from all possible messages.

Figure 5 shows a simple example of how OTP works; it is very simple and straightforward. The message's bits are XORed with the random key. If the key is composed by truly random bit the output is a random bit sequence.

[9] A cryptosystem is a 5-tuple (E, D, M, K, C), where M is the set of plaintexts, K the set of keys, C is the set of cipher texts, E: M x K → C is the set of enciphering functions, and D: C x K → M is the set of deciphering functions [4].

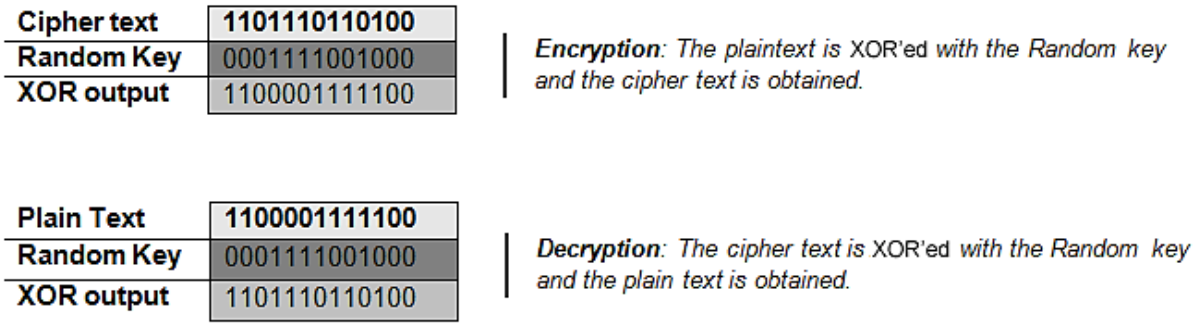


Figure 5 - Basic OTP

To achieve the characteristics of the definition of OTP given above it must meet the following conditions:

1. The key is generated by a true random number generator.
2. The key is only used once, to cipher one message.
3. The key is only known to the legitimate communicating parties.
4. The key length must equal the length of the plaintext.

Unfortunately, OTP has many drawbacks that make its use unpractical in most scenarios. First, the key must equal the length of the message, thus to encrypt a one megabyte of information one megabyte of truly random information is needed. This not only implies a large amount of space to store the key, but a large number of truly random bits is needed, since each key can only be used once! Moreover, to supply true random numbers in large quantities is a significant task.

The problem grows when keys distribution and protection are taken into consideration. To distribute and protect such a large number of big keys size is a complex and risky process which can compromise the whole cryptosystem if not handled properly.

Nonetheless, OTP still find its application in some scenarios and some advanced protocols such as *HMAC-based One Time Password* (HOTP) and *Time-Based One Time Password* (TOPT) are based on the original OTP assumptions although they have different aims.

2.3 Advanced Encryption Standard – AES

The United States of America's National Institute of Standards and Technology (NIST) announced in 2000 that the block cipher Rijndael would become the new Advanced Encryption Standard (AES), after a 3 year study of 15 block ciphers that were competing to become the new standard.

Block ciphers belong to the category of symmetric cryptography. These include ciphers such as Data Encryption Standard (DES), Triple DES, Blowfish, and of course AES.

The Rijndael design philosophy is founded on three basic principles:

1. Keep it Simple,
2. Performance is important, and
3. Use well understood components.

These principles have been at the center of the evaluations of AES's strength. Most of the arguments raised against AES comes directly from this philosophy. This idea that "*any design that can be understood must be insecure*" is rejected by Joan Daemen and Vincent Rijmen (Rijndael authors) who labeled this reasoning as inherently flawed [1].

After more than twelve years there are no known practical attacks that can break AES encryption, only a few algebraic attacks and theoretical attacks that describe a complexity of 2^{176} for AES192 (where 192 stands for the bit length of the key) and 2^{100} for AES256. All these attacks used a weakened version of AES, thus leading to the conclusion that AES is still secure (at least as July 2012).

Although AES is still secure it has all the drawbacks that are commonly associated with symmetric key cryptography.

Key Distribution — If two parties (call them Alice and Bob) want to use AES to communicate privately, they have first to exchange the key through a secure channel[10]. To establish a secure channel can be expensive, both in terms of cost and computational power.

Number of keys — In a network with *n* users, where each pair needs a separate pair of keys, there can be potentially $\frac{n(n-1)}{2}$ pair of keys and every user has to store *n-1* keys. Key management simplifies this process, but leads to a new set of problems which administrators have to deal with.

Non-repudiation — This group of algorithms does not provide non-repudiation of message the origin. Non-repudiation provides protection against denial by one of the entities involved in a communication of having participated in all or part of the communication.

[10] Secure channel: a packet, datagram, octet stream connection, or sequence of connections between two end-points that affords cryptographic integrity and, optionally, confidentiality to data exchanged over it [20].

2.4 Rivest-Shamir-Adleman - RSA

The Rivest-Shamir-Adleman (RSA) cryptographic scheme is nowadays the most used asymmetric cryptographic scheme [21]. RSA is the most successful among the public key scheme in use thanks to its highly versatile use in a wide variety of applications for different task.

Classical cryptography required the sender and the receiver to share a common secret (key) to encrypt and the decrypt a message, while public key cryptography is fundamentally different. In asymmetric cryptography each entity has two keys: a private key and a public key. The public key is known to everyone (hence it is public), while the private key is kept secret by its owner. When Alice needs to send a message to Bob, she will use Bob's public key to encrypt the message. When Bob receives the message he can decrypt it with his private key. Therefore private keys are never exchanged solving the old issue of symmetric cryptography requiring the sharing of the common secret between the parties.

The public-key scheme introduced a radical departure from all previous techniques, as asymmetric cryptography is based on mathematical functions rather than permutations or substitutions. The core of a public key scheme, is that the mathematical problem should be infeasible to solve or at least computationally too demanding to be solved in short time; for example mathematical problems such as finding the discrete logarithm of a random elliptic curve element or factoring large integers (used in RSA).

Public key cryptography must abide by some criteria to work properly:

1. It must be computationally easy to encipher or decipher a message given the appropriate key.
2. It must be computationally *infeasible* to derive the private key from the public key.
3. It must be computationally *infeasible* to determine the private key from a chosen plaintext attack [4].

In *Table 2* RSA is compared to other public key cryptography schemes.

Table 2 – Public Key algorithms comparison [25].

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
Diffie Hellman	No	No	Yes
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes

Before going into the details of how RSA works, let us note on the fact that RSA can offer confidentiality, integrity, and authenticity at the same time. This scheme is extremely successful and it is behind the whole security architecture of today's "web of trust"[11], it is implemented in every browser, web service, and almost all internet related applications.

[11] *Web of trust* (WoT) is a concept used in PGP, GnuPG, and other OpenPGP-compatible systems to establish the authenticity of the binding between a public key and its owner.

2.4.1 RSA algorithm

RSA makes use of modular arithmetic over rings for its encryption/decryption process. When RSA encrypts a plaintext x, the bit string representing the plaintext x must be an element of $\mathbb{Z}_n = \{0, 1, \ldots, n\text{-}1\}$ this implies that the binary string representing the plaintext x and the cipher text must be less than n. $\mathbb{Z}_n$ are the integers modulo n, $\mathbb{Z}n = \{0, 1, \ldots, n-1\}$ with addition and multiplication mod n. Thus, for example, in $\mathbb{Z}_9$ the multiplication of 3 by 4 results in 3 since $12 \equiv 3 \bmod 9$, and therefore 12 is identified with 3.

Let us define some parameters to illustrate the basic RSA algorithm. First some very long numbers are generated with 1024, 2048, or 4096 bit length. Those number are d, e, and n. The pair (n, e) is called the *public key* and d is the *private key*. If Alice wants to send an encrypted message to Bob, she will cipher the plaintext with the *public key* and Bob will decrypt it with his *private key.* The idea is that anyone can access the public key, but knowledge of the public key must insufficient to derive the private key. Therefore if e *and* n are known, it must be computationally infeasible to derive d. This is the key aspect necessary to understand the RSA algorithm.

This section will not cover how d, n, and e are generated, but it is crucial that those numbers abide by certain properties for RSA to work as intended. For more about the generation of these values please refer to section 3 of [5]. The RSA encryption and decryption operations are illustrated below.

RSA Encryption

Given the public key $k_{pub} = (n, e)$ and the plaintext x, RSA encrypts as follows [5]:

$$y = ENCRYPT_{k_{pub}}(x) \equiv x^e \bmod n$$

where $x, y \in \mathbb{Z}_n$.

RSA Decryption

Given the private key $k_{priv} = d$ and the cipher text y, RSA decrypts as follows [5]:

$$x = DECRYPT_{k_{priv}}(y) \equiv y^d \bmod n$$

where $x, y \in \mathbb{Z}_n$.

The relationship between the exponents *e* and *d* ensure that encryption and decryption are inverses therefore the plaintext *x* can be recovered through the decryption process.

The encryption and decryption process must be easy to compute to allow high speed communication and usability for everyday applications [26]. The scheme presented is the basic RSA algorithm which over the years has been refined and implemented in different ways to meet different requirements.

2.5 Keyed-Hash Message Authentication – HMAC

The Keyed-Hash Message Authentication Code (HMAC) describes a mechanism for message authentication using cryptographic hash functions. This technique combines the use of a hash function with a shared key. HMAC's security strength is based on the property of the underlying hash function, therefore HMAC must always be implemented with an approved and secure hashing procedure [2].

A Message Authentication Code (MAC) is a technique to provide integrity for the messages sent between two parties. HMAC expands the abilities of common MAC protocols by adding the property of origin authentication. If only the destination (Alice) and the source (Bob) know the HMAC key, this provides both data integrity and origin authentication for the data exchanged between them. Once the data have been transferred, if the HMAC is correct, this proves that this message must have been transmitted by the source.

Different hash functions can be used for HMAC algorithm and there exist many different implementations. Message Digest version 5 (MD5) and Secure Hash Algorithm version 1 (SHA1) are no longer considered secure [3]. However, HMAC security builds upon its underlying hash function. It is important to track the 2012 NIST SHA3 competition results, because this will probably define the new secure standard hash function for the upcoming years.

Section 2.5.1 will illustrate how HMAC works and show how easy and valuable is to ensure integrity and authenticity while requiring very little computational power.

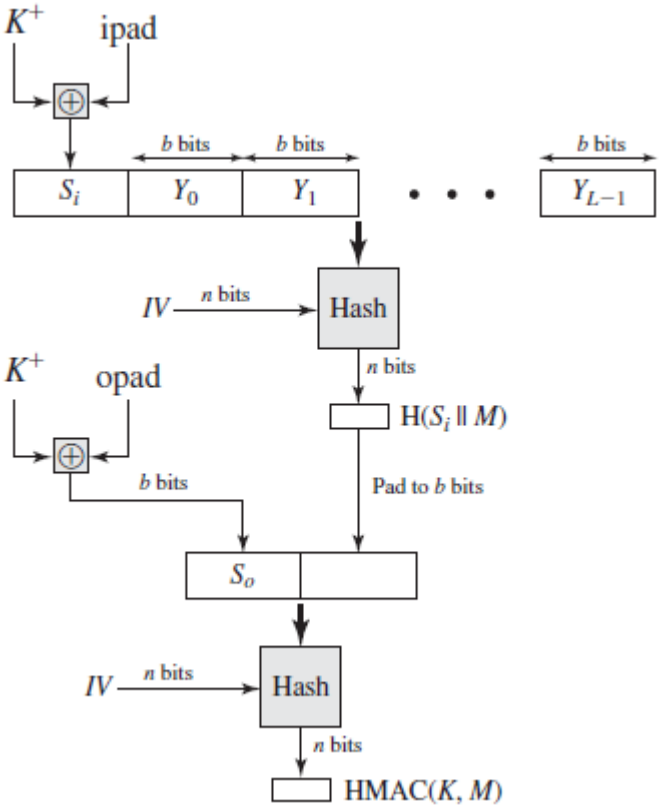
2.5.1 HMAC algorithm

First we define all the symbols and then illustrate the algorithm.

n	is the length of the hash code by the HMAC function,
b	number of bits in a block,
K	is the secret key. The length of the key should be ≥ than n,
H	is the HASH function used (e.g., SHA1, RIPEMD-160),
K^+	is K padded with all zero to the left so that the length is equal to b,
IV	Initialization Vector for the HASH function,
M	it is the message over which the HMAC will be computed,
L	number of blocks in M,
Y_i	ith block of the message M, $0 < i < (L-1)$,

ipad inner pad: the byte 0x36 repeated b times,
opad outer pad: the byte 0x5C repeated b times.

Figure 6 - HMAC schema, W. Stallings, Cryptography and network security : principles and practice[12].



To compute the HMAC over the message M the following calculation is performed (as shown in *Figure 6*):

$$HMAC(K, M) = H[\ (\ K^{+}\ XOR\ opad\ ), H[\ (\ K^{+}\ XOR\ ipad\ ), M\ ]\ ]$$

The algorithm involves the following steps [2]:

1. At the end of K append zeros to create a B byte string,
2. Bitwise exclusive-OR (XOR) *ipad* with the B string computed in step (1),
3. Append the 'message' stream to the B byte string resulting from step (2),
4. Apply H to the output of step (3),
5. XOR the *opad* with the B byte string computed in step (1),
6. Append the hash result from step (4) to the B byte string computed in step (5),
7. Apply H to the output of step (6) to produce the final result [2].

[12] The author would like to thank Williams Stallings for allowing the use of this image in this master thesis project.

2.6 Hash-based One Time Password and Time-based One Time Password

In section 2.2 we reviewed the basic logic behind OTP. The Hash-based One Time Password (HOTP) and Time-Based One Time Password (TOTP) are two advanced algorithms based on the OTP scheme. These algorithms are gaining popularity with the nearly ubiquitous adoption of mobile devices on the market. Both HOTP and TOPT have very low computational requirements and therefore are very suitable for smartphone hardware. Moreover these schemes are suitable for Two-Factor Authentication[13] (TF-A) which fits well with the smart-phone scenario in which devices are constantly connected to Internet services.

Earlier TF-A schemes did not spread widely because most of the implementations were closed, thus they were very expensive. Closed systems also bear an additional side effect, they do not enable others to innovate. HOPT and TOPT are part of a project called Open Authentication (OATH)[14] which aims to specify an algorithm that can be freely distributed to the technical community. The authors believe that "*a common and shared algorithm will facilitate adoption of two-factor authentication on the Internet by enabling interoperability across commercial and open-source implementations.* [8]"

Google has recently adopted TF-A for its mail service Gmail. Google developed a series of mobile applications for Android and Apple iOS devices that allow a user to retrieve the OTP code and to access their mailbox. This additional security layer is very effective against a large variety of internet threats which a simple password cannot prevent.

2.6.1 Hash-based One Time Password – HOPT

The HOTP is very simple and it is based on OTP scheme. It's based on an *increasing counter* value and a *static symmetric key* which are known only to the token and the validation service. The token is what the user "has" and the password represents what the user "knows". The validation service is in the above example, Google's Gmail service.

To demonstrate how HOTP works, HMAC-SHA-1 algorithm will be used, but other scheme based on different hash functions can be used. The output of HMAC-SHA-1 is 160 bits long therefore it must be truncated to something that can be entered by a user easily:

$$HOP\,T(K,C) = Truncate(HMAC - SHA - 1(K,C))$$

[13] *Two-factor authentication* is also called strong authentication. It is defined as two out of the following three proofs: (1) Something known, like a password, (2) Something possessed, like your ATM card, or (3) Something unique about your appearance or person, like a fingerprint. (Source: http://www.rsa.com/glossary/default.asp?id=1056).

[14] Open Authentication – OATH - http://www.openauthentication.org.

where:

K is the *key*

C is the *counter*. *Counter* is the moving factor and must be synchronized between the HOTP validator (server) and the HOTP generator (user client)

$Truncate$ is a function that converts the HMAC-SHA-1 value into an HOTP value. This function is described in section 5.3 of RFC 4226 [8].

The client increments its counter and calculates the next HOTP value. If the authentication server receives a value that matches the value calculated by the client, then the HOTP value is validated and the server increments the counter value by one.

From a security prospective HOTP is still very secure in July 2012. There are no weaknesses in HMAC-SHA-1 that can impact on HOTP, therefore the only attack available at the moment is a brute force attack which requires a sender to authenticate 2^{80} messages before an adversary can create a forgery [8]. The known attacks on SHA-1 do not affect the use of HMAC-SHA-1 as pseudorandom function.

2.6.2 Time-based One Time Password – TOTP

TOTP is a modified version of HOTP. The moving factor in HOTP is an increasing counter while in TOPT the moving factor is based on a time value. Using time as an input produces OTP values that are valid only for a short time interval enhancing security.

Let us first define $T0$ as the Unix time to start counting (i.e. the Unix epoch) and X as the time step in seconds (usually 30 seconds); both these value are system parameters. Then the formulation of TOTP is as follow:

$$TOTP = HOTP(K, T)$$

where:

T is an integer and identifies the number of time steps between current Unix time and the initial counter time $T0$. Thus:

$$T = \frac{(Current\ Unix\ time - T0)}{X}.$$

As for HOTP, a security analysis demonstrates that the only possible attack against the TOTP functions is a brute force attack, since this variant is based directly on HOTP. However, TOTP imposes a special requirement due to the time synchronization seen on section 6 of RFC6238[9].

2.7 Randomness requirements for security

The reasons for including this section come from the difficulty of generating cryptographic keys. The effort required to generate a key is often underestimated. A weak key can facilitate a successful cryptanalysis attack against a cypher text. RSA Laboratories developed a set of standards to define a cryptographic key how to derive from a password. Among their specifications there is the Password-Based Key Derivation Function (PBKDF2), a modern key derivation function. PBKDF2 applies a pseudorandom function to derive keys with two crucial parameters *salt* and *iteration count*. Salt is used to produce always a different key and "iteration count has traditionally served the purpose of increasing the cost of producing keys from a password, thereby also increasing the difficulty of attack" [sic] [6].

The key derivation function (KDF) is applied to a password P and the salt S to derive a key K. The iteration count is the number of times that KDF is applied.

$$K = KDF\ (P, S)$$

A question remains: how to define a good pseudorandom function? Is it possible to generate truly random numbers from a PC? How high is the entropy of the passwords?

Cryptographic algorithms are built to foil pattern based analysis attempts; however, this is completely dependent upon generating random secret quantities for cryptographic keys and passwords. RFC 4086 states: "The use of pseudo-random processes to generate secret quantities can result in pseudo-security."[7]

A truly motivated attacker may reproduce the environment that generated the secret quantities and try to locate them in the entire potential number space. It is a non-trivial task to choose these secret quantities correctly, therefore a small list of methods that can provide a reliable source of randomness will be given below.

There are two methods to approach this problem: hardware and software. Hardware random generators are the best solution and they are increasingly included in today's computer at very low cost.

Thermal noise (e.g. Johnson noise in integrated circuits) or a radioactive decay source are excellent fast source of random quantities, moreover high quality random data can be produced by an audio/video input device such as a microphone recording background noise. Another excellent source could be a spinning disk (hard drive); small random fluctuations are manifest in their rotational speed due to chaotic air turbulence. If this data is correctly processed even slow disk drives on old computers can produce a good amount of random data [7].

If hardware random generators are not available then DES or SHA-1 can be used to generate pseudo-random keys. Both Microsoft and UNIX offer modern solutions which are a combination of different methods to generate random numbers. Although it is important to underline that hardware random generator are increasingly available in new computers (i.e. on UNIX system it is possible to access */dev/sound* to generate random number from background noise).

2.7.1 PBKDF2 alternatives: Bcrypts and Scrypt

It is worth mentioning for completeness that there are alternatives to NIST directives on how to generate secure cryptographic keys. Two of these are Bcrypts and Scrypt.

Bcrypts by Niels Provos and David Mazières, is an adaptive hashing function which aims to be slow and it is based on the Blowfish[15] algorithm. Ideally it is desirable to have the password hashing function be as slow as possible for an attacker while not being intolerably slow for an *honest system user*.

Bcrypt was never officially accepted as a standard, but it has all the characteristics to be one: it has been public for 13 years, it attracted attention, and yet remains unbroken to date.

Some have argued that although *Bcrypt* has not been broken, it has been designed with a mindset of 1999. The response to this is *Scrypt*, a *bcrypt*-like function which requires much more memory. *Scrypt*'s author Colin Percival claims "*We estimate that on modern (2009) hardware, if 5 seconds are spent computing a derived key, the cost of a hardware brute-force attack against* scrypt *is roughly 4000 times greater than the cost of a similar attack against bcrypt (to find the same password), and 20000 times greater than a similar attack against PBKDF2* "[23].

[15] Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use. Blowfish was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms. *Source*: http://www.schneier.com/blowfish.html

3. Microsoft .NET

.NET (pronounced "dot net") is a framework for computing developed by Microsoft. Its purpose is to simplify application development, to make applications portable to different environments and to provide a common programming model where is possible to choose different programming languages to reach the same goal.

The two main components of .NET are the Common Language Runtime (CLR) and the Framework Class Library (FCL). Visual Studio is the glue that holds the entire framework and allows easy collaboration by software developers. The relationship between these components is shown in *Figure 7*.

Figure 7 – The .NET Framework architecture.

Figure 7 illustrates how the CLR translates anything above, to the operating system running below. Therefore, the CLR is a platform that hosts and executes the application and provides all the services that applications need to access resources, (such as operating system folders, arrays, etc.). Thus, the programming code is "managed" by the CLR giving birth to a distinction between "unmanaged" code and this "managed" code. This characteristic (as it will be analyzed later) will lead to several security issues related to copyright, original idea attribution, and code theft, that affect this programming model.

To allow the CLR to provide all these services the language compiler adds metadata to describe the types used to develop in .NET. This metadata are used to handle references to objects and handle or release objects when they are no longer in use.

Compilers in .NET do not generate code to be directly executed by the underlay processor rather they emit an intermediate language code called Common Language Runtime (CLR). CLR is a platform independent object-oriented version of assembly code, when this code is executed a Just In Time Compilation (JIT) process will convert CLR into the native processor instruction set of the host system and then this native code is executed.

The Framework Class Library (FCL) is a new set of functions which contains thousands of types. Basically all the APIs, libraries, and DLLs used in previous programming languages (C++,

Visual Basic, etc.) have been packed into a unique collection named FCL. To better manage the FCL, Microsoft has divided it into about 100 hierarchical name spaces (*Figure 8*). Each name space contains types and the classes that share a common purpose, for example the namespace System.IO contains all the classes needed for Input/output operations.

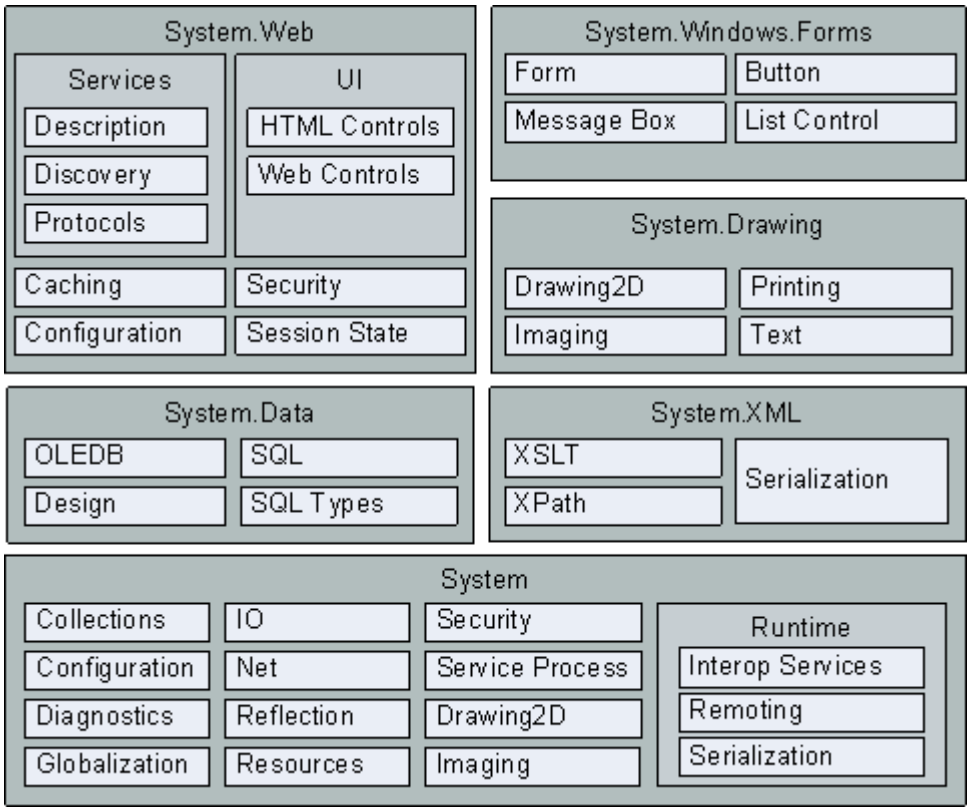


Figure 8– A simple schematic representation of the FCL.

As mentioned above this model of managed code gives birth to a series of security problems. The MSIL language can be easily decompiled into something that resembles the original source code with very little effort, thus allowing everyone to understand, reuse, and copy algorithms or entire portions of code. This may adversely affect many software products that are developed for commercial use and often are released under license. An attacker could study the code, patch it to skip any licensing mechanism, thus enabling free of charge software usage. Another scenario could be a company that develops software to manage its assets, but does not want its competitors to understand the secrets of its success. If a malicious competitor gets access to the binary (CIL) version of the software he or she could reverse engineer the application and understand the secrets behind the company's code. Moreover, by using reverse engineered code the competitors can introduce a compatible but improved version of the application affecting the original company's market.

Managed code is vulnerable to reverse engineering with very little effort as compared to unmanaged code; however, there are few solutions to mitigate this problem.

Some common anti-reverse engineering techniques are:

- Obfuscation – modifies the CLR in such a way as to make it very hard to read and understand, slowing any reverse engineering attempt and preventing inexperienced crackers from understanding the code.
- Third Party Packers – packs an executable into an encrypted Microsoft Windows application preventing certain types of attacks against the code.
- Custom Packer – a customized application packer is an effective solution to make cracking much harder because the attacker has very little knowledge of how to unpack it.
- Keep Secrets Away – this is actually the only good method of protection. Remove all the secrets from code that is executed on the user machine and run them as service on remote machines with access control handled by the company.

Unfortunately, none of these methods offers "bullet proof" security. There is no way to protect software from being hacked if it is executed on a hostile machine, because any experienced attacker with the sufficient amounts of will and time will eventually reverse engineer it.

Moreover the most used technique, obfuscation is based on the *security through obscurity* paradigm which is in contrast with the principle of open design[16] [10].

The next section will review the concept of security through obscurity.

3.1 Principle of open design

Matt Bishop suggests that complexity does not add security in the following quotation:

> *"The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation."* [4] Chapter 13.2.5.

This is one of the main principles that should underlie any good design and implementation with the purpose of ensuring security. An attacker can ferret out with disassembly, analysis or other non-technical means the details of the design and implementation. Bishop highlights this by stating:

> *"If the strength of the program's security depends on the ignorance of the user, a knowledgeable user can defeat that security mechanism"* [4] chapter 13.2.5.

History has proven repeatedly times the strength of this principle. Clear examples are WEP[17] encryption and the DVD Content Scrambling System's (CSS) protection mechanism which were broken easily once they were reverse engineered. A cryptographic system (or in general a security mechanism) must remain secure even if its description is available to an attacker.

[16] *Design principle*: "Specific design principles underline the design and implementation of mechanisms for supporting security policies. These principles build on the ideas of simplicity and restriction." [4] from chapter 13.

[17] *Wired Equivalent Privacy* (WEP) is a deprecated security algorithm for IEEE 802.11 wireless networks.

The application of this principle is complicated by the issues of proprietary software and trade secrets. Companies often do not want their design to be public and available to their competitors who might take advantage of this information.

3.2 Obfuscation

Obfuscation: "the act or an instance of making something obscure, dark, or difficult to understand."

Collins English-dictionary 2003.

As mentioned previously obfuscation is the most common technique to protect managed code. There are several reasons for this. First, it does not require much effort from the developers. The majority of the software obfuscators are available as a plug-in that integrates into Microsoft's Visual Studio. They allow to produce very well obfuscated code. There are several commercial products and many good freeware programs that perform this task. These are summarized in *Table 3*.

Table 3 – Commercial obfuscator software comparison. All product versions taken into analysis are the professional full featured version. *Source: producer websites.*

Name / Feature	Price	Producer	Hacked ?	URLs to List of Feature
DotFuscator	On request (~$2000)	*Preemptive*	Yes	http://preemptive.com/products/dotfuscator/compare-editions
CryptoObfuscator	8 developer license $2399	*Ssware*	Yes	http://www.ssware.com/cryptoobfuscator/features.htm
DeepSea	5 developer $597	*TallApplications BV*	Yes	http://www.deepseaobfuscator.com/features.aspx
Salamander	$1399 for 5-10 developers	*Remotesoft*	Yes	http://www.remotesoft.com/salamander/obfuscator.html
Goliath	Unlimited $350	*Cantelmo software*	Yes	http://www.cantelmosoftware.com/ita/obfuscator.html
Smart Assembly	Unlimited $1195	*Red-Gate*	Yes	http://www.red-gate.com/products/dotnet-development/smartassembly/features/

The column "Hacked?" refers to their current security status. If in the underground community documentation and software tools are available to compromise or totally disable the obfuscation then they are considered as "Hacked". As can be easily seen from this table all these software have been compromised.

Following any of the features links (*Table 3*), it is clear how each of this products has its distinctive features. Each of these vendors enumerates and emphasizes their features with unique, high-flown names. However, most of these use the same techniques to obfuscate the code. Unfortunately few of them are really effective in slowing down a potential attacker. Most of these products are also available as free version with fewer features or as a trial version.

The second reason why obfuscation is very popular is because it is indeed effective in deterring or slow down certain kind of attacks. Even if the hacker community has develop counterstrategies for each of these programs, these counterstrategies are not easy to understand or to perform, thus limiting a successful attack only to highly skilled individuals or groups with good resources in terms of time and money.

3.2.1 Obfuscation strategies

Different obfuscator may use different strategies, but the basics are common to almost every tool; although sometimes different name are used for the same approach.

Symbol renaming

Methods, parameters, classes, fields are renamed to a meaningless sequence of characters to make the code unreadable and to hide names that could reveal part of the code structure.

Method call hiding

Calls to methods and properties are hidden from external assemblies. This makes it hard to determine when and where such a method was used.

Control Flow Obfuscation

Some obfuscators inject false conditional statements and other misleading constructs in order to break and confuse decompilers. Others destroy code patterns that decompilers use to recreate the source code.

String Encryption

Strings are often used to store sensible information, such as password, therefore some obfuscators encrypt the literal in the strings.

Watermarking

This feature introduce in the code some unique tag used to mark the product and identify an owner.

Tamper Detection and Identification

Some obfuscators use techniques to identify if the code has been tampered and notify the owner of the application by e-mail at runtime.

*Anti-*HackingToolName**

Certain obfuscators such as CryptoObfuscator, implement specific features to block common cracking tools, such as decompilers or the Microsoft disassembler[18] ildasm.exe.

An interesting new approach was introduced in the Goliath Obfuscators developed by Cantelmo Software. They call this approach Secure Virtual Machine (SVM). Goliath transforms the Common Intermediate Language (CIL)[19] into a proprietary bytecode which is then executed in the SVM. The new instruction set is then randomized and enciphered with a non-specified Feistel cipher[20].

Although this approach sounds innovative there is a lot of security through obscurity in it. The proprietary bytecode, the SVM, and the non-specified Feistel cipher sound all very fragile. Proprietary solutions have often collapsed at the first sign of difficulties, for example by which standard is the "Secure Virtual Machine" secure? And last but not least, DES is one example of an encryption scheme based on the Feistel cipher and it is well-known to be broken by brute force and linear cryptanalysis attack.

In the addition to the above, obfuscation often is capable of speeding up the execution of the code and reducing the size of the application itself. However, one must be aware that if a feature such as *Control-Flow* is activated, then the additional injected fake code will slow down execution. The following chapter will review reverse engineering and its basics and then will draw some conclusions about the effectiveness of obfuscation as a security mechanism.

[18] A *disassembler* is a computer program that translates machine language into assembly language, the inverse operation to that of an assembler. A *disassembler* differs from a decompiler, which targets a high-level language rather than an assembly language. (*Source*: http://en.wikipedia.org/wiki/Disassembler).

[19] Formerly called Microsoft Intermediate Language or MSIL.

[20] A Feistel cipher is a special class of iterated block ciphers where the ciphertext is calculated from the plaintext by repeated application of the same transformation called a round function. (*Source*: *www.rsa.com/rsalabs/node.asp)*.

4. Reverse Engineering

Reverse engineering (RE) is a concept that could be applied to the approach that humans take to understanding of the universe. Humans study a complex system in order to define its internal design and essential functioning. Chikofsky and Cross describe reverse engineering as: "*Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.*" [12].

The term was born with the analysis of hardware components, where is common to decipher a design from finished products. It is important to note that reverse engineering can be applied to improve someone's products as well as to analyze an adversary in a military or commercial situation.

Over the years, main stream media have caused a distorted mental association (hackers → bad and reverse engineering → stealing software / ideas) which leads people judging these terms negatively, but the RE process is fundamental to understand how things works and we will use it to evaluate security mechanisms such as obfuscation. The relationship between RE and other terms is shown in *Figure 9*.

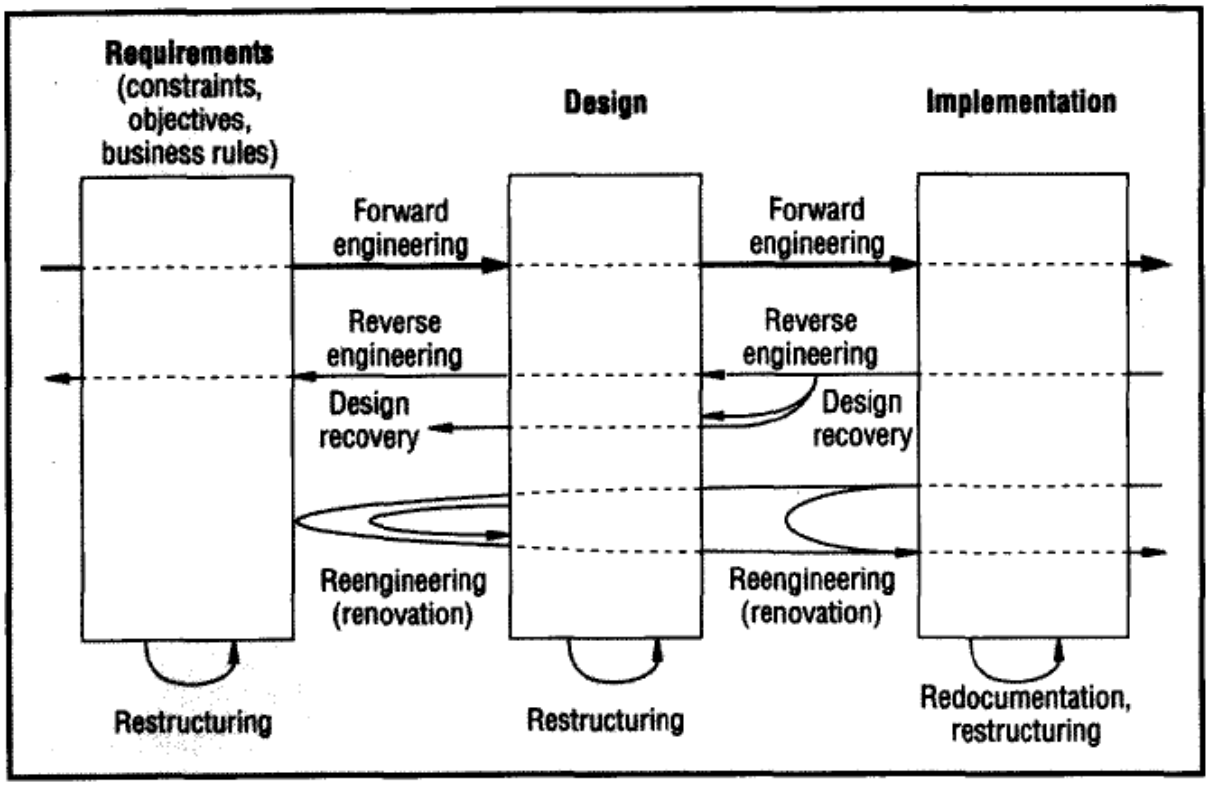


Figure 9 - Reverse engineering and related processes are transformations between or within abstraction levels, represented here in terms of lifecycle phases[21]. Source: [12]

[21] The author would like to thank Elliot Chikofsky for allowing the use of this image in this master thesis project.

4.1 Software Reverse Engineering

Software programs are a good example of a system where RE can be applied as methodology. RE can be used to improve the capability of a program and to enhance it both in its number of features or performance. This process is called re-engineering and finds its definition is "*re-engineering, also known as both renovation and reclamation, is the examination and alteration of a software system to reconstitute it in a new form and the subsequent implementation of the new form.*" [12]. This process is shown in *Figure 10*.

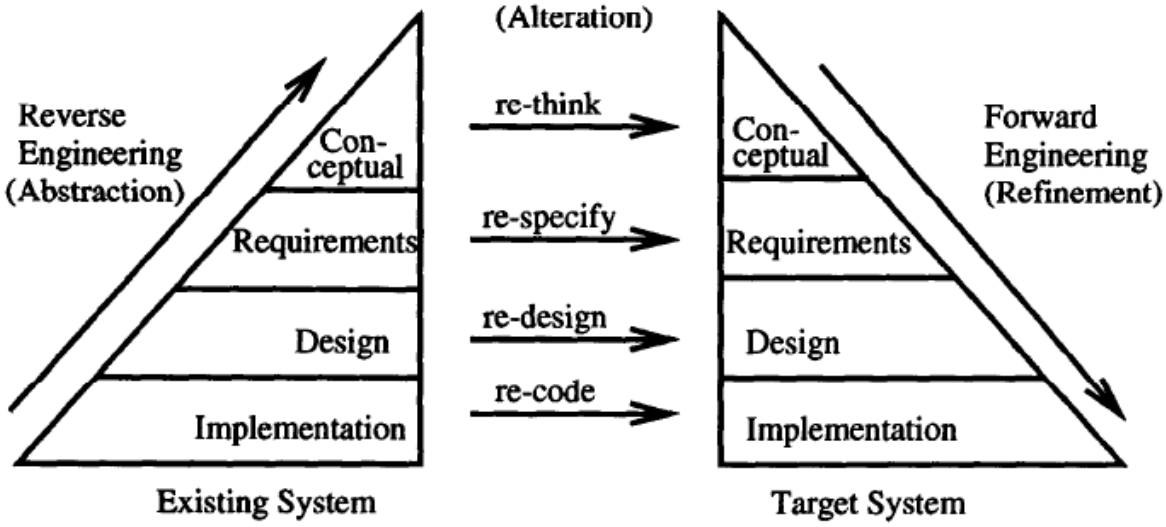


Figure 10 - General Model for Software Re-engineering[22]. Source: [13].

What about obfuscation? Obfuscation is a software technique therefore it can be reverse engineered and even re-engineered. We can illustrates two alternatives with two different scenarios:

- Scenario 1 – Alice is a cracker[23]. She wants to use a new application from Microsoft which is written in C# and its code is obfuscated. Alice can reverse engineer the software, discovers some vulnerabilities in the obfuscation process, and use them to her advantage to understand how the program works. At the end of the process Alice will be able to reproduce an illegal copy of the program or use the software without authorization.
- Scenario 2 – Bob is a developer and works for a company that produces software for obfuscating C# code. Bob is aware that there can be some pitfalls in the software so he

[22] The author would like to thank Elliot Chikofsky for allowing the use of this image in this master thesis project.

[23] The term *cracker* identifies a subcategory of the hackers. The main distinction is that crackers do not comply with any particular ethic; a cracker can damage or compromise a system, steal information, or create a false one with the purpose of money, fame, or fun.

decides to reverse engineer the company's software. He finds some vulnerabilities, understands them, and produces a new design which will be part of the new release of his obfuscation software.

In the first scenario, Alice applied a reverse engineering process. In the second scenario Bob re-engineered his software through the reverse engineering.

To study and evaluate the effectiveness of obfuscation, reverse engineering has been applied to all the popular obfuscation tool available and the results are clear. Techniques such as those listed in section 3.2.1 have pitfalls that can be exploited to different degrees. Often the source code obtained from the RE process was completely replicable and easy to understand (at last in its core parts).

In the internet underground several communities have flourished, each with its own tools to counter obfuscation techniques. *Table 3* a list of some active communities to February 2012.

Table 4 - List of active reverse engineering communities in July 2012.

Group	*Web Site URL*
Black Storm	http://portal.b-at-s.net/
Exetools	http://forum.exetools.com/index.php
ARTeam	http://forums.accessroot.com/
Tuts4you	http://tuts4you.com/
The reverse engineer community	http://www.reverse-engineering.net/

To mention one specific example, the *Control-Flow* technique (illustrated in section 3.2.1) can be exploited with the use of the free, open source decompiler (& debugger) ILSpy[24].

[24] *ILSpy.exe* author's webpage: http://wiki.sharpdevelop.net/ILSpy.ashx

5. SCANIA

SCANIA is one of the largest company in Sweden and one of the industry leaders in heavy trucks, buses, engines, and services. The company has more than 37000 employees operating in about 100 countries. R&D operations are concentrated in Södertälje, Sweden, with around 3000 people at this site. R&D develops the technology behind SCANIA success for their products, in terms of innovation, features, and reliability. The Electronic Control Unit (ECU)[25] Tools and System Test Transmission group (NEVE) is composed of 12 people who develop tools needed to configure and test the embedded electronics of SCANIA's vehicles.

5.1 NEVE Software Development

The NEVE group develops mostly in the .NET environment using the C# programming language. Their tools are used to configure the engine parameters, the gearbox, and the embedded electronics in a truck in order to test new settings or special configurations. These tools have full access to every aspect of a truck's configuration and it is of great importance to SCANIA that these are kept safe from theft, loss, or tampering. These applications also have an important role in SCANIA's warranty process: when a truck visits a workshop, before any maintenance is done the vehicle is checked against any engine tuning that would void the warranty. A malicious user can potentially use these tools to modify the truck to consume less diesel (causing more pollution and potentially violating local regulation on the matter), without losing the warranty because the workshop would be unable to identify the changes.

The ECU tools analyzed during this study are Tool-A, Tool-B, Tool-X and a special database called the Credential Database (CDB). These applications are involved in different phases of the production process of the configuration of the engine for trucks.

Figure 11 illustrates the production process in which these applications are involved. On the right side of the figure, a binary file (generated by applications outside the scope of this thesis project) is given as input to Tool-X. The application will perform a series of checksums which aims to improve the integrity of the final product. The output of Tool-X is given in input to Tool-B, which sets a calibration for the binary to be flashed inside the ECU. If this file will be a production file which is to be shipped in SCANIA's final products, then Tool-B will compute a digital signature[26] over the file to ensure the authenticity and the integrity of this code for the customers.

[25] An engine control unit (ECU), also known as power-train control module (PCM), or engine control module (ECM) is a type of electronic control unit that determines the amount of fuel, ignition timing, and other parameters an internal combustion engine needs to run optimally.

[26] A digital signature is a construct that authenticates both the origin and contents of a message in a manner that is provable to a disinterested third party [4] chapter 10.6.

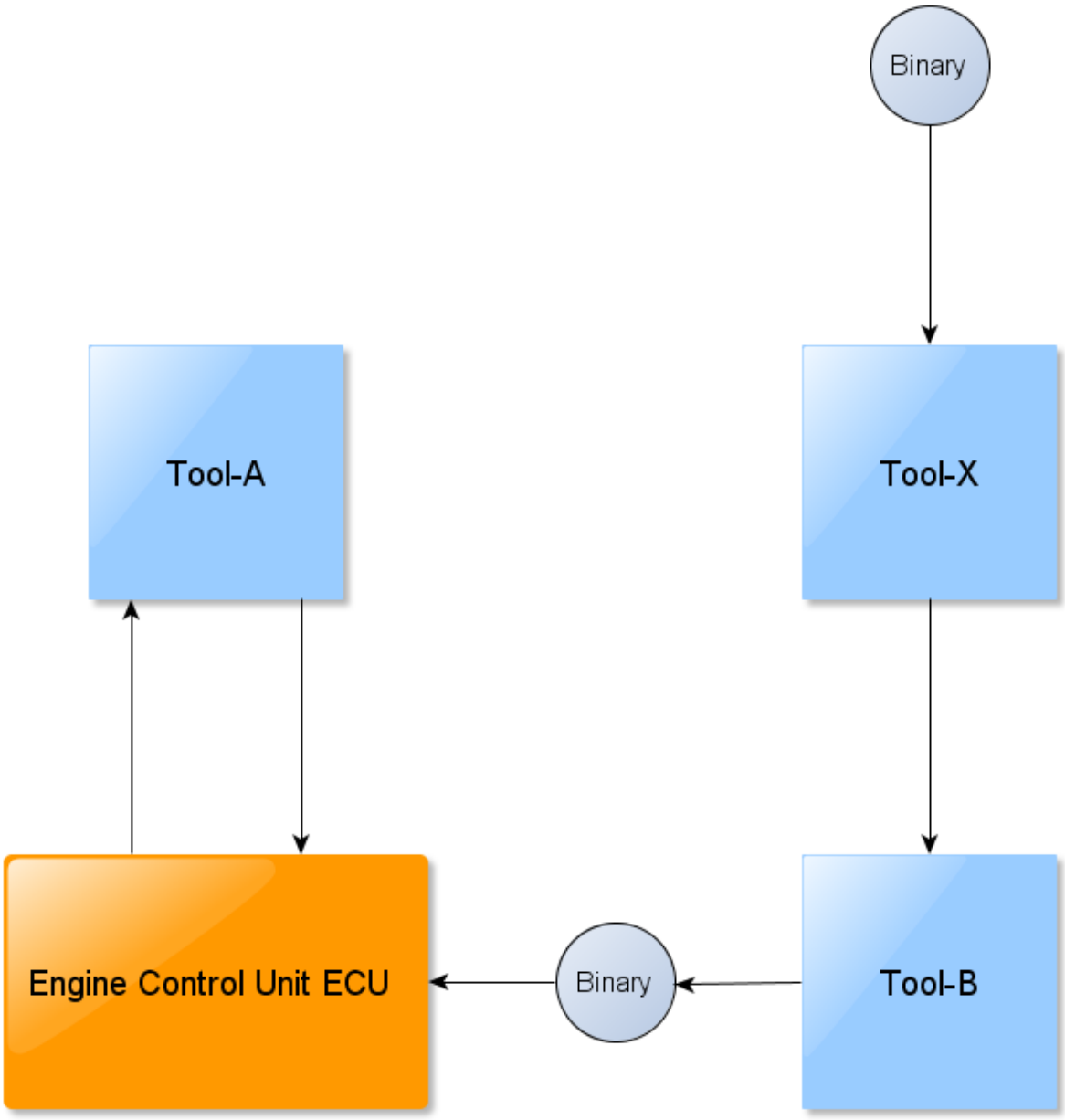


Figure 11 – Production process involving the software developed in the NEVE group.

5.1.1 Tool-A

Tool-A is a C# application developed by group NEVE and the Diagnostic Communication & Software Download (RESC) group. This code is obfuscated with DotFuscator professional version 4.8. NEVE mainly develops the main executable of this application, while RESC develops DLLs needed to communicate with the Electronic Control Unit. Tool-A can modify many parameters of the engine, potentially completely disabling the vehicle. In order for the ECU to accept external connections Tool-A has to identify itself using some credentials. This identification mechanism was developed by the RESC group who is in charge of lower level communication via the CAN[27] bus (this communications is outside the scope of this thesis project).

5.1.2 Tool-X

Tool-X is a C# application developed by NEVE and its code is obfuscated with DotFuscator professional version 4.8. This application is used when processing binary files for the ECU, thus it requires as input a binary file produced by software which is outside the scope of this thesis project. When Tool-X processes this binary file, it performs a series of computation necessary to ensure the integrity of the output, before that the final configuration is flashed inside the ECU.

5.1.3 Tool-B

Tool-B is a C# application developed by NEVE and obfuscated with DotFuscator professional version 4.8. This application is involved in the final step of the production process. The output of Tool-X is passed to Tool-B which sets a calibration for the binary file to be flashed inside the ECU. The binary file produced by Tool-B is ready to be flashed inside the ECU. If this file is a final approved release, then a digital signature is applied to certify the authenticity of the output of Tool-B.

5.1.4 Credential Database

The Credential Database is a special database (DB) which contains a collection of credentials. This database is used mainly by Tool-X and it is at the core of Tool-X's security therefore it must be secured to prevent unauthorized access.

[27] CAN bus is a message-based protocol, designed specifically for automotive applications, but now also used in other areas such as industrial automation and medical equipment.

6. Security Planning Through Threat Analysis

This chapter describes an analysis of the current security mechanisms and solutions implemented in the development tools at NEVE. For each tool potential threats will be described from a theoretical perspective while in chapter 7 an empirical approach will be taken to confirm the theoretical findings.

Some threats are common to multiple tools while others are tool-specific. Existing security measures are built upon previous decisions involving the work of different groups working within the company. Therefore substantial modifications cannot be proposed because they would affect a large number of developers and a well-known work-flow of using the current tools.

It is very important to emphasize that the theoretical analysis was conducted in *black-box* mode. Black-box testing means that the tester does not have access to the application source code, therefore all the knowledge is based upon understanding how the application has been engineered by examining the engineered artifact (for example examining the binary code, pairs of input and output, the timing relationship between input and output, *etc*.). The only knowledge provided beforehand was the applications' stack and operating environment.

6.1 Security Management

Information security management at SCANIA does not provide security guidelines for .NET software developers. The consequences are that each development group implements its own (often proprietary) solution. This lack of communication and planning hides potential vulnerabilities and increases the difficulty of introducing any new security implementation. Every little modification to the system requires a series of minor adjustments in all the other subsystems, resulting in substantial efforts. The results are that the development phase is slow because everything must be fine-tuned to co-exist with an environment subject to different rules and policies. Moreover, the step from development to production is negatively affected in terms of time. Introducing a common security language, able to unify security procedures in the departments is a long term highly suggested task. This will introduce standard security procedures common for all employees that will sensibly raise the security bar. The costs of new implementations would be reduced both in time and money. Moreover, such an environment would be more "user friendly" since all employees would follow one methodology (i.e. one authentication method for all the tools).

To resolve this issue is outside the scope of this report, but suggestions are presented in the analysis of the results in chapter 10.

6.2 Tool-A Analysis

As described in section 5.1.1 Tool-A is a major tool developed in .NET C#. *Figure 12* illustrates the logic stack of how it works. On top, Tool-A is the main executable. This and the *ECUcom.dll* are developed by the NEVE group, while *scomm.dll* and the rest of the stack were developed by different teams at SCANIA.

In order to execute Tool-A the user must have a dongle USB key. There are two main dongle keys, a *black* and a *green* one. The *green* key is considered outdated and its security is known to be broken by malicious workshops. The *black* key is considered secure. Although the *green* dongle is no longer delivered it has not been revocated and it still works with Tool-A.

A Google search revealed a Russian workshop selling a tuning kit consisting of software and hardware connector. Included in this kit is a *black* dongle key. With a little social engineering the following e-mail was received to the specific request to the workshop:

Request:

"I want to modify the diesel consumption parameters of my trucks engine"

Reply:

"*Hello,*

VCI2 will work on all Scania bus and trucks from 2004 and newer. If you have older trucks than 2004 you need VCI1. The SOPS file editor is not necessary if you want to make diagnostic and programing(sic). It is for editing special parameters." ... "Delivery by TNT 3-5 days - 100 EUR with insurance."

The author of this report was unable to verify if the *black* key provided in this kit is the new dongle key which is still considered by SCANIA to be secure. However this should be verified by ordering one of these kits and analyze the key because it could be a *green* key with a different color box. Further investigation is highly suggested.

The dongle is only one part of the security in Tool-A. The other security mechanisms are implemented via software. As all the .NET tools at NEVE, Tool-A is obfuscated with DotFuscator version 4.8. DotFuscator is a very popular obfuscator. The current release is version 5.0.1. This defines the current obfuscation as outdated; moreover being DotFuscator a popular tool, there is a great deal of understanding of its operations. To defeat obfuscation is just a matter of knowledge therefore DotFuscator is not a good candidate for this operation. The cost of this tool is the highest among all competitors just because of its popularity, however the benefits are very limited (see *Table 3*).

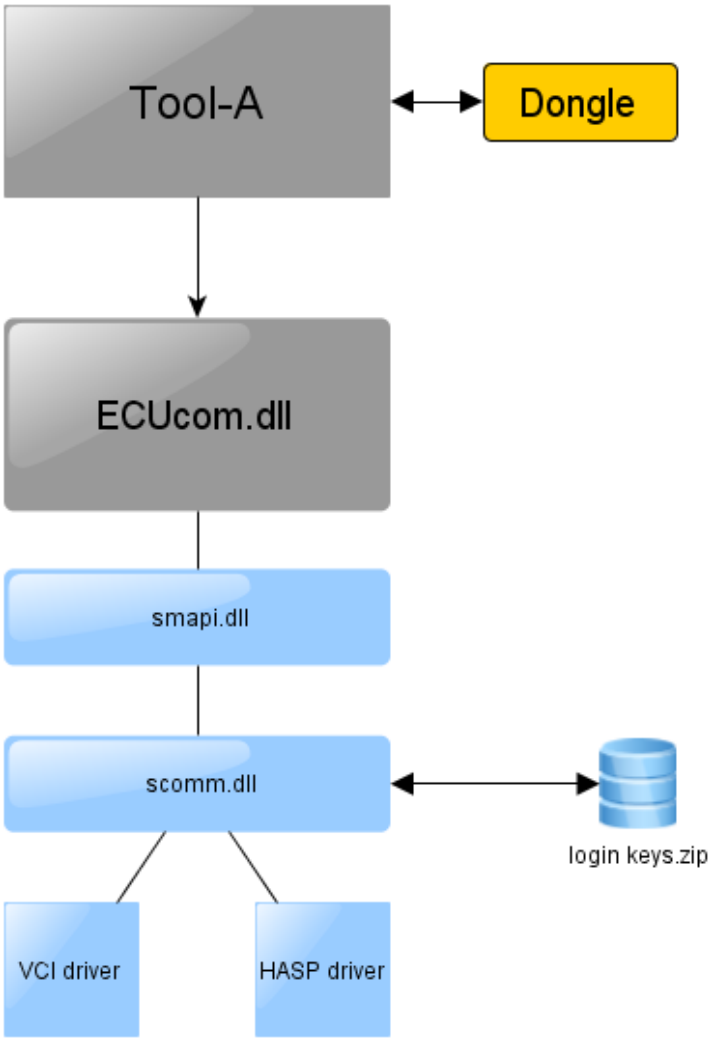


Figure 12 - Tool-A stack

When Tool-A is executed it performs a date check against the hardware clock provided by the dongle. The expiration date is hardcoded inside Tool-A. If the date reported by the dongle is beyond the expiration date of this version of Tool-A, then, Tool-A will not allow any connection to the ECU. In the lower part of the stack, further authentication keys are required by *scomm*. These keys are stored in a dedicated database. As mentioned in section 6.1 the lack of standards makes impossible to fully understand the authentication features of this scheme without being part of the responsible develop team. Tool-A is potentially vulnerable to Man In The Middle (MITM) attack[28]. Although the communication between the ECU and Tool-A is encrypted there is no documentation on how keys are exchanged or regarding what encryption scheme is used. Regardless of the encryption, eavesdropping the communication may lead to security threat such as a reply attack or a cryptographic key leak. An attack on this communication link has not been made as it was judged to be outside the scope of this thesis project.

[28] The man-in-the-middle attack (MITM), in cryptography and computer security is a form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker.

This section will conclude with a list of the major vulnerabilities of this tool. The first column lists the threat, while the second column reports its estimated severity level. The severity levels are defined as follow:

Minor severity: Vulnerability requires significant resources to exploit.

Moderate severity: Vulnerability requires significant resources to exploit, with significant potential for loss. Or, vulnerability requires little resources to exploit, moderate potential for loss.

High severity: Vulnerability requires few resources to exploit, with significant potential for loss.

Table 5 - Tool-A Security Assessment

Threat	Severity level
Managed Code Reverse Engineering	High Severity
Green Dongle Key Security	High Severity
Black Dongle Key Security	Minor Severity
Hard Coded Expire Date	Minor Severity
Man In The Middle	Minor Severity

6.3 Tool-X Analysis

Tool-X is a C# application which is used by everyone who needs to compile binary configuration files as a part of the compilation process of the ECU. It is executed as part of the build process of the binary file and the main purpose of this tool is to guarantee the authenticity and the integrity of the output file.

During its workflow (illustrated in *Figure 13)* Tool-X performs a series of checksums and employs a proprietary procedure to verify the integrity of the binary file.

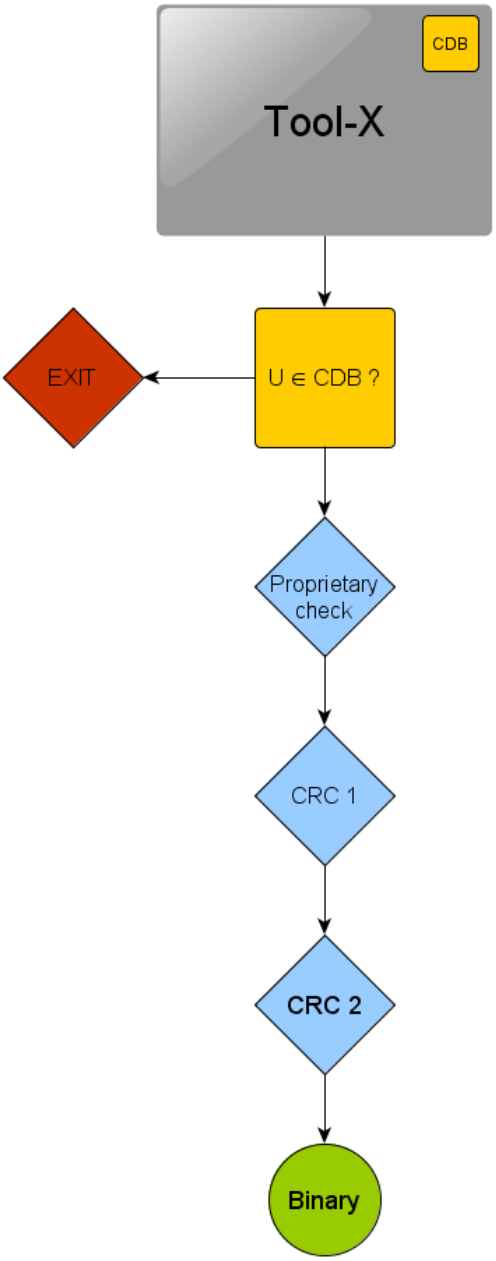


Figure 13 - Tool-X workflow

The operations that Tool-X performs are specified in an Extensible Markup Language[29] (XML) configuration file.

A crucial decision in the application is the authentication of the user or the machine the software is running. For this authentication procedure Tool-X relies on a file called CredentialDB. This file contains a list of credentials, of those entities who are authorized to execute the application. Weather the application runs on a production server or on a local user workstation, the Tool-X will check the CDB for the appropriate value before saving its output binary file.

[29] Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

The CDB is encrypted with a proprietary symmetric key algorithm. The application is obfuscated with DotFuscator 4.8. Therefore the only protection from reading the encryption key is provided by the obfuscation. This solution exposes the tool to a direct reverse engineering attack on the key as illustrated in section 7.2.

Another important aspect of Tool-X's functionality is that there is no control over the input. If a malicious user manages to access a working copy of the application he can input any configuration file and generate a false file. This situation can generate spoofed[30] output, i.e., an output that looks like the original but is not. Although extremely hard to perform without access to the other development tools, this attack may be performed by someone inside the development chain to sabotage the production line to benefit SCANIA's competitors, for mere vandalism or revenge against the company.

This section concludes with a list the major vulnerabilities of this tool. The first column lists the threat, while the second column reports its estimated severity level. The severity levels are defined as follow:

Threat	Severity Level
Credential Database - (CIA)	High Severity
Managed Code Reverse Engineering	High Severity
Spoofing	Moderate Severity

6.4 Tool-B Analysis

Tool-B is a C# .NET application that does not implement any specific security mechanism. It does not make use of a dongle as Tool-A does. As with the other tools, obfuscation of the code is performed with DotFuscator 4.8. This software does not contain an expiration date or other software controlled feature to prevent its execution.

This section concludes with a list the major vulnerabilities of this tool. The first column lists the threat, while the second column reports its estimated severity level. The severity levels are defined as follow:

Threat	Severity Level
Managed Code Reverse Engineering	High Severity

[30] In the context of network security, a spoofing attack is a situation in which one person or program successfully masquerades as another by falsifying data and thereby gaining an illegitimate advantage. In copyright context an illegal copy can be pushed as original producing benefit to a malicious entity of some sort.

6.5 Credential Database Analysis

The CDB is a database with a collection of credentials. This file is encrypted with a proprietary symmetric key encryption algorithm. To verify the security of this unknown encryption scheme cryptanalysis and reverse engineering technique must be used. These two approaches will create the necessary knowledge to evaluate if the encryption scheme is mathematically secure or not.

6.5.1 Credential Database Threat Table

To understand how the encryption is performed further knowledge must be acquired about the internal functioning of the credential database and its usage. This additional knowledge will provide a basis to evaluate the strength of the encryption of the CDB.

Chapter 7 describes the reverse engineering attempts that were made and will show the results of this process. This empirical testing will allow the evaluation of the current security mechanism.

7. The "Dark Arts"

The "*Dark Art*" is the Internet's name for *software reverse engineering*. Underground Internet communities often develop their own jargon and when relating to them is good to know how to blend in. This name provides a good alternative description of RE. Software RE relies on the user's experience and there is no manual to assist the user in rapidly gaining this experience. Common strategies and approaches exist, but almost every attack is unique and requires intuition, patience, and educated guesses. Therefore, the term "dark" indicates something that is not well defined and it is completely obscure to the general public.

The theoretical security analysis of SCANIA's software gives a broad prospective and illustrates several strategies to exploit software vulnerabilities. However, to achieve what the theory illustrates it is not an easy task. Therefore some attempts to make a successful attack defeating current security mechanism have been performed.

This chapter will describe two reverse engineering attacks performed during this master thesis project to provide empirical data to support the theoretical analysis and to better illustrate the subject. As noted earlier, these reverse engineering attempts were made by testing in black-box mode.

7.1 The Cracking Tools

As mentioned earlier in this thesis, there are tools to defeat the most common obfuscation techniques. We will take advantage of this existing knowledge using the CIL decompiler *ILSpy* and *DeDot* an anti-obfuscation tool which has been recently upgraded to support the techniqes used by many commercial obfuscators (including DotFuscator). These tools will be used for the first attack described in section 7.2.1. For the second attack illustrated in section 7.2.2 we will use the knowledge gained in the first attack to make a complex low level attack. Additional tools used in this second phase were *PEBrowse PRO* a static-analysis tool and disassembler for Win32/Win64 executable for Microsoft .NET and *CFF explorer* a suite of utilities including a Portable Executable[31] (PE) editor and a process viewer. We also used *ILDASM,* the CIL disassembler included in Visual Studio.

[31] The *Portable Executable* (PE) format is a file format for executables, object code and DLLs, used in 32-bit and 64-bit versions of Windows operating systems.

7.2 Attacking Tool-X

Tool-X is used to guarantee the integrity of binary files for the ECU. The security policy say that to execute Tool-X and output a configuration file the user must have valid credentials.

This attack will illustrate two different approaches: first, we will exploit the characteristics of managed code, trying to understand how the program works. The second strategy will illustrate a low level attack that will modify the application to void its protection routines.

7.2.1 Attack 1: Cracking the Credential Database

The aim is to understand what is inside the CDB and if there is way to "cheat" Tool-X so that it allows execution by a malicious user who is not in the CDB. We know that Tool-X's code is obfuscated with DotFuscator. DotFuscator implements different strategies, but since it is not known how it was configured the process begins by inspecting the CIL code. To inspect the CIL we will use ILSpy. Most of the modern obfuscators implements mechanism to prevent direct code inspection with ILSpy. This is not the case for Tool-X but a simple example is given to introduce the reader to the method.

7.2.1.1 Defeating anti-ILSpy protection

Most modern obfuscators implement a series of mechanism to prevent the use of debuggers or decompilers. ILSpy is a powerful IL disassambler which is commonly used by reverse engineers. To prevent the use of this tool obfuscator such as as GOLIATH ads invalid instructions at the beginning of the PE's procedure which will cause ILSpy to crash. The code snippet below will be used as example to illustrate this particular technique. Each line of the code represents a CIL instruction. To understand what each instruction does there are several online resources available.

The GOLIATH obfuscator implements this technique by introducing the invalid instruction *FE22*.

```
IL_0000: /* 38 |  */ br IL_0007
IL_0005: /* FE22 | trash instruction */ unused
IL_0007: /* 06 | */ ldloc.0
```

To counter this mechanism one can simply replace all invalid instruction such as *FE22* with the no-operation (NOP) instruction *0x00*.

7.2.1.2 Inspecting the code with ILSpy

Inspecting the code may require a long time depending on the application's complexity. An attacker must identify which methods are involved in the security of the application, then study and understand them. For Tool-X the search aimed to find cryptographic procedures or operation involving the CDB.

Under *ILSPy* inspection the code shows that *code-injection* and *string encryption* are used. *Methods renaming* has also been applied, but DotFuscator use a weak renaming strategy therefore with some experience it is easy to understand what the code is doing.

The inspection reveals that the software is using a Microsoft's Windows's parameter to identify the user who is logged in the machine. Our hypothesis is that, if this parameter is listed in the CDB, then this user is allowed to manipulate the binary file with Tool-X. There are different strategies to attack this security protection. In this attempt we inject an invalid username inside the CDB. *Figure 14* shows part of the security code.

Figure 14 - The code reveals a call to get a Windows's parameter. (Note that the figure has been intentionally obscured at the request of the student's employer.)

To inject a new user inside the CDB we need first to decrypt it. So the next step is to look for some cryptographic keys and a decryption function that will reveal more about the algorithm being used. DotFuscator implements strings encryption therefore the key probably will look like a string of random symbols. After locating the decryption function we realized that a fundamental part is *char* manipulation. Tracing back the char array used in the function we are able to identify three encryption/decryption keys stored in char arrays. Since DotFuscator encrypts *string* types, but not *char arrays,* keys are shown in clear! This example shows how programming style can affect the security of the software. *Figure 15* shows one of the secret keys in clear text.

Figure 15 - Encryption/Decryption keys are shown in clear text (Note that the figure has been intentionally obscured at the request of the student's employer.)

7.2.1.3 Writing the CDB crack

The final stage of the attack is to decrypt the CDB using the keys we retrieved. Using C# we develop the decryption function as a separate program. This can be achieved by studying the code and understanding what operation are performed. The core operation is shown in *Figure 16*. With some experience it is possible to distinguish the fake injected code of DotFuscator from the real code.

Figure 16 - Part of the decryption function (Note that the figure has been intentionally obscured at the request of the student's employer.)

7.2.1.4 Retrieving the RSA public key

During the code analysis a call to the *RSA CryptoServiceProvider()* of .NET was spotted. With the knowledge gained so far we were also able to identify a string type value which might hide a hardcoded key. As shown in *Figure 17* the RSA key is probably used to decrypt the binaries, but it is stored in an encrypted string, therefore not usable directly. To resolve this problem we need first to defeat DotFuscator string encryption. To do so we used *DeDot,* a tool that will extract DotFuscator keys from the PE and automatically decrypt all *string* type value. The results are shown in *Figure 18* where the public RSA key has been compromised.

Figure 17 - RSA encrypted public key (Note that the figure has been intentionally obscured at the request of the student's employer.)

Figure 18 - The Decrypted RSA public key (Note that the figure has been intentionally obscured at the request of the student's employer.)

7.2.1.5 The results

Once the cracking program is ready it was executed against the CDB file. *Figure 19* shows the encrypted CDB, while *Figure 20* shows the same file decrypted. The last step is to append a line with a new credential, then save and encrypt the database. Now malicious users can use Tool-X as an authorized user. In addition, during this attack we have been able to retrieve the RSA public key which enables an attacker to **verify** binary files to be processed by Tool-X.

Additionally, now that we know where the key is stored it is easy to identify the code that uses this key, hence we can path the code to skip the verification step - as a result the modified Tool-X can process files that do not have a valid signature!

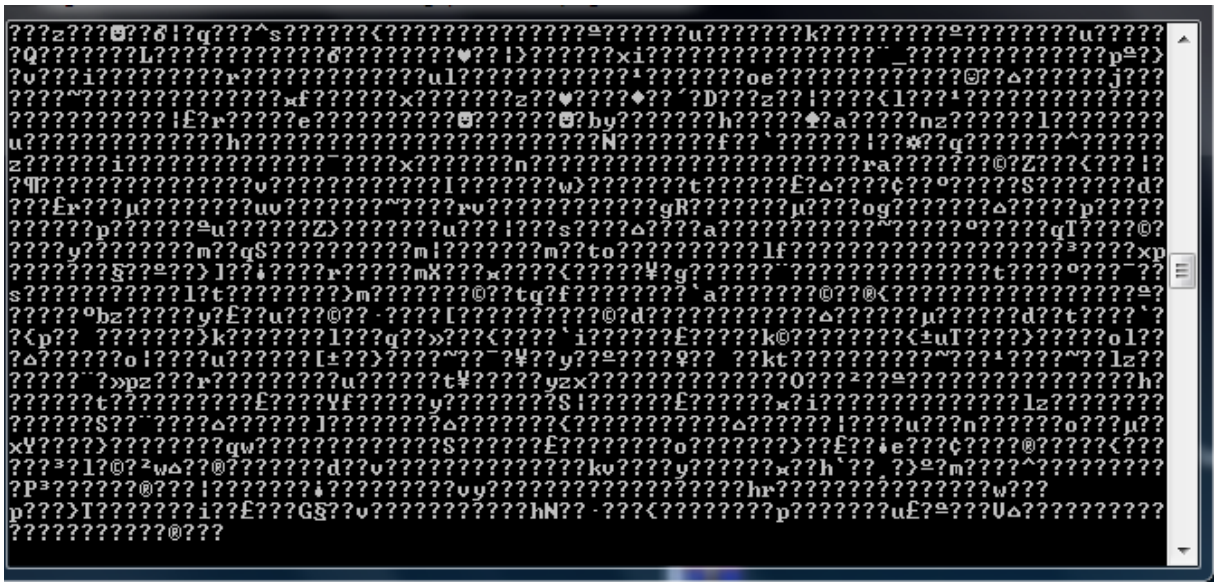


Figure 19 – Part of the encrypted CDB

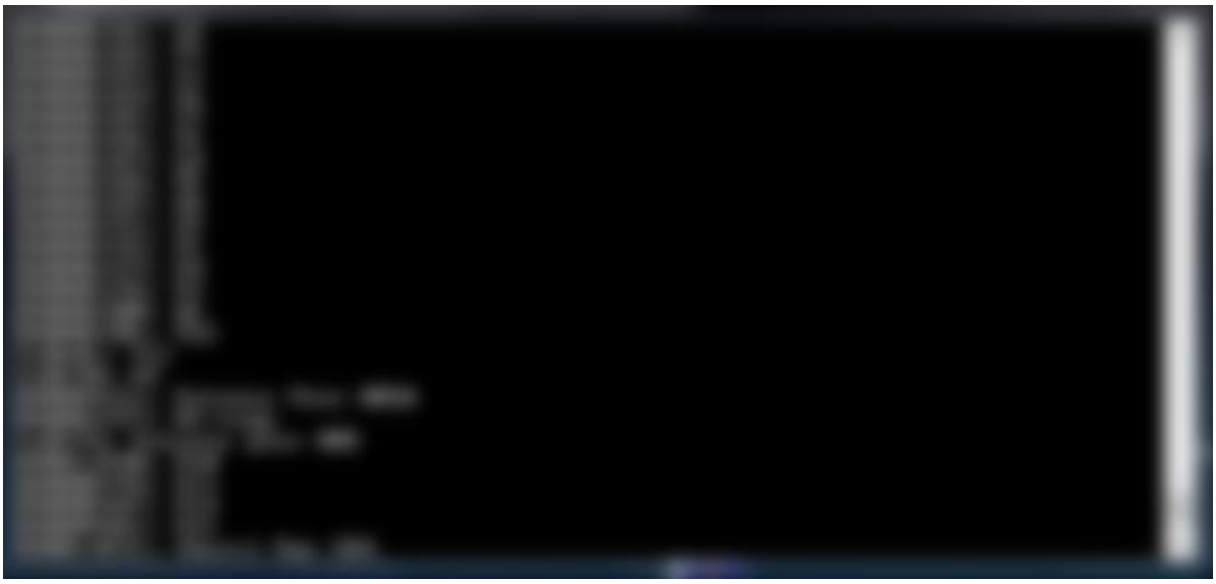


Figure 20 – Part of the decrypted CDB

7.2.2 Attack 2: Patching the Tool-X executable

This attack is much more powerful that the attack described in section 7.2.1, but is more complex and requires a longer time to perform. It is not possible to illustrate the full procedure in this thesis. This type of attack can defeat any kind of software protection by voiding or skipping instructions involved in software security.

7.2.2.1 The attack logic

Using the knowledge gained during *attack 1* we know that Tool-X invokes a function that validates the Windows's user. If this function returns a *false* value the user is invalid, if the function returns a *true* value then the user is authorized to manipulate the binary file. In this attack we will simply change the return value of the function to always *true* regardless of any other condition. The advantage of this is that we do not have to know the key used to encrypt the credential database. Another advantage is that we do not have to insert the name of the malicious user into the credential database - avoiding leaving a trail of evidence of who is misusing this tool.

7.2.2.2 Where to begin?

Usually security functions are executed at the very beginning of an application. This is to ensure that the user is valid before any other action is performed. Based upon this reasoning we expect that the assembly code that we will need to modify is probably at the beginning of our application.

Using ILDASM.exe (the .NET disassembler that comes with Visual Studio), we identified the function that validates the user in Tool-X. In the options of the IL disassembler there is the possibility to switch to *byte* view. As shown in *Figure 21* the second column shows the byte while the last column shows the CIL instruction.

```
IL_004f:  /* 28   | (06)00007C       */ call

IL_0054:  /* 17   |                  */ ldc.i4.1
IL_0055:  /* 2A   |                  */ ret
IL_0056:  /* 18   |                  */ ldc.i4.2
IL_0057:  /* 1B   |                  */ ldc.i4.5
IL_0058:  /* 8D   | (01)00000D       */ newarr
IL_005d:  /* 0B   |                  */ stloc.1
IL_005e:  /* 07   |                  */ ldloc.1
IL_005f:  /* 16   |                  */ ldc.i4.0
IL_0060:  /* 72   | (70)001AF3       */ ldstr
IL_0065:  /* A2   |                  */ stelem.ref
IL_0066:  /* 07   |                  */ ldloc.1
IL_0067:  /* 17   |                  */ ldc.i4.1
IL_0068:  /* 28   | (0A)00005E       */ call
IL_006d:  /* A2   |                  */ stelem.ref
```

Figure 21 - ILDASM byte view stack.

7.2.2.3 How to identify what to change

ILDASM shows a direct representation of the internal bytecode. By changing this bytecode we will make changes to modify the application's behavior. Microsoft's .NET is essentially a stack machine; this means that utilizes a stack rather than using registers. To move a value from a certain location X to Y, the value X is pushed to the stack and then is popped from the stack into Y[32]. Below an example of how C# source code is translated into CIL:

The C# code *user.isValid = false;* is translated into three stack-related lines:

ldarg.0
ldc.i4.0
stfld bool Tool-X.frmMain::isValid

Using the reference manual [27] we can understand each of these instructions:

ldarg.0	load argument 0 into the stack,
ldc.i4.0	push 0 into the stack as four byte integer (I4),
stfld	replace the value of *field* of the object with a value.

This is equivalent to saying, *(arg0).isValid = 0*;

To initialize it at true we need to change this into:

(arg0).isValid = 1;

This means *changing the second instruction to ldc.i4.*1. Using the reference manual or ILDASM we can translate this code into its byte representation which in this example would be the byte sequence *0x02167D.*

7.2.2.4 Patching the executable

Using *PEbrowse PRO* we disassemble the executable end explore it looking for the byte string *0x02167D*. It is important that we use the longest bytecode search pattern to have higher chances to identify the correct portion of the code. This byte squence might not be unique so it is possible to find more than one occurrence of it. The byte code for the instruction *ldc.i4.1* is *0x17* so once we identify the right portion of the code we can substitute the instruction with the HEX editor included inside *PEbrowse PRO.* A lot of support for finding instruction and modifying the code is directly provided by the tool, as it is able to show the code during execution so that we can track where the application is jumping to, highlighting all the interesting instructions. This is extremely helpful during this phase. *Figure* 22 shows how this tool can reveal useful information.

[32] A stack is a last in, first out (LIFO) abstract data type and linear data structure. A stack can have any abstract data type as an element, but is characterized by two fundamental operations: *push* and *pop*. The push operation adds a new item to the top of the stack, or initializes the stack if it is empty. Pop either reveals previously concealed items, or results in an empty stack

Figure 22 - PEbrowse is a powerful tool for reverse engineering of .NET application

7.2.2.5 Results and final considerations

Once the application has been modified we can simply save it and execute it again to verify the correct execution of the modification that we have made. This attack has been greatly simplified in this thesis. The complete version of this procedure includes entry-point discovery using the tool *CFF Explorer*. Moreover the byte patching affects the program and other operation such as recalculating the application's offset must be performed. This brief introduction to practical reverse engineering techniques, should give the reader an idea of how powerful this method can be with the correct tools.

8. Developing a Security Solution at SCANIA

Chapters 0 and 7 have described from a theoretical and an empirical perspective the vulnerabilities that existed within the software tools that were the target of this study. During this master thesis, several solutions have been suggested to improve the security of these tools. This chapter will review only the solution accepted by SCANIA while discarded solutions will be described in Appendix II – `Discarded Solutions`. The solution that has been adopted has been refined and tailored to meet SCANIA's needs over the course of weekly meetings and regular feedbacks from the NEVE development group.

First we will introduce the solutions developed for Tool-A and Tool-B which operates in the same environment. Tool-X and the CDB required a different solution that will be described in section 8.6.

The solutions presented in this chapter have been developed with the aim to minimize cost and complexity, with a focus on usability. The reasons for adopting this metric are:

Minimize complexity

To minimize complexity not only increase usability, buts simple solutions are easy to understand and less error prone. Moreover, lower complexity often leads to lower expenses. The maintenance of complex systems requires specialists and dedicated hardware which adds cost in both time and money.

Minimize cost

Often several different solutions offer the same functionality but with different prices. Minimizing cost requires finding the right balance between what is needed and what is superfluous. Obtaining the maximum result at the minimum cost is desirable.

Minimum impact on the final user in order to improve usability (user friendly solution)

The main requirement for this project was to provide a user friendly solution. This means that the added security should not require extra work from the everyday user.

8.1 A New Scheme for Tool-A and Tool-B

Tool-A and Tool-B operate in similar environments therefore a common solution could be developed for both the applications. This solution is composed of four main components (or phases) each one deployed against a specific threat. This schema is presented in *Figure 23.*

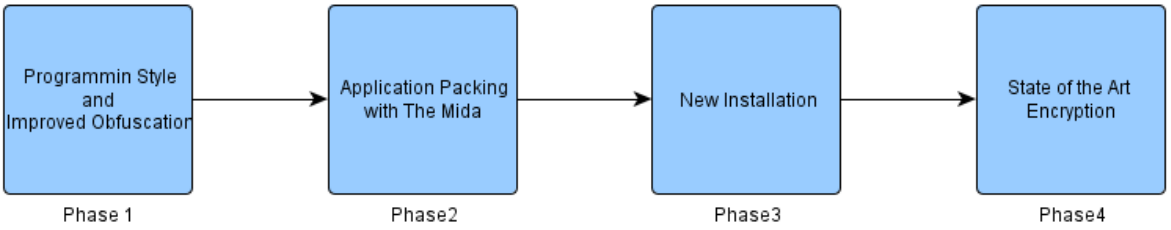


Figure 23 - Solution Schema

Each of these phases introduces new security concepts to the IT operations of the NEVE group. Given the complexity of the SCANIA environment some solutions could not be fully implemented, thus leaving space for future improvements.

8.1.1 Phase one

When developing code there are programming strategies that can improve the security of the software. Moreover, when developing managed code it is good practice to adopt new programming style that will improve the effectiveness of obfuscation. It is important that security is kept in mind from the beginning of the project's development to avoid writing multiple times the same code in order to fix issues that might come up over time.

8.1.2 Phase two

Software security is well known issue that unfortunately there is not yet a definitive solution. To effectively address the most common issue software developers often employ commercial security software. The market is overwhelmed by different licensing software with each one promising to keep crackers away and to protect the intellectual propriety. Although no commercial solution offers a truly secure solution, these solutions are very helpful in deterring inexperienced and intermediate crackers. Section 8.4.1 will present the commercial software packager "The Mida" which is the core of Phase 2.

8.1.3 Phase three

Because of the restrictions in the environment in which the tools operate (no internet connection, untrusted users, *etc.*) the installation process has been heavily modified. The installation procedure is the only time when NEVE can identify the users wanting to install the software because the user has to connect to the company's intranet. For this reason a new

installation schema has been implemented which will allow NEVE to uniquely identify the user and to introduce software limitations, such as enforcing an expiration date.

8.1.4 Phase four

The new installation schema will introduce the possibility to add a new feature: encryption. Using AES256 the tools will be stored encrypted inside the engineers' hard drives. Moreover, instance of each tool will be bound to the identity of a specific SCANIA engineer.

This will introduce two distinct features:

1. Each engineer will be able to use only their own copy of each tool, preventing unauthorized access. This is achieved by using AES256 encryption together with unique cryptographic keys for each user.
2. Entrust the engineers creating a "pact of trust". If one of the tools leaks to the public SCANIA may track the source of the leak given the unique binding between the engineer and the tool.

8.2 Improved Obfuscation

During this master thesis's project different obfuscators have been tested. To perform this comparison the same application has been obfuscated with different obfuscator software and then inspected with ILSpy. The obfuscators tested were Smart Assembly, GOLIATH, DotFuscator, .NET Reactor, and Eazfuscator.NET (listed in *Table 3*).

The obfuscated applications were inspected with ILSpy and subject to a disassembling analysis. The obfuscators that performed well in this test were Smart Assembly and .NET reactor. The code was completely obfuscated and it was very hard to find correspondences even knowing the original source code. These obfuscators also mangled the *char* arrays so that secret keys stored in this format were not displayed as clear text. The obfuscator that performed worst was DotFuscator; not only the code was easy to read, but the secret keys stored in *char* arrays were in clear text.

The second test performed involved a literature research for known technique to void obfuscator protection. Since defeating obfuscation is just a matter of knowledge, the more knowledge about an obfuscator the weaker it is considered. Under this condition .NET Reactor and Smart Assembly were the worst. Their great popularity lead the underground community to develop automated tools that are able to retrieve the original code (or something very similar to it) from the obfuscated PE. In the particular case of Smart Assembly, the cracking tool *{SA}* did an impressive job generating in output large portions of the original C# code. It is a great wonder that Red Gate software, the producer of Smart Assembly, is still in the market.

GOLIATH and Eazfuscator are two minor products. GOLIATH was developed by a very small company and has limited pool of users. Therefore there is very little knowledge of its internal

functioning. Some features such its anti-ILSpy function, are easily defeated (as described in section 7.2.1), but others are not. Eazfuscator has the enormous advantage of being free. Moreover, Eazfuscator offers a very strong string encryption which prevents dynamic decryption.

8.3 Programming Style

This section suggests some programming advice that can add additional difficulties to a potential cracker. These methods are presented to illustrate how a little extra effort while programming can make reverse engineering job much harder. For all of the solutions presented there exist known countermeasure that an experienced RE can use.

8.3.1 Breakpoints

Breakpoints allow breaking the execution of an application at any point to study its internal functioning and to spot potential weaknesses. Breakpoints are also fundamental when debugging an application during its development process.

There are three types of breakpoints which are generally used by reverse engineer: hardware, memory, and INT 3h. These are essential because they allow to perform live analysis of an application.

The most common type of breakpoints is the INT 3h represented by the opcode CC (0x00) or byte sequence 0xCD 0x03. The following code shows how to remove this breakpoint but it is important to note that this approach can generate false positive.

```
/*********************************/
/*** detect INT 3h breakpoint ***/
/*********************************/
bool CheckForCCBreakpointXor55(void* pMemory,  size_t SizeToCheck)
            {
            unsigned char *pTmp = (unsigned char*)pMemory;
            unsigned char tmpchar = 0;

            for (size_t i = 0; i < SizeToCheck; i++)
            {
            tmpchar = pTmp[i];
            if( 0x99 == (tmpchar ^ 0x55) ) // 0xCC xor 0x55 = 0x99
             return true;
            }

    return false;
 }
```

Intel implemented in their processor architecture a hardware breakpoint controlled by the use of special registers: Dr0 - Dr7. One method to detect such breakpoints is to call the *GetThreadContex()* and *SetThreadContext()* of Win32 as shown in this example:

```
// CheckHardwareBreakpoints returns the number of hardware
// breakpoints detected and on failure it returns -1.
int CheckHardwareBreakpoints()
{
    unsigned int NumBps = 0;

    // This structure is key to the function and is the
    // medium for detection and removal
    CONTEXT ctx;
    ZeroMemory(&ctx, sizeof(CONTEXT));

    // The CONTEXT structure is an in/out parameter therefore we have
    // to set the flags so Get/SetThreadContext knows what to set or get.
    ctx.ContextFlags = CONTEXT_DEBUG_REGISTERS;

    // Get a handle to our thread
    HANDLE hThread = GetCurrentThread();

    // Get the registers
    if(GetThreadContext(hThread, &ctx) == 0)
        return -1;

    // Now we can check for hardware breakpoints, its not
    // necessary to check Dr6 and Dr7.
    if(ctx.Dr0 != 0)
        ++NumBps;
    if(ctx.Dr1 != 0)
           ++NumBps;
    if(ctx.Dr2 != 0)
           ++NumBps;
    if(ctx.Dr3 != 0)
        ++NumBps;

    return NumBps;
}
```

8.3.2 Removing Portable Executable Header

This is an anti-dumping technique that removes a PE header from memory at runtime. If the application is dumped it would be missing important information such as the Relative Virtual Address (RVA) of tables, the entry point, and other details that are needed by Microsoft's Windows. This technique may compromise Microsoft's Windows API which tries to access the resource legitimately.

```
// This function will erase the current images
// PE header from memory preventing a successful image
// if dumped
inline void ErasePEHeaderFromMemory()
{
    DWORD OldProtect = 0;

    // Get base address of module
    char *pBaseAddr = (char*)GetModuleHandle(NULL);

    // Change memory protection
    VirtualProtect(pBaseAddr, 4096, // Assume x86 page size
            PAGE_READWRITE, &OldProtect);

    // Erase the header
    ZeroMemory(pBaseAddr, 4096);
}
```

8.3.3 Size of Image

This method increases the size of the *SizeOfImage* field in the IMAGE_OPTION_HEADER of a PE file at runtime. This block reverse engineering tool that were not developed to handle this issue. An example is given below.

```
// Any unreasonably large value will work say for example 0x100000 or
100,000h
void ChangeSizeOfImage(DWORD NewSize)
{
    __asm
    {
        mov eax, fs:[0x30] // PEB
        mov eax, [eax + 0x0c] // PEB_LDR_DATA
        mov eax, [eax + 0x0c] // InOrderModuleList
        mov dword ptr [eax + 0x20], NewSize // SizeOfImage
    }
}
```

8.4 Application packing

To better protect the tools developed by NEVE, top-notch commercial anti-cracking technology has been evaluated. Most of the commercial tools on the market protect the application by packing the application's files in a proprietary format.

8.4.1 The Mida

A notorious packager in the reverse engineering community is "The Mida®", developed by Oreans Technologies. The Mida is basically a virtual machine. It translates the original application into a proprietary byte code and then executes it inside cryptographically protected layers of virtual machines. Among the many features that this tool offers are anti-debugging

features, anti-memory dumpers, and code permutations. This solution is innovative in the panorama of commercial solutions. Moreover, if compared to other commercial solutions The Mida is offered at a very low costs (for example, Themida x32 with a company license costs 299€ and Themida x32/x64 with a company license costs 399€ according to http://www.oreans.com/Themidax32.php and http://www.oreans.com/Themidax32x64.php respectively).

This packager works best with unmanaged code because allows to insert special macros that can virtualized the code and allows the metamorphic engine to scramble the assembly code instructions. Since NEVE's application are written in C# these macro cannot be used but there a workaround. Sensitive code can be deployed in an unmanaged DLL, then protected with the various macros that The Mida implements. Later the application can be protected by embedding the DLL with XBundler which is a plugin for Themida. Xbundler allows compressing DLLs and packing them to one executable file (A copy license for XBundler costs 159€ according to http://www.oreans.com/XBundlerPlugin.php).

It should be noted that there is no guarantee that software protected by The Mida cannot be reversed. A highly sophisticated attack such as the one executed against Skype [28] can perform twin process debugging with hardware breakpoint and allow an insight of the application (it has to be noted that such attack took years according to the authors).

Although not in the scope of this master thesis project, some simple tests have been conducted to verify some of the properties of this anti-cracking software. When executing debuggers such as IDA[33], OllyDebugger, or Visual Studio's Debugger, The Mida protects the software by crashing the process immediately preventing the attacker from attaching the debugger to the running process.

The second test performed was to dump the memory of the machine running The Mida. A first analysis does not reveal any sensitive value stored in the memory because the stack of cryptographically protected virtual machine adds complexity that is not easily understood.

However, the use of this anti-cracking technology has some drawbacks. First and foremost, the use of virtual machines and virtualized code can slow down the performance of the application. A second issue is that anti-virus software may report false positive since computer viri use similar techniques to hide their malicious code.

8.4.2 The Mida Known Issues

When protecting an application with The Mida all the assemblies are removed from the PE header and they are decrypted at runtime when necessary. This causes a known problem with reflection/serialization. The Microsoft's *csc.exe* runtime compiler is used to compile assemblies at runtime. As the assemblies are not visible from the PE header the csc.exe will fail to compile the code.

[33] IDA http://www.hex-rays.com/products/ida/index.shtml and OllyDebugger http://www.ollydbg.de/.

To counter this issue, the following solutions are suggested:

1. Write the reflection code in a new DLL. Protect the executable file and call the DLL when needed.
2. Visual Studio includes a "Generate serialization assembly" option that pre-creates a serialization DLL instead of doing it reflectively at runtime. This DLL can be included with XBundler plugin into the executable, thus solving the problem.

Another issue which can occur when protecting an application with The Mida is that antivirus software will detect it as a potential virus. Antivirus software uses heuristics to identify a potential virus based upon certain features. Since this software cannot understand the operations performed by software protected with The Mida they often consider the application as polymorphic. This characteristic is typical of a computer virus, therefore the antivirus programs trigger an alert message.

To resolve this issue it is sufficient to add the target application to the exception list of the antivirus program in use. The exception list is a list of files that the antivirus will not scan for virus.

8.5 A New Security Schema

One of the major limitations for this thesis project is that these R&D tools must operate without an internet connection. In fact these applications are often used in the field where no Wi-Fi or cellular connectivity is available. As a result of this lack of connectivity, several solution considered, ranging from the use of external hardware such as a dongle key or a smart phone to authenticate the user but since usability is the main goal for NEVE a different approach has been taken. The idea is to exploit the only certain instant when the users are connected to the intranet. This connection happens only during the installation phase when the engineers must be connected to SCANIA's network in order to download the software to their computer.

The newly implemented schema will allow the following features:

1. User Authentication,
2. Application leak tracking,
3. Unique encryption key for every application.

The schema is presented in *Figure 24* This solution was the best schema of all of those evaluated because it is very efficient in regard of user wait-time. In less than three seconds after requesting the tool the engineer will receive in his mailbox the batch file needed to install the requested software. Section 8.5.1 defines how the user is authenticated, while section 8.5.2 and section 8.5.3 will explain how the user tracking is achieved and how the new features introduced by using encryption.

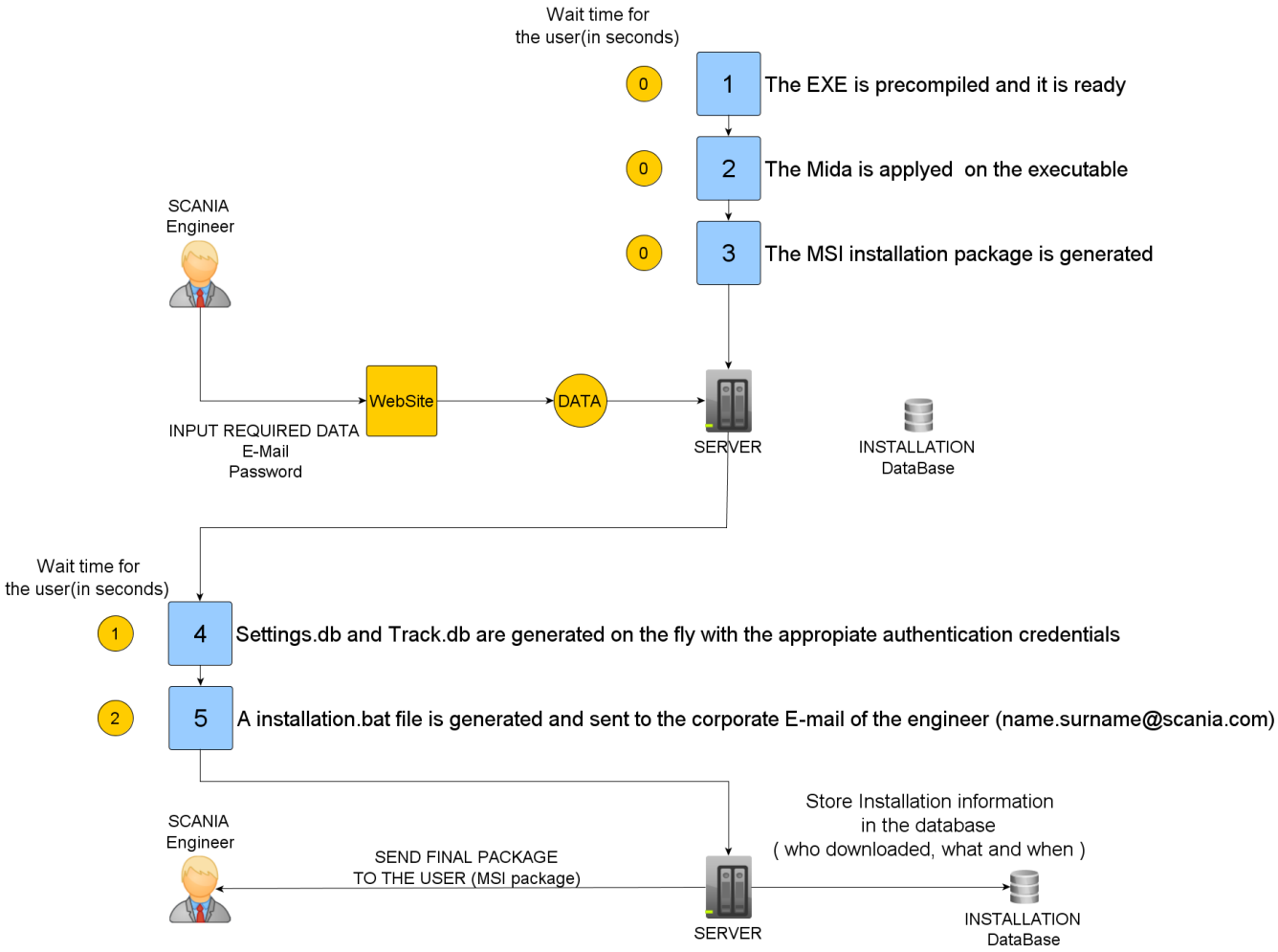


Figure 24 - A new installation schema

8.5.1 User Authentication

As mentioned in previous section, the installation process is the only time when the SCANIA's engineers **must** be connected to a network. This is because the research and development tools are stored on servers accessible only through SCANIA's intranet. The new installation schema in contrast with the previous procedure, requires the user to be authenticated before they can retrieving the software.

To achieve authentication without installing any additional identity management software the decision was to utilize the engineers' corporate e-mail work account. At SCANIA each engineer has a personal e-mail account in the form: name.surname@scania.com. Since access to this mailbox is limited to a specific engineer the idea is to send all the information needed to install the software via the employees' e-mail.

This first phase consists of the following steps:

1. The user open his web browser and connects to the new software repository page,
2. A newly designed C# web application will request the user's e-mail and a personal password to be used for this instance of this application,
3. The user inputs the required information,
4. The user receives an e-mail with detailed installation instruction to his or her corporate mailbox. The message contains a batch script run the software installation.

The newly developed web application will perform a series of operations completely transparent to the user in order to prepare the installation files in a random network file system path that will be deleted as soon as the installation is completed. To further restrict access to each application an "allow list" of e-mail can be configured; in this case if the employee's e-mail is not listed he or she will be denied access to the specific tool. The web application is written in C# and runs under version 4.0 of the .NET framework. Note that issues of securing the e-mail system, the file system of the servers, and the delivery of the e-mail are outside the scope of this thesis project.

8.5.2 Encryption And Parameterization

As already mentioned this new schema allows the introduction new restrictions settings on the application usage and utilizes encryption.

Based on the password selected by the user, utilizing the PBKDF2 standard, a cryptographic key is derived and is used to encrypt the development tools while they are stored inside the engineers' hard-drive. This method will prevent software leak from occurring when a computer (laptop) is lost or stolen. The encryption scheme used is AES256 as discussed in section 2.3 robust and secure (as of July 2012). It has to be noted that any password can be used by the engineer to protect the application and not only the password he or she uses to access the corporate e-mail account.

Along with the application a new file is generated. This file named "*settings.db*" is encrypted with AES256 and contains a configurable list of settings that can be used to control the application's behavior. This file is mandatory and the application will not execute nor be decrypted if this file is missing or modified by malicious users.

The *settings.db* file is generated at installation time and contains a list of parameters. Examples of parameters are given in the following paragraphs, but parameters can be add or removed if necessary.

8.5.2.1 Expiration Date

Since hard-coding an expiration date into the program's source code can be vulnerable to reverse engineering attacks, the expiration date has been moved to a cryptographically

protected file which cannot be decrypted without the correct key. The application will not be decrypted nor executed if the expiration date is past. Depending on the tool, different date checks can be performed. Software such as Tool-A which makes use of a dongle key, can check the date against the hardware clock of the USB key. For software that runs without this technology the date can be checked against the computer clock. The computer's clock is not as secure as the hardware clock provided by the dongle and can be manipulated by a malicious user.

8.5.2.2 Usage Timer

To further improve the time restriction a time-counter can be added. At every execution the application stores cumulative *execution time* in *settings.db.* When the *execution time* equals the *maximum execution time,* then the software will not execute.

8.5.2.3 Hash

When the application launcher reads the *settings.db* it will compute its own hash using the SHA512 scheme. If the computed hash is different from the hash stored in the *settings.db* the application launcher is considered corrupted and the process will terminate. The use of SHA512 does not affect the performances of the applications (the user does not notice few microseconds difference) because the computation is performed by high performance hardware.

8.5.2.4 Maximum Number of Executions

A further customization allows limiting the maximum number of executions of the program. At runtime a counter is updated, once it reaches the maximum value the application launcher will not run the software.

The *settings.db* allows us to store a variety of settings cryptographically protected by the AES256 scheme. Moreover this file is never stored in decrypted form on the hard-drive. The decrypted data is only temporally stored in memory and deleted as soon as the information is no longer needed. The memory where the file is decrypted is encrypted by The Mida's virtual machine giving additional security. When packed inside The Mida, the *settings.db* is protected by anti-memory dumping and anti-memory patching by use of a cyclic redundancy check (CRC).

8.5.2.5 Tracking Capabilities

Among the files that are copied to the user's computer at installation time there is a file named *tracking.db*. This must be present to execute the application correctly. If this file is missing the

application launcher described in section 8.5.3 will terminate. Inside *tracking.db* is stored the information about the identity of the user who downloaded the program. During the installation the web application encrypts this file with AES256 with a secret key known only by NEVE and kept secret from the users.

When a tool is stolen or lost there are two possible scenarios:

1. The software is retrieved (i.e. by investigation or by chance). In this scenario SCANIA can decrypt the "*tracking.db"* and identify the source of the leak.
2. The second scenario occurs when the software is not recovered. If a malicious user is in possession of the program, the chances are that his computer operates with an internet connection active. In this case the application launcher's code will detect the presence of an internet connection. If a connection is found then the *tracking.db* can be sent back to SCANIA, with all the information concerning the hostile machine such as its Internet Protocol (IP) address, Media Access Control address (MAC) address, operating system information etc.

These information can be useful to identify where the program is located and can uniquely identify the engineer who leaked the program. The Swedish and EU laws and regulations concerning data privacy will need to be considered, but this is considered to be outside the scope of this thesis project.

8.5.2.6 The Network Code

The network code is executed as part of the application launcher routine. It is executed as an independent thread and silently tries to reach SCANIA's server. If the connection is available, then the *tracking.db* file and a complete report of the host machine are uploaded. This report includes the contents of the tracking.db file, the whole list of IP interfaces, operating system information, number of CPUs, and several other parameters. Parameter can easily be added or removed from the report with few lines of code in the launcher. The application launcher is built in such a way that any exception occurring during this phase is ignored so that the user is never informed about this background operation. An expert malicious user can prevent the launcher to connect to the SCANIA's server by adding a rule in its firewall. SCANIA should consider the privacy implication related to this feature, which are outside the scope of this project.

8.5.2.7 The Report Parser

The report parser is a C# application developed to monitor the incoming messages to SCANIA's server. The parser is dedicated to processing the leaks reports. As soon as a new report is uploaded the parser decrypts the report log, and generates an e-mail for the administrator. The e-mail contains detailed information about the report and lifts the engineers from manually decrypting and checking the presence for new reports. This tool has been developed solely for purpose of simplifying the processing of these reports. Moreover, very few (if any) reports are expected to be uploaded over time. The time delay from when a report is generated in a hostile machine to when the administrator is informed about the leak is only few seconds (5 or less depending on network performances).

8.5.3 The Application Launcher

For applications that are executed manually by operators and not as part of a scripted procedure (such as Tool-X) an application launcher has been developed. This multithreaded software is developed in C# .NET and its task is to ensure that all the restrictions concerning the security are respected. If the requirements are met the application launcher will decrypt the program (e.g. Tool-A) and execute it.

As illustrated in *Figure 25* the launcher behaves has a security envelope for all this type of applications.

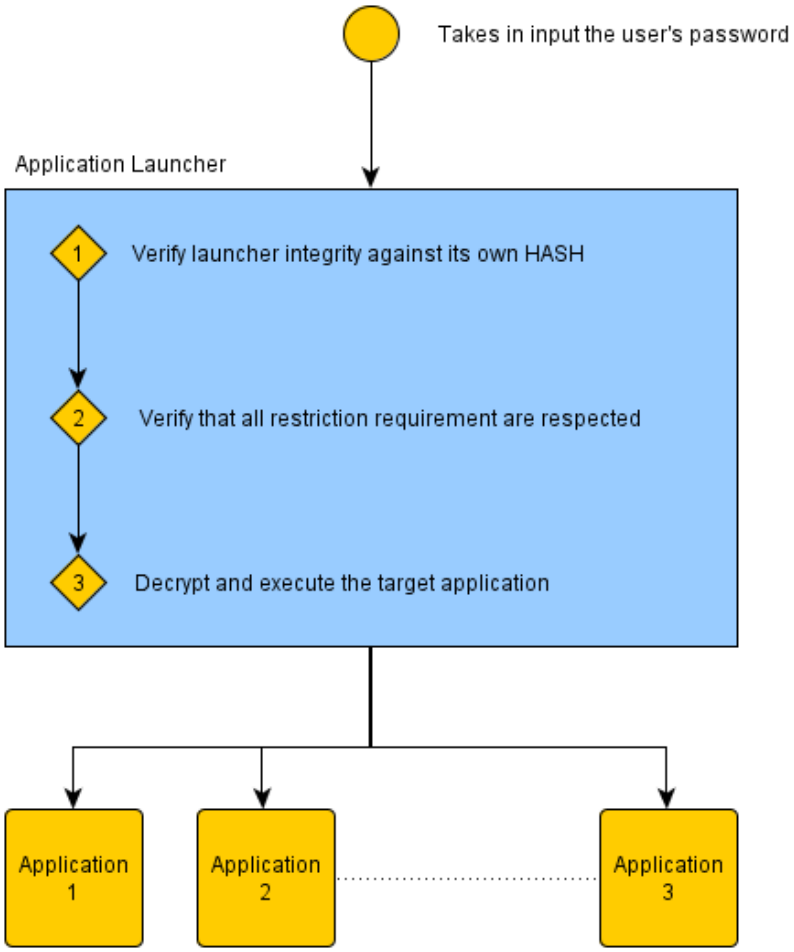


Figure 25 - The application launcher workflow

When the user runs the launcher he or she is prompted with a password request. If the password is the same as the one submitted during the installation procedure, then the correct cryptographic key will be generated and the development tool can be decrypted. This mechanism make no use of hard coded keys therefore even if the application launcher is subject to a reverse engineering it will not leak information to the attacker. All cryptographic keys are generated following the Randomness requirements for security in section 2.7.

If all pre-conditions are met, then the launcher will decrypt the *settings.db* file in memory. The file is parsed and all the parameters are checked. The parameters inside the *settings.db* can be configured with a great deal of flexibility. If all the conditions described by the settings inside this file are met, then the launcher will proceed.

As described in section 8.5.2.6 the application launcher includes a network code to report the tool usage. This code is executed by a separate thread which silently tries to connect to SCANIA's server and uploads a complete report of the machine where it is being executed. In parallel with this the launcher will decode and launch the application.

8.6 Adapting the New Schema to Tool-X and the CDB

Tool-X is executed as part of a routine process but can also be run by an engineer. This software is automatically executed on the "build server" when the configuration binary file for the engine is compiled and ready for production vehicles.

This server environment does not allow the use of an application launcher based solution that requires input of a password by a user. Additionallly, this software cannot use any external hardware such as USB dongle key. This forces the developers to store the cryptographic key in areas that are directly accessible from the application. Therefore Tool-X required a different strategy to improve its security.

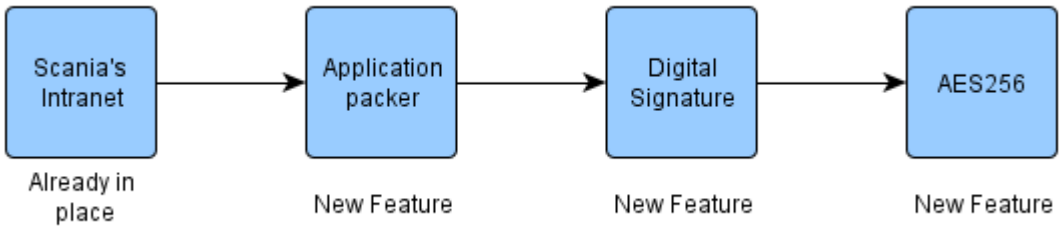


Figure 26 - Tool-X and CDB solution schema

8.6.1 Tool-X's New Features

Tool-X is mostly executed on remote virtual server inside SCANIA's intranet (but it can also be executed on local users' machines). This environment is not easy to reach therefore, it was decided to discard a solution with external hardware technology where to safely store a cryptographic key (such as a dongle USB drive). The solution scheme that has been adopted introduces three new features to increase Tool-X's security:

1. AES256 encryption for the CDB,
2. RSA technology to digitally sign documents and,
3. The anti-cracking tool The Mida.

8.6.2 New Encryption for the Credential Database

The first improvement introduced was to replace the broken encryption scheme utilized for the CredentialDB with AES256. This will utilize state of the art encryption to protect the database when stored on the hard-drive.

If all the security restrictions are met, then the CDB is loaded into memory and decrypted. To facilitate the developer in the operation of manipulating the database a new C# application to encrypt and decrypt files was developed. The cryptographic key can be dynamically controlled by seeding the program with different passwords. The keys are generated using PBKDF2 standard. While in memory the CDB is protected by the anti-cracking tool The Mida. The solution ensures that memory is virtualized, encrypted, and anti-memory dumping features are active.

8.6.3 Digital Signature for Tool-X's Configuration Files

Although Tool-X operates inside the secure and trusted environment of the SCANIA's intranet and in the developer's machines, additional precautions have been put in place. As illustrated in *Figure 11* Tool-X takes as input a binary file. The operations to be performed on the binary are specified in a XML configuration file. To prevent unauthorized users from modifying the XML file and performing malicious operations (such as manipulating unauthorized binary files) RSA technology has been implemented.

With Visual Studio, three C# applications have been developed:

GenerateRSAkeys – Generates new RSA cryptographic key pairs for each new release of Tool-X. The keys are output as XML files. This format allows the engineers to easily share and integrate these keys for different application or solutions.

SignXML – NEVE developers can use this program to sign XML files. In this particular scenario this tool will be used to sign the configuration files of Tool-X

VerifyXML – Is used to validate XML's digital signature

When Tool-X is executed it will verify the origin and the integrity of its input with the provided public key. If the digital signature is valid then the binary file is processed.

8.7 Onion Structure

Both the solutions consist of a layered structure. Each layer protects the application from different threats. Like in an onion, to reach the core of the vegetable all of the layers must be peeled off. The same applies to these schemes, to reach the source code of the application several security mechanisms have to be bypassed. This concept is illustrated in *Figure 27* where the native application is in yellow, the obfuscation is represented by the green color, and the

packaging with The Mida in blue. The orange box represents the application launcher which performs the security checks defined in the *settings.db* file before decrypting any application.

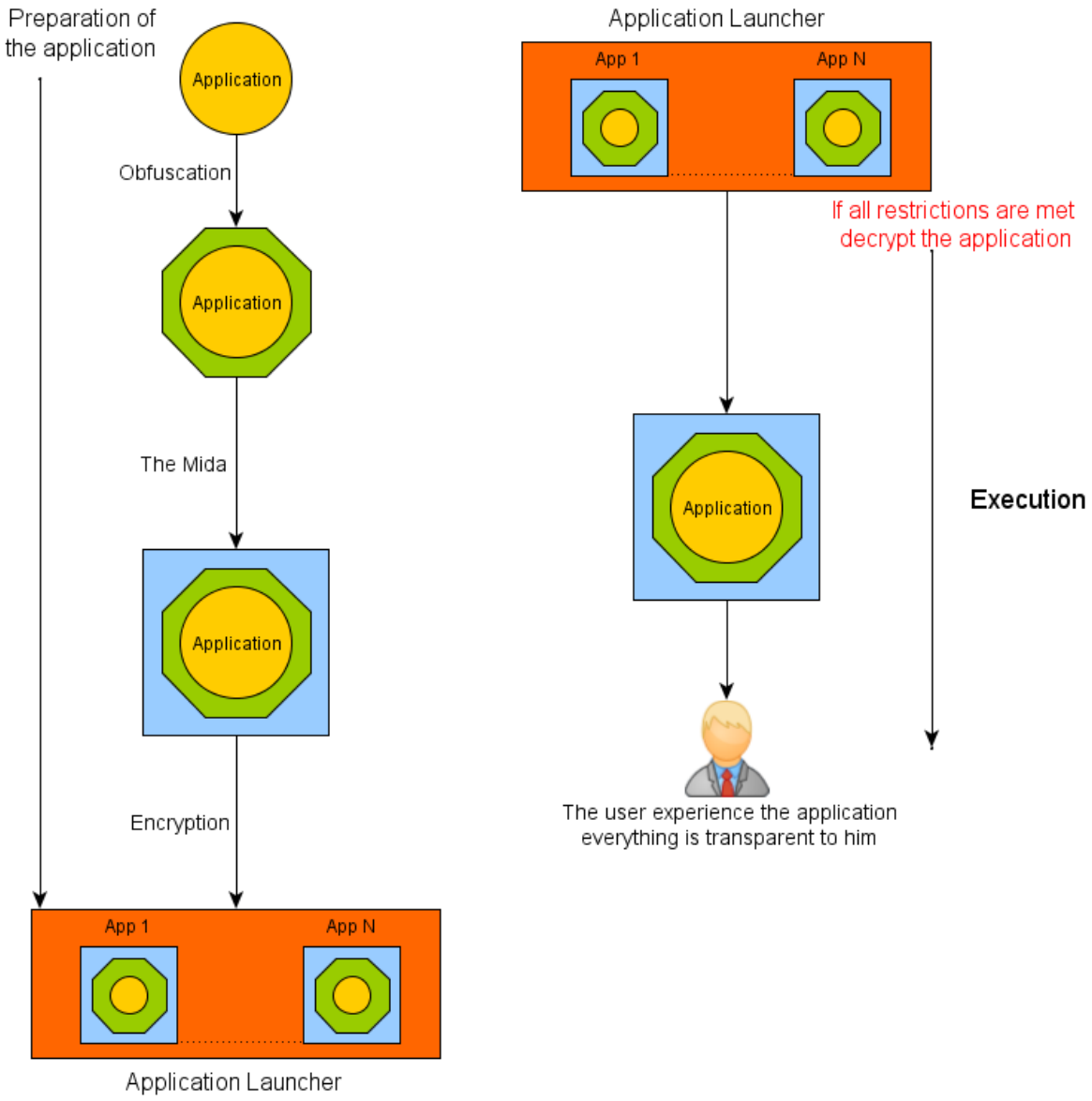


Figure 27 - The onion like structure of the application

The solution for the Tool-X application has a different structure. There is no application launcher, but there is a digital signature verification of the XML configuration file. *Figure 28* illustrates the workflow for Tool-X. Notice that Tool-X is still protected by the same layers as the other applications.

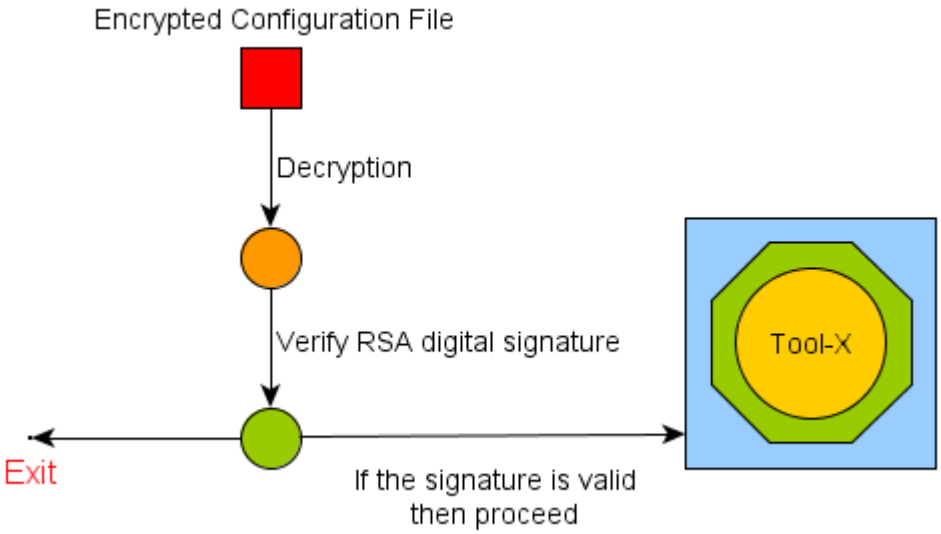


Figure 28 - Tool-X security workflow

9. Results Description

This chapter will introduce our results and categorizes them while Chapter 10 will analyze these results in detail. This study produced both technical and non-technical results due to the holistic nature of information security. To facilitate the analysis of these results this chapter will structure the results and describe the reasons behind these categorizations.

Information security is strictly a matter of knowledge (as for any subject dealing with information). The knowledge required to secure a system must not be limited to an individuals (i.e. the Chief Information Security Officer of the company) but must be shared among all the individual operating within the selected environment. This means that to secure a system the information about how to achieve this must be accessible and redistributable at different level s to the different individuals responsible for the different parts of the system.

A clarification is needed at this point. Within a certain environment it is not required that every individual is a security expert, but rather that every person receives and understands the minimum necessary knowledge to enable them as employees to perform their daily task without negatively affecting the system.

Therefore IT engineers must have the same understanding of the information security problem of non-technical personal. What differentiates them is that the IT engineers apply their knowledge to implement and maintain security mechanisms based on valid security policies.

This distinction allows categorizing the results of this study into technical results and managerial results.

Table 6 - Results and their categories

Managerial Results	Inter-department sharing of knowledge	Provide basic best security practices knowledge to all the employees	Ensure that information security requirements are met
Technical Results	Identification of security threats	Empirical test of theoretical results	Development of a .NET security scheme

9.1 Description of Managerial Results

Although not in the scope of this study some managerial results were produced. Security needs a holistic view therefore technical and non-technical aspects must be considered.

Inter-department sharing of knowledge

The lack of a centralized security management for the development of the R&D tools impairs the sharing of good solutions. If a team develops a good security solution the other team would be unaware of this solution.

Provide basic best security practices knowledge to all the employees

Although SCANIA has some generic security guidelines to which the employee is introduced, the employees of different department should have basic security concepts related to their specific work. Moreover, this knowledge should be review and updated with the emerging security threats, such as phishing or USB-drive virus.

Ensure that information security requirements are met

Even though security policies are in place they must be enforced. Personal dedicated to verifying that security policies are respected is needed because security policies are ineffective without an enforcement mechanism.

9.2 Description of Technical Results

This study has shown how .NET software can be easily exploited and code obfuscation alone does not guarantee the protection of any intellectual property encoded inside an application. The three key technical results are described in the following there subsections.

9.2.1 Identification of security threats

An extensive literature study has shown that several security schemes exists that can act as solid layer for software security. Unfortunately in many cases, due to environmental restrictions excellent security schemes such as TOTP cannot be implemented. These restrictions collide with Ashby's Law of Requisite Variety. Uncertainty is reduced through information when the control mechanism exhibits the same amount of variety as the system to be controlled exhibits [29].

"*Variety kills variety*", Law of Requisite Variety (Ashby 1963) [29].

The theoretical uncertainty introduced in this environment translates into poor authentication methods, weak integrity, and easy loss of confidentiality.

Table 7 summarizes the major threats that have been identified. This table links each threat to a specific environment. In this table ".NET Security" refers to the security that comes with using .NET, "Engineering" refers to actions of the engineers in SCANIA R&D groups, and "Engineering and Management" refers to both SCANIA management and the engineers in the various groups.

Table 7 - Threat table for .NET and systems related to the framework

Threat	Environment
Managed Code Reverse Engineering	.NET Security
Obfuscation	.NET Security
Misuse of Hardware Security Token	Engineering and Management
Hard Coded Secrets	Engineering
Man In The Middle	Engineering
Spoofing	Engineering
Proprietary solutions for known problems	Engineering

9.2.2 Empirical test of theoretical results

Chapter 7 tested the .NET vulnerabilities identified in chapter 6. The results of this reverse engineering attack confirmed the theoretical weakness of obfuscation. Obfuscation is **wrongly** regarded as the most effective security practice to protect software intellectual propriety for .NET applications.

9.2.3 Development of a .NET security scheme

This master thesis project produced a new security scheme for .NET applications at SCANIA's NEVE group while meeting the restriction of section 1.5. This scheme consists of different layers of security which aims to increase the complexity of the applications. The added complexity provides extra security because a potential attacker requires a substantial amount of knowledge and specific conditions in order to defeat the proposed security scheme. The new security proposed is based on AES cryptography and unless the pact of trust between the users and the publisher (SCANIA CV AB in this case) is broken it is very unlikely to be bypassed. This statement holds true until vulnerabilities are discovered within the AES encryption scheme.

10. Results Analysis

This chapter will analyze the results described in chapter 9. Section 10.1 will analyze theoretical results corroborated by the empirical attacks described in chapter 7, while section 10.2 will describe the improvements offered by the new security scheme.

10.1 Theoretical and Empirical Results Analysis

All types of software hide vulnerabilities inside its code. Managed software though implies some additional risks because of the JIT of the CIL code. C# and other managed programming languages are translated into assembly instruction only at run time, therefore when the software is stored on the hard drive the CIL is available to anyone who can access the file. The CIL can be decompiled with the ILDASM disassembler. Moreover, as illustrated in chapter 7 tools such as ILSpy can extract from the CIL entire pages of "ready to run" source code.

Due to these characteristics it is a bad engineering practice to hide in the source code any secrets. The .NET framework itself does not provide security features that could prevent the retrieval of any secrets such as a hard coded parameters or a secret algorithm encoded in the application.

10.1.1 Obfuscation Analysis

Almost every software producer that develops within the .NET framework resorts to obfuscation to protect their intellectual property. Using obfuscation is a good practice, but obfuscation is not a valid security mechanism. Obfuscation does not protect the intellectual property embodied inside the application's source code. It is merely a tool to delay an attacker understanding.

As shown in *Table 3* obfuscators are very expensive tools. Today companies put too much trust in this technique, hence they invest money that could be better used to implement other solutions.

The attacks against obfuscation in chapter 7 reveal that this technique is easily defeated. Moreover the analysis in section 8.2 suggests that, since obfuscators are defeated when their techniques become public, the development of a proprietary obfuscator would provide greater benefit than any commercial solution. A proprietary solution would force the attacker to develop a specific tool to de-obfuscate the application causing a considerable effort in time. Moreover, this solution would be more effective deterring less skilled attacker. However, the costs of developing and maintaining a proprietary obfuscator might be better spent providing real rather than pseudo-security.

10.1.1.1 Is obfuscation a good security mechanism?

No software application should rely on obfuscation to secure its sensitive data. It is poor engineering strategy to hardcode any security key, password, or secret algorithm and to relay on obfuscation for their protection. No company should use obfuscation with the intention to secure their secrets. Reverse engineering techniques illustrated in this report are able to retrieve any secret from the obfuscated code [15], [16], [17], [18], *Table 4*, and chapter 7.

10.1.1.2 Is obfuscation ineffective?

Obfuscation is indeed useful in several scenarios. First of all is a good method to slow down inexperienced attackers or to deter a non-fully determined one. It is a very good technique for small software companies who frequently release new version of their software by the time a version has been hacked a new version is already out. There is also a *marketing* perspective to obfuscation, which can be read the following way:

Users who are keen to buy your application will probably do so, while a malicious user who does not want to buy your software will never do so, obfuscation or not. Therefore obfuscation is a good technique to keep a good user loyal without having them reading your source code and developing a quick alternative [15], [16], [17], [18], *Table 4* and chapter 7.

10.1.2 Reverse Engineering

Chapter 7 provides practical examples of basic and advanced RE techniques. A reverse engineering attack can bypass security functions of software applications or retrieve hard coded secrets. To mitigate the power of this technique the only solution is to act on the human factor. Since RE is a technique that requires meticulous manual work from the attacker the solution is to increase the required skills and the patience of the perpetrator. To achieve this, a high degree of complexity must be added to the application. Complexity can be added by different means, with a specific programming style, third party applications, or other techniques. Section 8.3.1 illustrates how some programming techniques can slow down the process of RE by adding specific instructions to eliminate breakpoints or to modify the PE header (section 8.3.2).

The use of third party software packers (section 8.4) greatly improves the security of .NET applications against RE. Tools such as The Mida, are capable of disabling the common RE tools used for disassembling and memory dumping. Moreover, the CIL of the .NET application is translated into a proprietary byte-code and executed in a virtualized environment boxed into several virtual machines. All of these operations force the attacker to perform additional time consuming hard work to understand the system that is being attacked.

Cryptography offers the first layer of defense against RE. If software is always stored and encrypted following best security practices RE becomes infeasible if the attacker possess only the encrypted copy of the application. Good cryptographic keys must be generated following

standards such those presented in section 2.7 and use secure algorithms such as AES(section2.3) or RSA (section 2.4), must be used. Obfuscation is only a barrier to inexperienced attackers even if a good obfuscator is used and properly configured.

10.1.3 Software Packagers

Software packers are really good tools and greatly increase the complexity of a .NET application. The latest generation of these tools is exemplified by The Mida which introduces virtualization technology that makes direct memory dumping ineffective. The use of such tools is highly advised to protect .NET applications.

These software packers are frequently subject to the attentions of crackers. It is very important to always utilize the latest version of the packager. Moreover the use of this anti-cracking tool by itself does not guarantee a good security. Different security mechanisms should be used and appropriate security policies and enforcement of these polices are needed to guarantee the effectiveness of these tools.

The Mida has been selected due to the high quality of the features it implements and the innovation capability of its producer. Moreover, it is highly regarded as one of the toughest anti-cracking tool by the reverse engineering communities.

Most of the evaluation of packagers occurred relating with experienced (anonymous) reverse engineers who gently answered the author's questions on the matter. No details of the interviews conducted are given. In this thesis to respect the privacy of this communication (which was promised before the interviews)

10.1.4 Misuse of hardware security token

The use of a hardware token by itself does not guarantee the security of software. Mismanagement of these tools may lead to new threats. This is the case with Tool-A which uses a USB dongle key (section 6.2). The old dongle called the "*green*" key is known to be broken, but has not been revoked. As a consequence of this a malicious user might utilize the old hardware token and therefore avoid the new security implemented by the new "*black"* dongle key. It is important to implement a revocation system that forces users to update to the version of the application to enforce the use of the security features of this latest version of the application. The NEVE group is highly advised to revoke the old "*green*" key, so that new efforts in security are not corrupted by old threats.

10.1.5 Hard Coded Secrets

As demonstrated in sections 7.2.1 and 7.2.2 hard coding secrets into an application is a bad engineering practice in the .Net framework. It is highly recommended that any algorithm or key

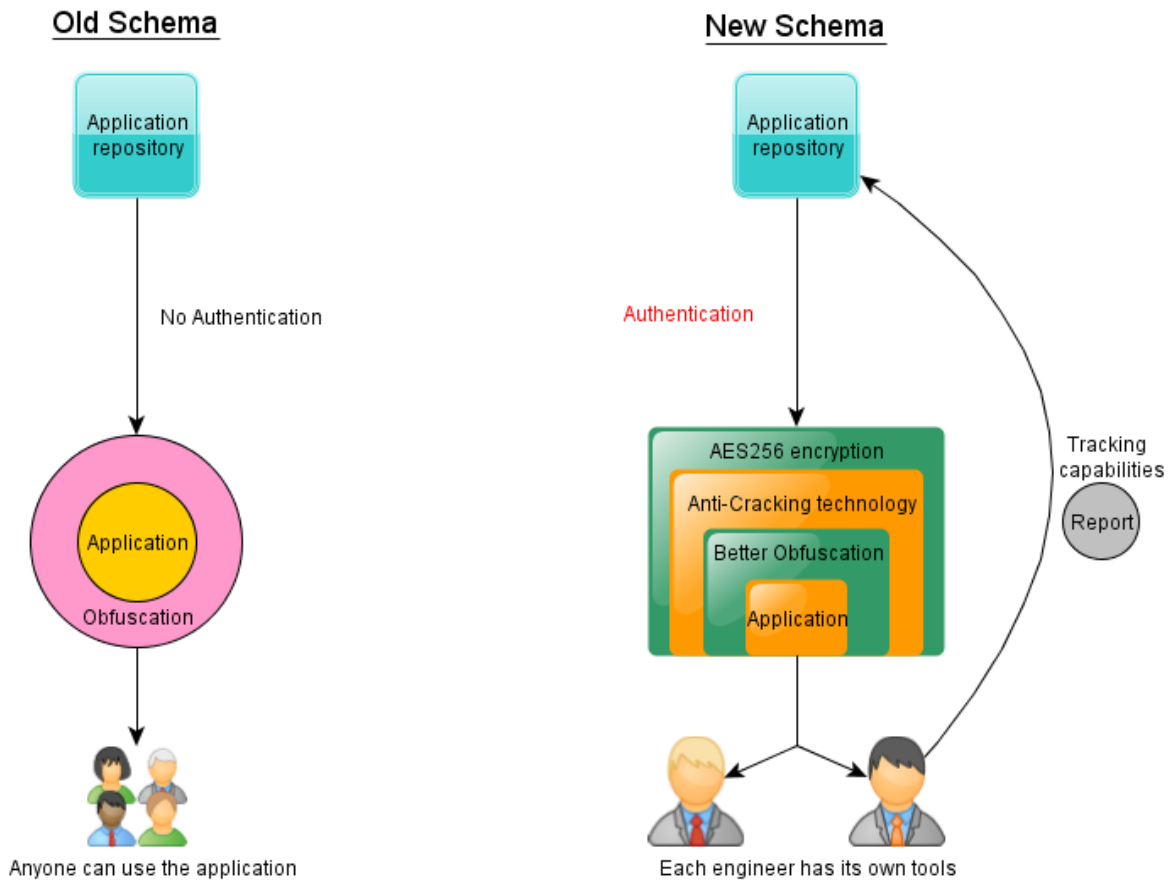
that needs to be hardcoded is written in a separate unmanaged software language and then loaded as a separate DLL by the application. It is a good psychological approach to consider any hardcoded information as already public when programming and therefore should engineer the software accordingly.

10.2 Evaluation of the New Security Scheme

The new scheme introduces several characteristics. In first place multiple layers of complexity are added to the .NET application. This is achieved with the introduction of cryptographic techniques, better obfuscation, software packers, and improved programming styles. Moreover, the new installation procedure for Tool-A and Tool-B introduces unique new features. In first place the applications' publisher will have more control over the distribution of its tools. Additionally this scheme utilizes unique cryptographic keys for each instance of each tool, building a pact of trust since the tools are now uniquely bound to the users. The user should be made aware of the unique binding; therefore, potentially malicious users will be reluctant to give out information that would uniquely identify them. The use of tracking netcode will allow tracing of possible application leaks and facilitate identification of the source. This feature may prevent future leaks and enables better monitoring of the use of the development tools. The information that is collected instruments will also ease any investigation of a leak.

The visual comparison renders a clear image of the improvements. The improved programming techniques illustrated in section 8.3 constitute an additional layer of security within the application.

Figure 29 - Evaluation of the new solution



10.3 Artifacts Description

During this project several artifacts were developed. This section lists and summarizes the scope of these artifacts and their main characteristics.

Application repository web page

A web page was written in HTML5 with input checks and basic graphics. This provides the user interface to the web application.

Web application

A C# .Net web application was written to handle the installation operations. The main features are IO operations, cryptography procedures, hash functions, and e-mail handling features. This program also creates and configures the *settings.db* and *tracking.db* files.

RSA digital signature for XML files

Three tools have been coded. One tool can generate RSA key pairs of an arbitrary bit length. A second tool is used to sign XML files. A third application is used to verify the RSA digital signature of an XML file.

Application launcher

This multithreaded artifact is the main result of the new security scheme. It has been developed in .Net C# and it performs numerous operations. It handles cryptographic functions, hashes, IO operations, utilizes multiple threads, and incorporates stealth feature of machine information gathering. The netcode can upload silently reports to a remote FTP server. The launcher implements a simple graphical interface with a text field to input the secret password.

The report monitor

This program was developed in C#, monitors the FTP server for new reports that may occur if an application leaks. If a new report is available it will decrypt it and deliver it to the NEVE's mail address.

AES256 file encryptor

This program developed in C# and can decrypt or encrypt any file based on an arbitrary password which can be selected at runtime.

11. Conclusions

The results of this study produced a promising method for securing managed software. Adding several layers of security mechanisms to increase the complexity of a .NET application is an effective methodology to delay software piracy. The study reveals that even for producers with limited resources it is possible to protect .NET application against the majority of software related threats. Powerful techniques such cryptography and virtualization limits the effectiveness of reverse engineering attempts to a very small pool of highly skilled individuals. In particular strong cryptographic schemes can protect software even from entities with unlimited amount of resources such as governative agencies [30].

This study involved a series of limitations for .NET applications. The software considered during this research is not connected by any mean to the Internet network. This limitation restricts the control over the applications. Moreover, restricts the possibility of authenticating users effectively. Nonetheless a form of authentication was achieved by exploiting the installation procedure. This authentication mechanism uniquely identifies the users, binding the user to a digital identity and allows the implementation of a basic monitoring mechanism.

The identification is achieved exploiting the unique binding between the employee and their corporate e-mail. The binding between the user and the application is achieved by securing a hash signature with cryptography.

12. Reflections and weaknesses

Despite the substantial improvements that can be achieved with the right methodologies, managed software security is still an open issue. The best solution is still to avoid delivering software to the users that contains any sort of knowledge that the vendor wants to preserve. With the spread of the Internet this can be achieved easily by delivering to the users only client interfaces and storing the algorithms and programming secrets on the server side. When this is not possible a shift from managed to unmanaged software is desirable since disassembling unmanaged software is a non-trivial task.

13. Future Work

This thesis project leaves room for additional improvements. This chapter describes the first few improvements that have been discarded due to the short amount of time available and the limited resources and then will illustrate a major project that would greatly increase the security of the .NET applications.

13.1 SSL connection

The first improvement that should be implemented is to establish an SSL connection during the installation process. Although the installation takes place inside the trusted SCANIA's intranet it is a good practice to ensure that the passwords between the browser and the web application are not sent in clear.

13.2 Yubikey

Among the discarded solutions the use of the Yubikeys was suggested. Yubikey is hardware authentication token that looks like a small USB memory stick, but it is actually a keyboard. All YubiKeys holds two separate identities that can be easily configured to any of the following:

1. Yubico Standard OTP - 12 character ID + 32 character OTP, for Yubico servers,
2. OATH OTP - 6 or 8 digit OTP, for third party OATH servers,
3. Static pass code - 1-64 characters for legacy login applications,
4. Challenge-response - Using client software.

One example illustrating how such a key might be used in SCANIA's current systems is to authenticate the users' login into their Windows workstation with pGina. pGina[34] is a pluggable Open Source GINA[35] and credential provider replacement.

A major project derived from this thesis and is to engineer proprietary obfuscator and packager software. Since obfuscation relies on the obscurity of its transformation, develop a proprietary obfuscator would greatly improve this feature. There would be no knowledge available on the internet about the technique used in such a tool. This will force the attacker to perform deeper and complete study of the tool consuming a considerable amount of time and most probably discouraging the attacker.

[34] Open Source Windows Authentication. http://pgina.org/

[35] A Graphical Identification and Authentication dynamic-link library (DLL). The GINA is a replaceable DLL component that is loaded by the Winlogon executable. http://is.gd/V9uBbt

References

[1] J. Daemen and V. Rijmen, “The First 10 Years of Advanced Encryption,” *Security Privacy, IEEE*, vol. 8, no. 6, pp. 72 -74, Dec. 2010.

[2] H. Krawczyk, M. Bellare, R. Canetti, “RFC 2104 - HMAC: Keyed-Hashing for Message Authentication.” [Online]. Available: http://tools.ietf.org/html/rfc2104.

[3] Ron Rivest, [Python-Dev] hashlib - faster md5/sha, adds sha256/512 support". Mail.python.org. Retrieved 2010-08-09. Available: http://mail.python.org/pipermail/python-dev/2005-December/058850.html

[4] M. Bishop, *Computer Security : Art and Science*. Boston: Addison-Wesley, ISBN 0-201-44099-7, 2003.

[5] J. Jonsson, B. Kaliski, “RFC 3447 - Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1.” [Online]. Available: http://tools.ietf.org/html/rfc3447.

[6] B. Kaliski, “PKCS #5: Password-Based Cryptography Specification Version 2.0,” *Internet Request for Comments*, vol. RFC 2898 (Informational), Sep. 2000.

[7] D. E. 3rd, J. Schiller, and S. Crocker, “Randomness Requirements for Security,” Internet Request for Comments, vol. RFC 4086 (Best Current Practice), Jun. 2005. [8] “RFC 4226 - HOTP: An HMAC-Based One-Time Password Algorithm.” [Online]. Available: http://tools.ietf.org/html/rfc4226.

[8] D. M’Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, “HOTP: An HMAC-Based One-Time Password Algorithm,” Internet Request for Comments, vol. RFC 4226 (Informational), Dec. 2005.

[9] D. M’Raihi, S. Machani, M. Pei, and J. Rydell, “TOTP: Time-Based One-Time Password Algorithm,” *Internet Request for Comments*, vol. RFC 6238 (Informational), May 2011.

[10] J. H. Saltzer and M. D. Schroeder, “The protection of information in computer systems,” *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278 - 1308, Sep. 1975.

[11] Microsoft, .NET Framework Developer Center. [Online]. Available: http://msdn.microsoft.com/en-us/netframework/.

[12] E. J. Chikofsky and J. H. C. II, “Reverse engineering and design recovery: a taxonomy,” IEEE Software, vol. 7, no. 1, pp. 13–17, Jan. 1990.

[13] E. J. Byrne, “A conceptual foundation for software re-engineering,” in *Software Maintenance, 1992. Proceerdings., Conference on*, 1992, pp. 226 –235.

[14] W. L. Simon and K. D. Mitnick, *The Art of Deception : Controlling the Human Element of Security*. Indianapolis, Ind.: Wiley, 2002.

[15] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella, “The effectiveness of source code obfuscation: An experimental assessment,” in *Program Comprehension, 2009. ICPC ’09. IEEE 17th International Conference on*, 2009, pp. 178 –187.

[16] S. K. Udupa, S. K. Debray, and M. Madou, “Deobfuscation: reverse engineering obfuscated code,” in *Reverse Engineering, 12th Working Conference on*, 2005, p. 10 pp.

[17] B. Anckaert, M. Madou, B. De Sutter, B. De Bus, K. De Bosschere, and B. Preneel, “Program obfuscation: a quantitative approach,” in *Proceedings of the 2007 ACM workshop on Quality of protection*, New York, NY, USA, 2007, pp. 15–20.

[18] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella, “Towards experimental evaluation of code obfuscation techniques,” in *Proceedings of the 4th ACM workshop on Quality of protection*, New York, NY, USA, 2008, pp. 39–46.

[19] J. Alex Halderman, Ariel J. Feldman, Voting machine hack https://jhalderm.com/pacman/.

[20] N. Williams, "RFC 5056 - On the Use of Channel Bindings to Secure Channels", November 2007 [Online]. Available: http://tools.ietf.org/html/rfc5056.

[21] "RSA Laboratories - 1.3 What are some of the more popular techniques in cryptography?" [Online]. Available: http://www.rsa.com/rsalabs/node.asp?id=2158.

[22] Robert B. Ash, "Abstract Algebra: The Basic Graduate Year", Ring Fundamentals, 2012. [Online]. Available: http://www.math.uiuc.edu/~r-ash/Algebra/Chapter2.pdf

[23] Colin Percival, "Tarsnap - The scrypt key derivation function and encryption utility." [Online]. Available: http://www.tarsnap.com/scrypt.html.

[24] Rohit Dhamankar, Mike Dausin, Marc Eisenbarth, James King, Wolfgang Kandek of Qualys, Johannes Ullrich, Skoudis Lee, Rob Lee, "*The Top Cyber Security Risks*", SANS Institute faculty, September 2009, [Online] http://www.sans.org/top-cyber-security-risks/

[25] W. Stallings, Cryptography and network security: principles and practice. Boston: Prentice Hall, 2011.

[26] V. Yakovyna, D. Fedasyuk, M. Seniv, and O. Bilas, "The Performance Testing of RSA Algorithm Software Realization," Lviv-Polyana, Ukraine, 2007, pp. 390–392.

[27] CIL instructions source: http://msdn.microsoft.com/en-us/library/812xyxy2%28vs.71%29.aspx, http://en.csharp-online.net/CIL_Instruction_Set.

[28] Reverse Engineering of Skype http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-biondi/bh-eu-06-biondi-up.pdf

[29] W. R. Ashby, *An introduction to cybernetics*. London: Methuen, 1976.

[30] James Bamford, http://www.wired.com/threatlevel/2012/03/ff_nsadatacenter/all/1, March 2012,

[31] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel, "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses," in Security and Privacy, 2008. SP 2008. IEEE Symposium on, 2008, pp. 129–142.

[32] Robert N. Charette, http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code, February 2009

Appendix I – Non Disclosure Agreement

This document is released under a non-disclosure agreement between the author and SCANIA CV AB. If you want to know if you are eligible for a full copy of this report please contact the author at tommaso.galassi.de.orchi@scania.com.

SCANIA CV AB Confidentiality classification:

Handling and protection	Internal	Confidential
Storing / filing • In office/desk	Ok	Locked storage area
• Shared files	Ok	Access control, Individual traceability.
• Laptop / Other portable devices	Encrypted Laptop / Ok	Encrypted
• Outside Scania / Non Scania device	Approved by line manager	Approved by Information owner. Risk Analysis shall be performed.
Printing	Allowed on network printers	Allowed on local or network printers, if supervised / Secure printing
Copying and passing it on	Allowed	Decided by Information owner
Destruction	No requirements	Approved manner or device (example shredding, burning, cutting, secure container).

Original report classification: **INTERNAL.**

This version of the report is: PUBLIC.

The PUBLIC version of the document may contain missing sections or paragraphs. Not all images and data are available in the PUBLIC version of this report. Please for a full list of restriction in place contact the author.

The author is subject to the following code of conduct:

1. Unwavering loyalty and mutual confidence between Scania and all employees,
2. As an employee, you must not disclose anything of a confidential or secret nature concerning Scania's business or other relationships,
3. Employees have a duty to comply with all Scania security rules.

Appendix II – Discarded Solutions

This appendix describes some discarded solutions.

The first discarded solution in *Figure 30* was to use a mobile device such a smartphone to authenticate the engineer before logging into the application. The idea is that the mobile device could have stored an algorithm or a key needed to run the application. One possible implementation was to use QR codes. The code would have been used to store a cryptographic key. Then thanks to the laptop's webcam it would have been enough to read the QR code and log the user in.

A second discarded solution illustrated in *Figure 31* was to implement a TOTP authentication. An application on the smartphone would have provided a time value in synch with the authentication server at SCANI's headquarter.

There are multiple ways to implement these solutions. Under certain condition smartphone are not required and hardware token can be used such as the YUBIkey. All these solutions have been discarded for different reasons. The main factor was that not all SCANIA's engineers possess a smartphone. Moreover SCANIA did not want to enforce the use of mobile during the work time therefore not simple J2MEE solution could be evaluated.

Other negative comments were the user unfriendliness and the costs of external hardware.

Figure 30 - Discarded solution: Mobile token authentication

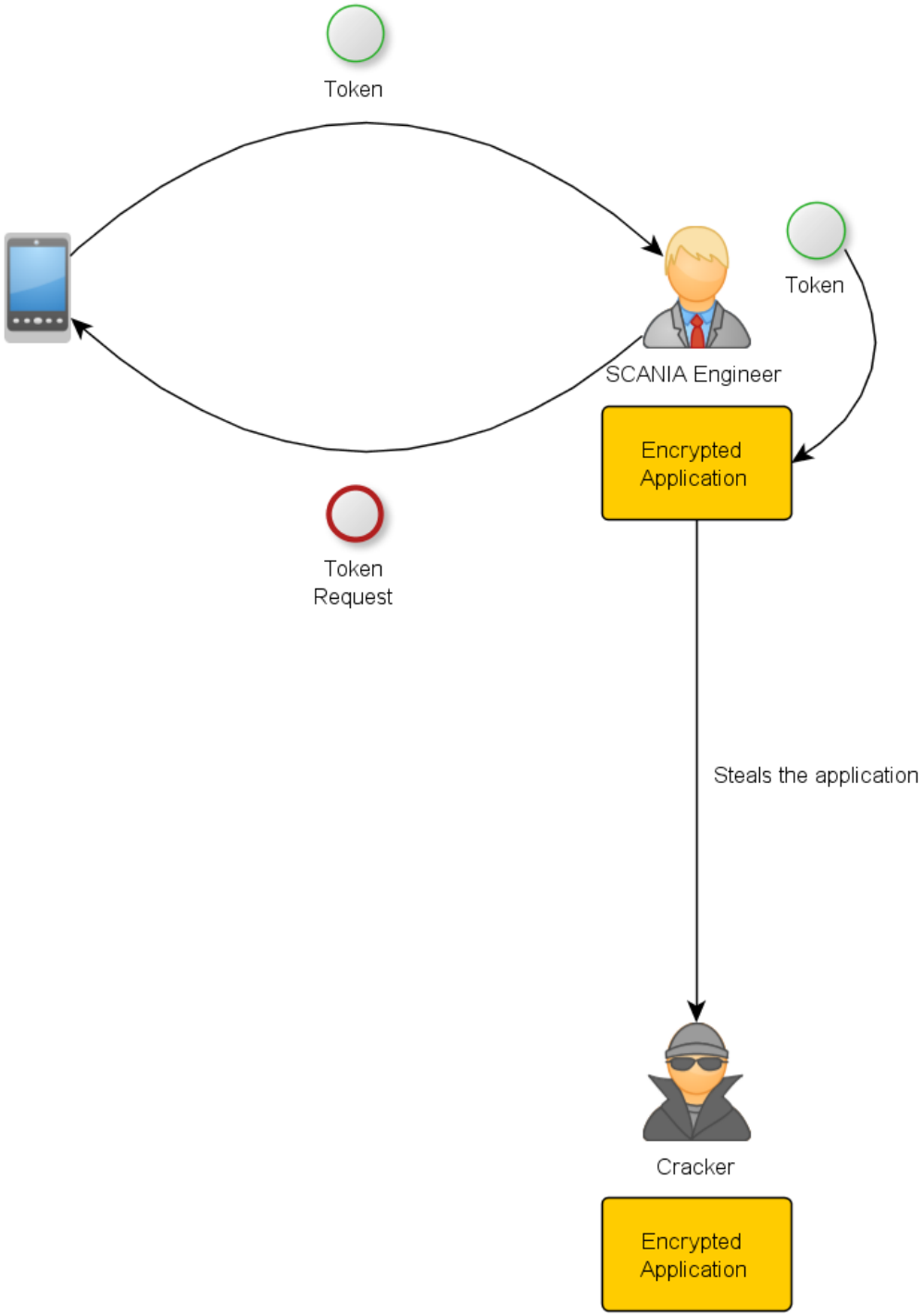
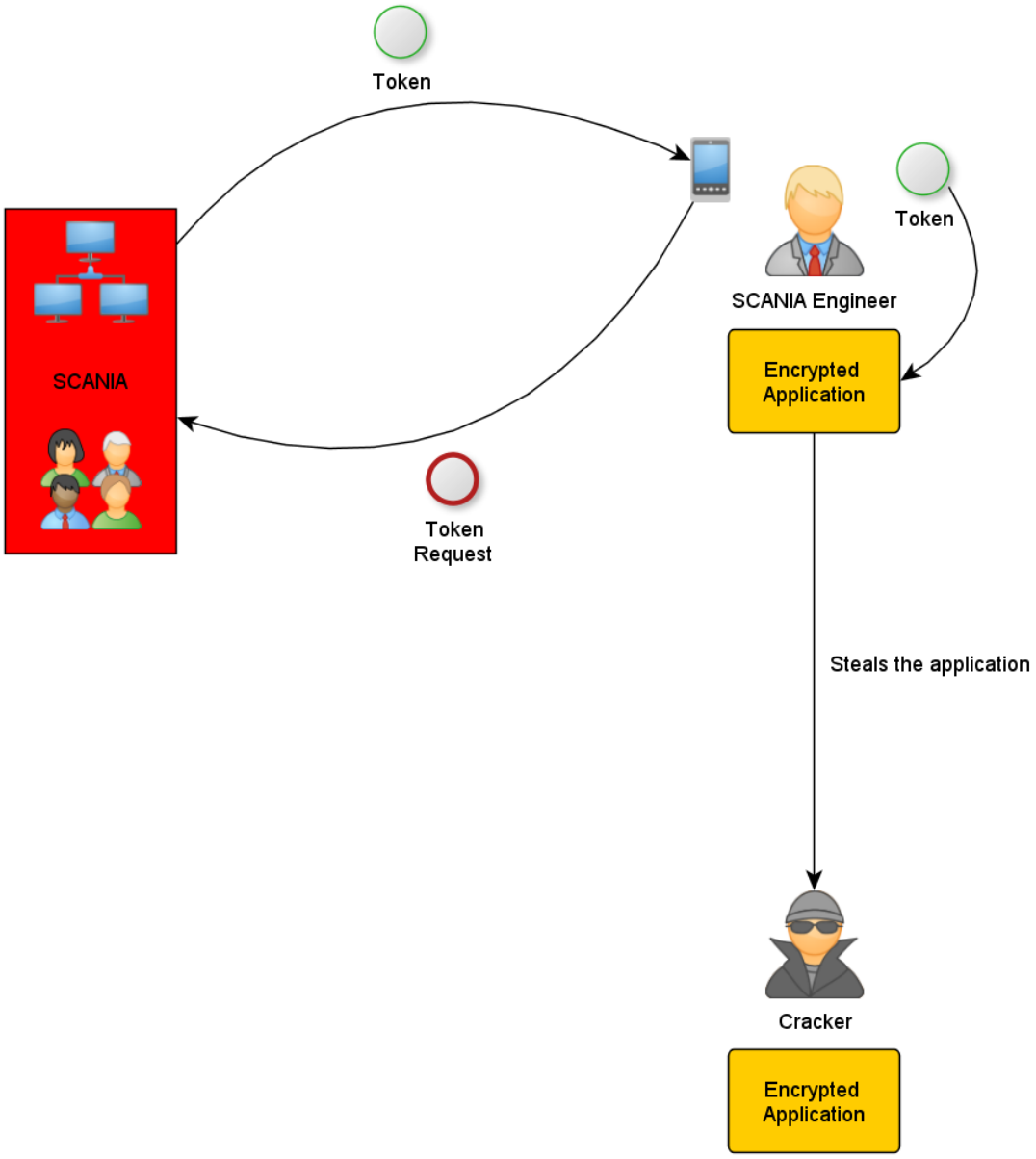


Figure 31 - Discarded solution: TOTP schema



TRITA-ICT-EX-2012:149

www.kth.se