UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Computer Science

Aleksei Gorny

Analysis of Chip-card Based Authentication

Bachelor's thesis (6 EAP)

Supervisor: Sven Laur, PhD

Author: ..... "....." June 2009
Supervisor: ..... "....." June 2009

Admitted to thesis defense

Professor ..... "....." June 2009

Tartu 2009

Contents

Introduction

Most chip card solutions for personal computers assume they are used in a secure environment and that the communication between the card and applications cannot be modified. This assumption is largely unjustified, since most users lack technical knowledge to have sufficient control over their machines. Indeed, as yet another attestation of the fact, earlier this year, a botnet comprising of approximately 1.9 million infected devices, of which many belonged to government and business institutions, was discovered [15]. The scope of this work is the use of chip cards in untrusted environments. The research has been conducted in accordance with the agreement between the author and AS Cybernetica and some findings may have been left out from this paper version due to contractual obligations.

For simplicity, we explain the basic concepts of chip card use by the example of the Estonian ID-card. ID-card is the Estonian primary personal identification document, issued by the national Citizenship and Migration board. As it is the case with national chip cards of most countries, the card enables its holder to create digital signatures and to authenticate to both state and private enterprise services. This functionality can also be used online from a personal computer equipped with a smart card reader, some third-party software and an internet connection. Important web-based services accessible this way include electronic banks, e-voting polls, health and education related information systems etc.

In this work, we review the situation where a cardholder authenticates to a remote server over a network. We show that due to implementation issues there currently exists a way for a malevolent party with temporary user-privileged access to the cardholder's computer to effectively tamper with the process. This may potentially result in the honest user unintentionally authenticating to some other entity instead or the server believing that the client resides at a different IP address. The consequences of this may in turn include loss of privacy and personal data, financial losses and incorrect depiction of the cardholder's opinions and preferences to other entities.

In the following chapters we present a detailed overview of the authentication process with the currently employed chip card software, describe possible attacks, evaluate their feasibility and conclude by introducing obvious security-enhancing modifications to the affected components. We also comment on the exploits as we implemented them on the Ubuntu Linux operating system for demonstration purposes. The code itself does not accompany this work, but is available on request from AS Cybernetica.

1 Background and technical details

Though the basic design principles of different chip cards are similar, specific cards often have technical peculiarities and are subject to different standards and legislation. In this section, we review the concepts by an example national chip card, the Estonian ID-card. The development process of the card has been mostly open for public inspection and information found here can also be gathered from online sources.

Since the introduction of the ID-card to the public in 2002, the number of cardholders and applications for the card has been steadily increasing. The Citizenship and Migration board reported that by the first of April 2009, there were 854 675 registered ID-card owners, a large number, considering the population of Estonia. It is expected that more than half of the cardholders use the card to access online services.

The reason ID-cards exist and are increasingly popular is the joint efforts of government institutions, security researchers and businesses. At the same time when the Citizenship and Migration board became interested in replacing the passport with a simpler modern identification document in 1997, researchers from AS Cybernetica and Hansapank were working towards developing a solution for digital authentication and signing. Eventually, the goals of both interest groups were united in the ID-card and legislation was modified to accommodate electronic authentication mechanisms [5] and acceptance of digital signatures [6]. Businesses and institutions were encouraged to adopt ID-card based authentication methods and use digital signatures to reduce paperwork and allow simple and secure digital access to various services, making it favorable for citizens to switch to the new technology. The full timeline and the development story can be found at the ID-card support site [1].

As now, the card can be physically used for authentication in place of the passport for travel in the European Union, customer cards of several Estonian shops and store chains, library cards of most of the Estonian libraries, etc. A large list of web service providers allowing ID-card based authentication is maintained online [1]. The list includes but is not limited to the following:

- Financial institutions: Swedbank, The SEB Group, Eesti Krediidipank, Sampo Pank, Nordea Bank, BIG Bank, Parex Bank, Nasdaq OMX Estonian securities market,
- Government services: the National Electoral Commitee, Estonian Motor Vehicle Registration Centre, the Commercial Register, Estonian Tax and Customs Board,
- Education services: study information systems of the Tartu University and Mainor Business school, the eKool system,
- Medical services: patient information systems of the East Tallinn Central Hospital and the Medicum health center.

For a cardholder, accessing these services from a personal computer is as easy as purchasing a smart card reader and installing software from the ID-card support page. The necessary software consists of drivers for the card and the reader and extensions for the browser that allow it to query the drivers to access the card functionality when needed.

What would happen if a vulnerability in the authentication process was found? Some services like e-banks and e-voting polls would remain relatively secure from the functional standpoint, as they require digital signatures for finalizing critical transactions like money transfer or vote casting. However, an attacker able to exploit such a vulnerability could still perform many potentially damaging, but non-critical operations without the cardholder's knowledge. Additionally, one could gain unauthorized access to personally identifiable sensitive information contained in above-mentioned information systems, like financial status, health conditions, study grades, electoral preferences etc. A possibility of large scale exploitation, for example, if the vulnerability was common to national chip cards of multiple countries, would serve as motivation for cyber criminals and bear drastic consequences to card users.

In this chapter, we review the prerequisites for understanding the current state of ID-card based online authentication. We look at the functionality the chip of the card provides, the specification of the authentication protocol that is used and at how this protocol is implemented in software.

1.1 ID-card hardware

From the hardware perspective, the Estonian ID-card is a chip card conforming to the ISO-7816 standard [9] and based on the Orga Micardo Public 2.1 chip [7]. It hosts some minor technical modifications that allow it to be used with a larger variety of smart card readers and changes to the instruction set that forbid formatting and EEPROM memory initialization. This way the card is better suited for wide-scale public use and prevents users from deleting or improperly modifying important data it contains.

The data stored on the card and available operations are subject to a strict immutable access policy. Significant objects accessible to a common user constitute of the cardholder's personal information file, signing certificate and authentication certificate.

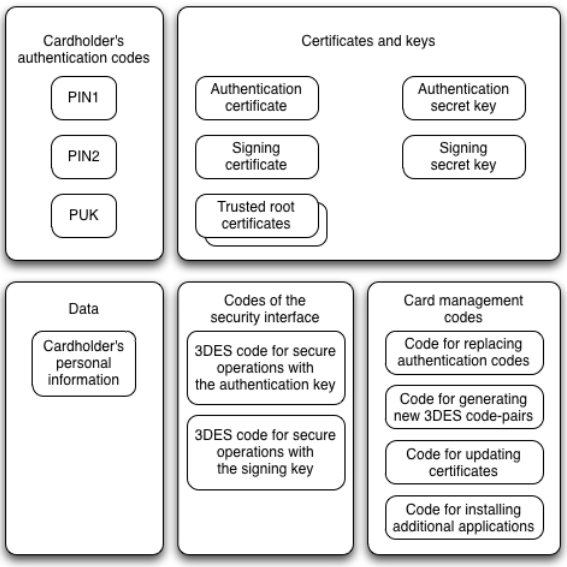

Figure 1: Objects on the ID-card. Translated from [3].

In addition to the operations for reading these objects, the card provides cryptographic operations with internal secret keys – in the case of authentication, computing a response to a SSL/TLS challenge [13] using RSA or SHA1 with RSA. Cryptographic operations require the cardholder to set different security environments on the card by supplying appropriate PIN codes. Optionally, additional codes may be entered to enforce the interaction between the card and host applications to be encrypted using 3DES.

PIN codes are protected from brute force attacks by counters of consecutive incorrect entries. After three unsuccessful entries, a PIN is blocked and has to be revalidated or replaced using the PUK code, that is itself subject to an analogous counter. The PUK code can, however, be unblocked only at accredited Token Management Centers - bank offices and the service offices of the Citizenship and Migration board. Other card management operations, like updating user data and certificates, are also performed under official supervision - either on-site and the centers or remotely over the network, secured via cryptographic means. This again is beneficial for protection against unauthorized or accidental data modification and deletion.

For a full list of objects and operations supplied by the card, one may refer to its reference manual [3].

1.2 The Transport Layer Security protocol

Transport Layer Security (TLS) [13], the successor of Secure Socket Layer (SSL) [14], is a popular protocol for providing communication confidentiality and integrity by establishing a reliable private channel between two peers. TLS achieves its security goals by using symmetric cryptography with unique keys generated for each connection and message authentication codes. The TLS handshake sub-protocol provides a secure and reliable way to negotiate the parameters of a connection and allows peers to authenticate to each other using asymmetric or public key cryptography. In a typical setting, TLS dwells on an available public key infrastructure and is unilateral, meaning that the server gets authenticated to the client, while the latter may remain anonymous.

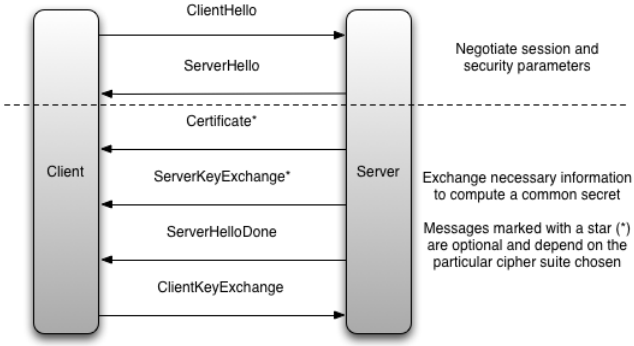

Figure 2: Negotiation of anonymous or unilaterally authenticated TLS

In the first part of the TLS handshake, the negotiation phase (Fig. 2), the client and the server agree on the strongest cipher suite and hash functions they both support, exchange random values and agree on a common secret. During this phase, the server

usually shares its certificate with the user, who is expected to verify the server's identity. If necessary, the server sends an additional message containing cryptographic data allowing the client to communicate the premaster secret. Both parties then compute a master secret key based on the premaster secret and random values. This key is used for symmetric encryption of the final handshake messages and communication later on.

The handshake finishes with the peers validating its correctness. First, the client informs the server that all of its following communication shall be sent encrypted using the freshly computed symmetric key. Next, it sends an encrypted message containing a MAC over the protocol transcript (Fig. 3). The server decrypts the message, verifies the hash and responds with two analogous messages. Now, if either party fails to decrypt the received final message or verify the MAC inside, the connection is terminated and has to be renegotiated. This ensures both peers agree on the generated security parameters and keys and that the handshake has not been tampered with.

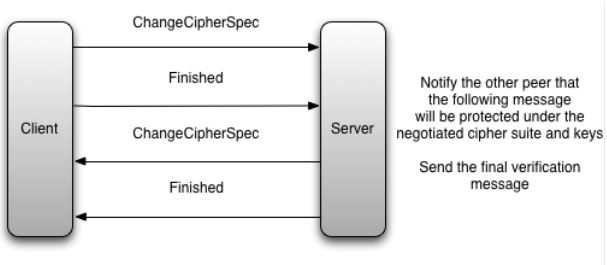

Figure 3: TLS handshake final messages

TLS also supports a bilateral mode, known as mutual authentication. In this case, the client also sends out a certificate and afterwards proves that one indeed owns it by showing one has access to the corresponding private key (Fig. 4). For this, the client sends a certificate verification message, containing the concatenation of all previous handshake interaction signed with the key. If the protocol is successful, then unlike the unilateral case, both parties are assured of each other's identity.

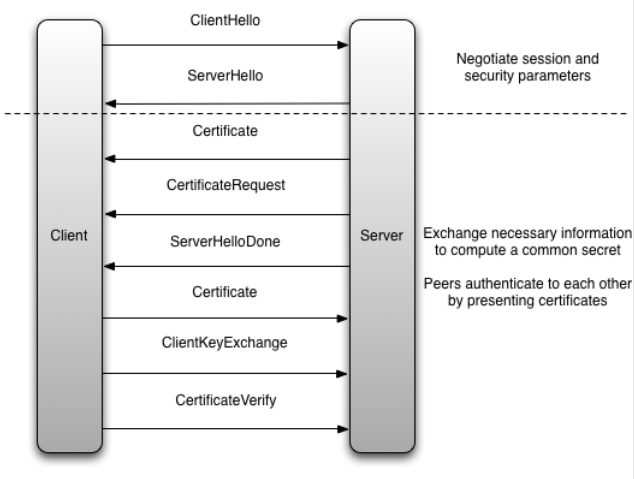

Figure 4: Negotiation of mutually authenticated TLS

After the handshake is finished, relevant data messages can be transferred in a way similar to the final handshake message - encrypted with the negotiated symmetric key

and verified with a message authentication code.

TLS is one of the most common protocols for securing application layer data when communicating over customized networks or the Internet. It can run on top of a reliable transport protocol such as TCP and beneath application layer protocols such as HTTP, FTP, SMTP etc. TLS can also be used for creating virtual private networks by tunneling the entire network stack. For our purposes, we are mostly interested in the scenario where mutually authenticated TLS is used in combination with HTTP for accessing various web services.

1.3 Software architecture

In practice, communication between a web service and a chip card passes through multiple modularly composed software layers. The layers are similar for all operating systems, so for conciseness we describe the detailed architecture as it is common for Ubuntu Linux. Low-level operations, like communicating bits to and from the card, are handled by a low-level driver for smart card readers, typically OpenCT [11] or PC/SC Lite [12]. On top of the driver resides OpenSC [10], an open-source framework for high-level operations with smart card tokens. It implements methods for recognizing different card hardware and vendor-specific hexcode instructions.

For integration into existing applications, OpenSC compiles a dynamic library based on the PKCS#11 standard [8]. PKCS#11, also known as Cryptoki, is a widely-used standard for cryptographic token libraries that specifies common names for objects and operations. This dynamic library interface is, for example, used by most of the popular browsers, so that when the token is initialized, they are able to request data and perform card-assisted cryptographic operations by communicating with the driver.

As an illustration, let us see how these components interact when a user attempts to authenticate to a server using a capable chip card and TLS mutual authentication (Fig. 5). In this case, the user's browser takes care of the TLS protocol messages and makes two requests to the driver. The purpose of the first request is to retrieve the authentication certificate and the purpose of the second is to compute the response to the protocol challenge using the secret key stored on the card and corresponding to the certificate. As computing the response requires toggling the security environment on the card, the driver expects the browser to obtain the PIN code from the user.
This architecture is secure under the assumption the user actually has full control over the client machine. Indeed, an adversary is then unable to eavesdrop or modify the communication between both the software and hardware components - effectively between the browser and the smart card. Also, properly executed mutually authenticated TLS, as reviewed earlier, eliminates the possibility of client- and server-side identity switching on the network.

The authentication process becomes insecure once we assume the adversary has temporary user-rights level logical access to the machine. Note, that on modern operating systems, this type of access is sufficient for local installation of software packages and browser extensions, but does not allow to change preferences of the system itself or files of drivers and properly installed applications.

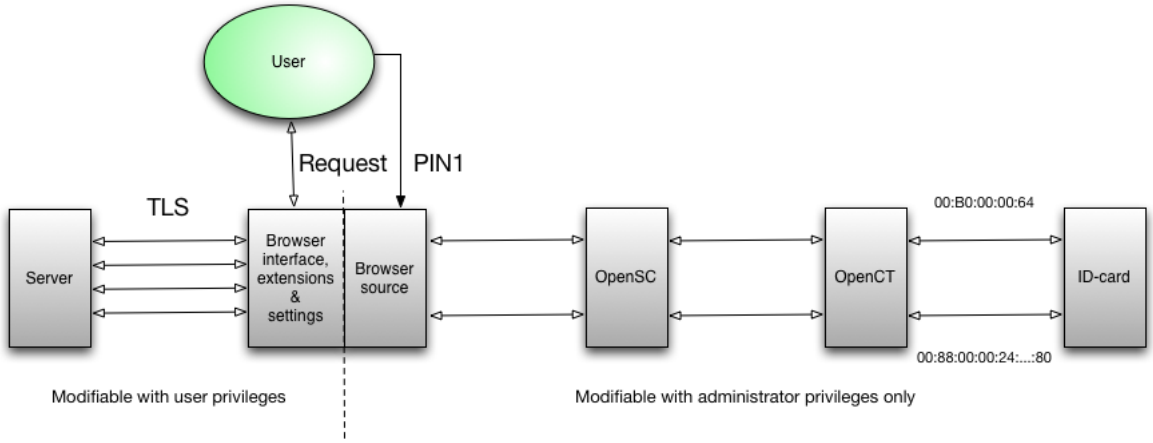

Figure 5: Authentication with a chip card

The main problem lays in the fact that the relation between the TLS session the user is attempting to establish and the challenge message send to the driver is never verified. This implies that in customized software, the PIN code of the user may actually be used for computing the a challenge response for another session. This problem is not unique to any particular chip card, but common to all chip cards used with this software framework.

An example exploit for this would be a custom browser extension that hijacks chip card based authentication sessions by sending the driver challenges that are different from the intended ones (Fig. 6).

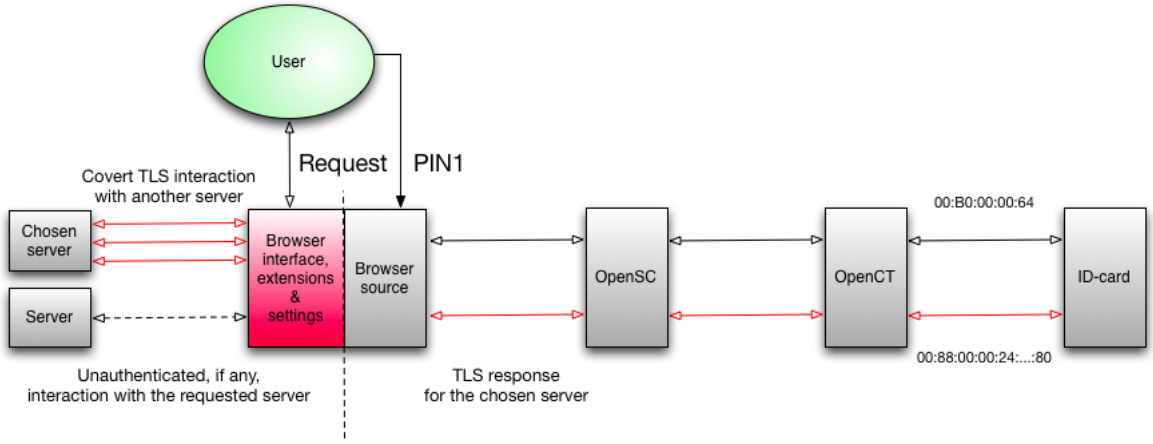

Figure 6: A malicious browser extension

For a second example exploit, temporary access can be used for local installation of a malicious Cryptoki-based library. Here the attacker can rely on the fact that pointing out the location of OpenSC to the browser needs user rights only. The substitute library could then act as a mediator between the browser and the chip card and perform operations like publishing PIN codes, modifying user queries or establishing unwanted TLS sessions (Fig. 7).

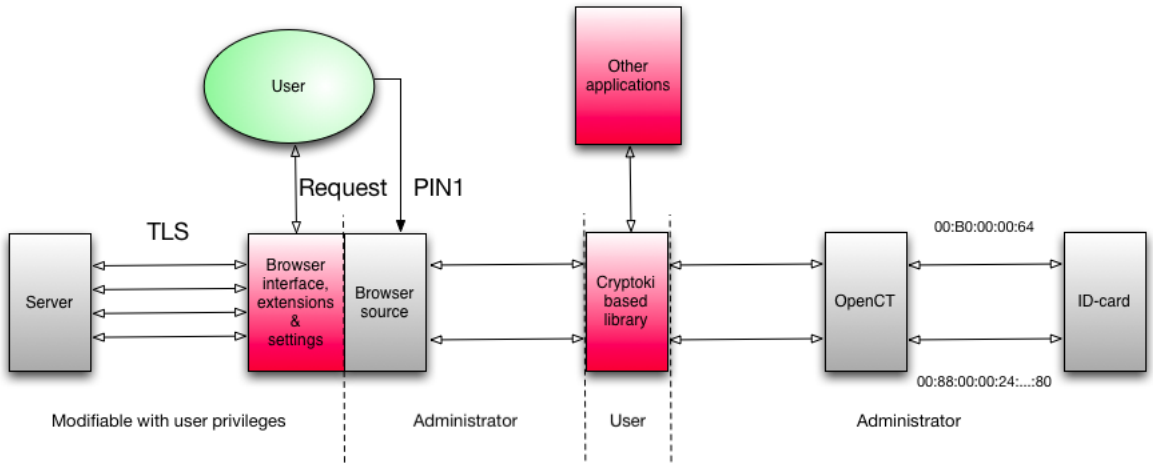

Figure 7: OpenCT is substituted for a malicious library in the browser

For the cardholder, the consequences of these exploits may be as severe as mentioned in the beginning of this chapter. Note that active presence of the adversary that introduces software modifications is not necessary in either case. After the software is changed, session hijacking and other activities, for example using the hijacked session for gathering personal data, can happen in an automated fashion.

2 Attacking the authentication process

In this chapter, we review some attacks associated with authentication and evaluate their successfulness and feasibility against chip-card based authentication. These attacks can be conducted by cyber criminals or otherwise malevolent individuals against honest users using chip cards or other methods to authenticate to some entity over a network. For each attack, we first give its general description and explain its background and then reflect on our experiments with it in practice.

2.1 Logging authentication codes

Keystroke logging in general refers to a practice of noting the keys pressed on the keyboard, often without the computer user's knowledge. It is a common method for obtaining sensitive data from a user when the adversary has physical access to the machine or is able to either install or convince the user to install custom software. There are various mechanisms for logging keystrokes, ranging from software based solutions, for example hook based loggers that utilize the operating system functionality to subscribe to keyboard events, to acoustic and electromagnetic loggers that log the pressed keys based on the physical behavior of the keyboard.

Countermeasures against this attack include drivers with signed code, anti-spyware applications able to detect loggers either by their activity or resident files and alternative data input methods like speech-to-text applications or on-screen keyboards.

Key logging is effective against standard username and password combination authentication, but does not, for example, work against one-time-password methods, where a password is rendered obsolete once it is used. Against two-factor authentication methods, like the chip card based approach, key logging does succeed in collecting PIN codes. However, the codes on their own are useless without the ownership of the physical token.

One can circumvent the additional protection added by two-factor authentication, if one knows the PIN code and is able to partially control the machine at the time the chip card is inserted. Large scale attacks of this type, where the adversary has collected several PINs and has control over the user machines, are, however, rather unlikely due to the need for synchronization and in any case much more difficult then simple password logging. Session hijacking, an approach discussed later on, is a variation of an automated attack against chip-card authentication, which does not rely on logging the PINs, but instead utilizing them for chosen operations in real-time when the user is trying to access some of the card's functionality.

2.2 Logging authentication codes: implementation

For implementing a PIN logger, we wrote a simple patch to the OpenSC library. The patch modified the function for forwarding the codes to Micardo cards. When the function was accessed, it wrote down the PIN to a file on the hard drive (Fig. 8).

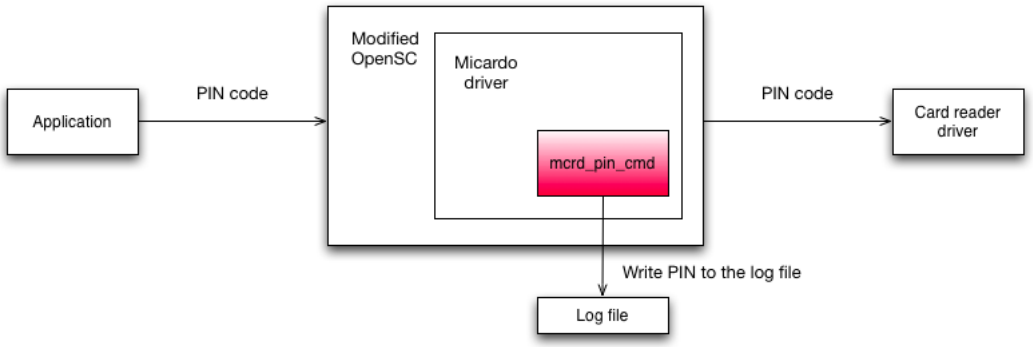

Figure 8: The PIN logging setup

Summary. This modification was trivial to write, once the correct place in the driver code was located and the custom C structures used understood. Both tasks require minimal knowledge of the C language. For setting up the logger up on the client computer, it is necessary to have temporary user- privileged access to either locally compile or transfer a pre-compiled modified version of OpenSC to some chosen location on the machine and point the Firefox browser to use it. This can be done via adding the library directly to the `secmod.db` file stored in the browser user profile folder, adding it via the browser graphical interface or adding it from a webpage using Javascript hooks for handling PKCS#11 libraries.

The exploit enables an attacker to gather chip card PIN codes entered by the users of the infected machines. To guard against this, the browser should secure the `secmod.db` file storing the locations of token libraries by requiring administrative rights or alternative authorization for its modification. However, this might be hard to implement and maintain in practice, as in multi-user machines it often makes sense for users to have different token libraries installed.

2.3 Phishing and substituting certificates

The term phishing describes the process of a malicious entity masquerading as a trustworthy one in electronic communication, in attempt to acquire sensitive information or involve an unsuspecting user in unsafe transactions.

For example, a user may receive an email or an instant message claiming to be from a bank and requesting for some reason to reply with the password to the e-banking service or follow a link to a webpage hosting a fraudulent password entry form. Alternatively, a user browsing the web may, due to network anomalies or webpage vulnerabilities, be redirected to a fake website that visually resembles some legitimate site, but employs different functionality. The general mechanisms of phishing are well explained in [19].

Due to its relative technical simplicity and high success rates, phishing is a popular form of electronic crime. PhishTank [17], one of the major phishing-report collators, reported a monthly average of 6000-8000 websites positively identified as phishing sites in the first months of 2009. Preventive measures to combat phishing include user education, spam filters that filter out phishing emails by general characteristics and publicly

maintained blacklists of servers that often host fraudulent websites. The prevalent reactive approach is issuing site take-down notices based on verified used reports.

One of the solutions browsers provide to remedy the problem is employing public key infrastructure (PKI) to verify the server's identity. PKI refers to a binding between the public key of a host and the host's identity, established by the means of a trusted authority. The authority verifies the authenticity of the binding claim and issues a certificate to confirm it. In practice, an individual or a company interested in obtaining a verified certificate for one's webpage typically generates an appropriate certificate request and forwards it to a commercial authority. The authority then takes the necessary steps to verify that the webpage really belongs to the claimant, charges for the service, and signs the request with its private key. For other parties to be sure of the authenticity of the signature, the authority provides a self-signed certificate issued to its own name. It is obvious, that these self-signed certificates have to be distributed to the users in a secure manner, since if a malicious authority gets to be trusted, all webpages certified by it will be seen as trusted as well. Modern browsers ship with a built-in list of audited popular certificate authorities, so webpages certified by these authorities are trusted by default. Trusted pages are typically identified by a padlock displayed in a dedicated area of the browser's graphical user interface (Fig. 9).

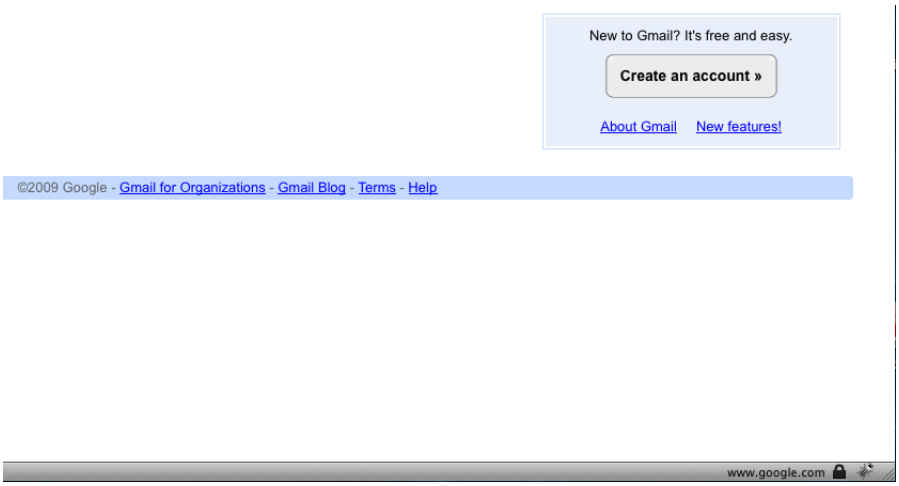

Figure 9: A padlock displayed in the lower right corner of the Firefox browser when connected to a webpage with a trusted certificate

Firefox makes a clear distinction between authority certificates, private keys corresponding to which can be used for signing other certificates, and simple server certificates and has default assumptions about their trustworthiness. If a webpage aiming to establish a secure connection presents a certificate signed by an authority unknown to the browser, the user is shown an option to mark the domain name as a security exception. An exception like this does not, however, grant trust to the certificate authority that had signed the freshly accepted certificate. This means exceptions set this way are valid on per-domain basis only and cannot be used for gaining implicit trust for websites not visited by the user.

Certificates still leave several problems open. First of all, due to organizational hurdles, browsers often contain certificates crafted using encryption or hashes that have been rendered insecure by recent cryptographic research. For example, at the time of

writing, the latest version of Firefox for Mac OSX, Firefox 3.0.10, still had ca. 15% of its about 150 built-in certificates using the MD5 hash function. MD5 was proved not to be collision resistant by 2004 by the latest [20] and the fact was exploited to fake certificate validity in practice in 2008 [21]. Firefox also holds several MD2-based certificates, although MD2 is considered broken as well [25]. Second, as the number of certificate authorities has grown, it has become increasingly easy and cheap to get a domain name certified. There have been reported cases of well-known authorities issuing certificates for arbitrary domain names without proper ownership verification. As a result, a certificate alone cannot serve as an adequate indication of the website's identity.

Extended validation certificates, a concept developed by the certificate authority and browser forum [16], add additional visual cues (Fig. 10) for convincing browser users that the viewed page can be trusted. Before issuing such a certificate, the authority has to take extra steps to verify the trustworthiness of the requesting party. The steps include establishing the legal identity and the operational and physical presence of the website owner, verifying ownership and control status for the domain name in question and confirming the identity and authority of the individual representing the website owner. The list of EV certificate authorities in the browsers cannot be modified using trivial means.

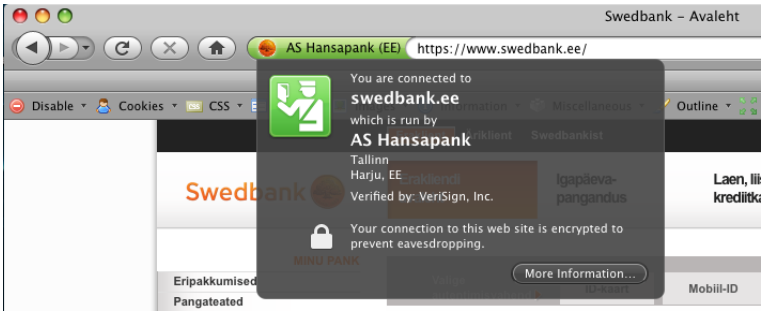

Figure 10: A green address bar and additional information on a webpage with an EV certificate

Studies of the effectiveness of visual notifiers of both common and EV certificates seem to indicate, however, that uneducated users still find it hard to distinguish between legitimate and untrustworthy sites and tend to ignore security warnings. For example, see [18].

In our model, where we assume the adversary to have user-priveleged access to the machine, all browser-related anti-phishing protection can be effectively circumvented. The adversary can, for example, add self-generated certificates for arbitrary domains or untrusted certificate authorities to the browser's safe list, so chosen webpages would appear secure to the user. Alternatively, the adversary can just install a browser extension that changes the graphical user interface of the browser so visual cues corresponding to certified or EV-certified webpages would appear without actual grounds.

In regard to phishing, two-factor authentication methods like chip-card based authentication seem to have an obvious advantage over simple knowledge factor methods like

username and password combinations. By this we mean that since authenticating with a chip-card requires both ownership of the physical chip card and the knowledge of the PIN codes, then even if an adversary learns the PINs, one will not be able to establish an authentication session to a third entity. The best one can achieve is to present a user with fake functionality, which either simply deters the user from accessing some real system or prompts the user to disclose further sensitive data and perform unsafe transactions, for example issuing digital signatures for documents created by the adversary.

Some security experts have argued that two-factor authentication methods are inherently insecure against man-in-the-middle phishing attacks, where the malicious server simply forwards the changing part of the authentication credentials is to the legitimate server in real-time [22]. This may be true for one-time-passwords, but does not hold for chip cards. Indeed, here the changing credential, namely the response to the TLS challenge, depends on the identities of the peers participating in the protocol, as it is a signature over all protocol messages, including the certificate messages of both peers. Now, the adversary does not know the secret keys of the legitimate server and the chip card, so if it forwards the certificate of the legitimate server to the client, one will not be able to decrypt later communication. On the other hand, if the client is presented with a different certificate, one will not be able to respond to the TLS challenge of the legitimate server based on the user's response. This again shows that in the case of chip cards, phishing itself does not allow the adversary to authenticate to a third party using the client's credentials.

2.4 Phishing and substituting certificates: implementation

As a target website, we chose a site of a widely used information system. The particular choice was motivated by several reasons. First of all, the site has acquired its certificate from AS Sertifitseerimiskeskus, an Estonian certificate authority not trusted by default in Firefox. This implies, that for a first-time user, the webpage displays an appropriate warning anyway, prompting to add a security exception for its certificate. Sertifitseerimiskeskus does not issue EV certificates, so there is no visual indication of the connection being secure, except for the standard padlock. Second, the information system offers optional chip-card based log-in, so we were able to play through the phishing scenario with card based authentication.

Note, that as described in an earlier section, phishing alone is not sufficient to mount a man-in-the-middle attack against the chip-card based authentication. Still, it can be successfully used to provide fake functionality, deter the user from accessing the actual system and obtain sensitive data via web-forms. In the case of our information system, the attacker could use the fake website to request the user to update personal details or preferences and prevent the user from accessing the time-critical functionality of the legitimate system.

For the set-up, we generated a certificate chain consisting of three certificates having the same human-readable parameters as in the original chain using the OpenSSL console utility. We then set up a wireless router with a fixed DNS entry for the domain of the information system, pointing to a dedicated server computer, a Ubuntu desk-

top with the Apache 2.2.8 web server extended by mod_ssl 2.28 and OpenSSL 0.9.8g modules (Fig. 11). The server was configured to require a secure connection with optional client authentication accepting chip cards, display a page visually similar to the original and present the fake certificate chain.

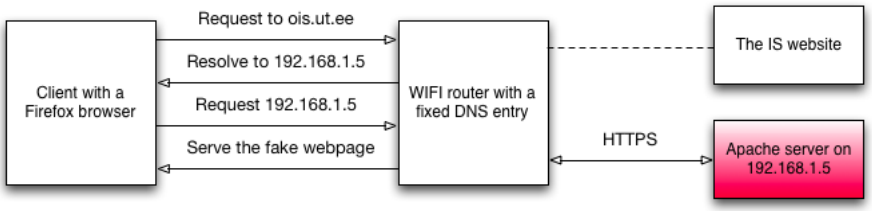

Figure 11: The phishing setup

This situation corresponds to a scenario where the adversary gets hold of the configuration of a public WiFi access point or sets up a rogue access point. In most local switched networks, the situation can also be achieved by successful ARP or DNS attacks that grant a man-in-the-middle status to the attacker. Based on the resulting user view, we believe an incorrectly installed certificate makes it fairly easy to fool even a technically competent user into thinking a connection to be legitimate.

As a side-note, during the course of our work we discovered that the server of the information system was vulnerable to the automated HTTPS cookie hijacking attack [24] and notified its adminstrators.

Summary. The attack requires the experience of generating and manipulating OpenSSL certificates with various parameters, configuring a web server and creating simple webpages. It also requires the ability to lure the user to the set-up page, either by effectively gaining man-in-the-middle status on the local network, poisoning entries of DNS servers or using standard phishing practices like spam and social engineering. For creating an illusion of a secure interaction, the attacker needs user-privileged access for adding certificates to the victim's browser.

Typical methods to combat phishing are described in the previous section. However, even with good user training, it can be difficult to recognize that a fake webpage is being served instead of the original one, when the look-and-feel are almost identical and the browser displays visual security cues. In addition to using standard methods, one can install additional extensions to the Firefox browser, that try to correctly identify websites based on secondary parameters like behavioral differences in different sessions etc.

2.5 Session hijacking

Session hijacking refers to the scenario, where a party attempts to establish a communication session with a certain entity, but an adversary-controlled session under the name of this party is established with some other entity instead, possibly without the victim party's knowledge.

Session hijacking scenarios vary depending on the underlying technologies, in this section we will concentrate specifically on chip-card based authentication.

2.6 Session hijacking: implementation

The general idea of our exploit was to modify the OpenSC driver or specifically the part of it that handles the functionality of Micardo cards to turn to an external script when the challenge response computing operation was called. The script would then initiate a TLS connection to some chosen remote server and switch the challenge bytes sent to the card (Fig. 12).

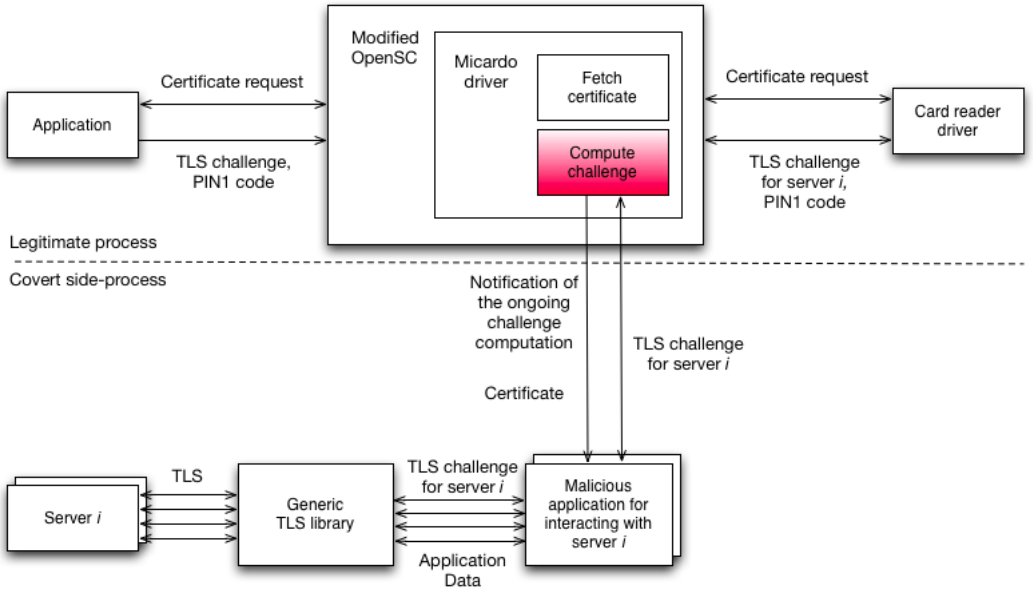

Figure 12: Hijacking the TLS challenge request

The implementation process worked out as follows. We wrote a simple python script that created a HTTP connection over TLS. For handling the TLS protocol, we used a public domain python library `tlslite` [23], which we had to modify in order to accommodate token-based client authentication. It was worth the effort though, as in addition to taking care of the protocol, the library provided our sample script a convenient interface for later requests to the server. We then introduced changes to the part of OpenSC concerned with the challenge response computation of Micardo cards. The changes consisted of bindings using python/C API, so that our script was called once the operation was accessed. The script initialized a connection to a chosen server and returned the bytes needed to be signed to the driver, which took advantage of the user's authorized status to obtain a valid response from the chip card. The script then used this response to complete the authentication to the server.

For testing the implementation, we used the same server software setup as in the previous section. This time, the server was assigned a self-signed certificate and configured to require client-authenticated SSL/TLS on the index page and log all visits. After the setup, we opened Firefox, linked it to the modified OpenSC and tried to access the information system introduced in the phishing section. Indeed, our server

logged a successfully established secure connection and a HTTP-GET request to the index page.

Summary. In theory, the attack requires only superficial understanding of the structure of the OpenSC driver suite and sufficient programming skill to write a script interacting with a webpage via HTTP queries and to bind this script with an appropriate TLS library. In our case, some time was spent on linking the script and OpenSC and on extending `tlslite`. As this was a proof-of-concept implementation, we also made some simplifications: the script would only react to the signing event of Micardo cards and a specific subset of possible cipher suites and fixed TLS protocol behavior was chosen, so that the script would work with our server. These technical restrictions can be overcome by spending more time on testing the TLS library with different servers and testing OpenSC with different chip cards. For deploying the modified library, the same methods can be used as in the case of the logging application. For users to guard against the attack, we again recommend the current situation with the database file storing locations of dynamic token libraries to be reviewed.

All in all, we believe it is moderately easy for an experienced programmer to develop a session- hijacking module for all OpenSC-supported chip cards that can be extended to communicate with chosen servers. This module could work in a covert manner and activate with a certain probability, so that the only indication of its presence would be the once in a while increased latency and connection failures when authenticating online. The latency obviously results from the need to establish a new connection after the browser has contacted the driver. For additional stealth, the script could act depending on the speed of the internet connection and seize its operations after a fixed timeout. In our test case, the internet connection was fast enough for the script to go unnoticed to a typical user.

The shortfalls of our current approach are the need for local compilation and the resulting dependence on the environment. Implementing the session hijacker as a browser extension, as described in the first chapter, would solve these problems, as Firefox aims to be a cross-platform browser. We started developing such an extension, but the task grew rather complex. This came from the fact that Firefox provides many options for extending browser functionality, but not always for modifying the existing one. For example, graphical user interface components and their behavior can be easily replaced or overloaded, but determining whether the next connection shall be mutually authenticated TLS is non-trivial. Nevertheless, such an extension can surely be engineered, for example if it is to target authentication attempts to specific websites based on the URL address. With the current policies for developing browser extensions, there would be no way to guard against the attack but to monitor the list of the activated extensions in the browser. This list is implicitly stored in the user's browser profile folder, so an example solution would be an automated administrative process monitoring this folder.

3 Building a better driver

By now we have shown, that the solution for smart card based authentication employed as now may be vulnerable to certain types of misuse, most notably, session hijacking. In this section, we investigate whether it is possible to change the current software and hardware architecture so the implementation improves with respect to security against the above-mentioned attacks. For this, we first formalize the strongest security we can achieve assuming the operating system core and drivers cannot be compromised. We then discuss the feasibility of changes that bring the architecture closer to the defined security goal.

3.1 Security model

One can view a computer network as consisting of physical entities - network nodes and computers - and logical entities, the users of physical devices (Fig. 13). We assume that each physical entity is used by a single logical entity at a time. If necessary, servers and other multi-user devices can be seen as groups of multiple network nodes. In our model, the adversary is mobile, meaning it is able to dynamically corrupt any number physical entities for arbitrary periods of time. The adversary is also assumed to have full control of the network traffic.

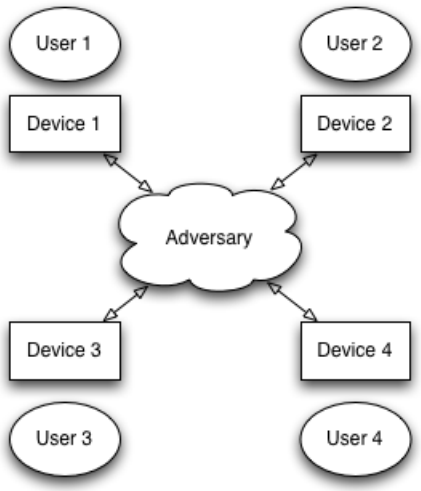

Figure 13: A computer network abstraction

The goal of entity authentication protocols is to establish an association between physical and logical entities. In practice, if a protocol is successful, it is followed by a communication session, as seen in the example of TLS. It is obvious, that if the adversary has compromised a network node, one can modify the contents of this session. However, it is unclear, whether the adversary can still modify the session once one has lost control over the node. Also, it is unclear, whether the adversary can force the user to authenticate to some entity other than the user intended even if one has control of the user's node. This motivates us to say, that the architecture of a device achieves *maximal security with respect to a entity authentication protocol* if the following conditions are met.

- The adversary cannot under any circumstances force a user to authenticate to some other entity unintentionally.
- When the adversary does not actively control either of the physical endpoint nodes, it cannot modify the content of the communication session.

One can see, that due to the possibility of hijacking attacks described in the previous section, the current architecture of the chip card related software stack does not provide maximal achievable security with respect to TLS.

3.2 Suggested solutions

There do indeed exist authentication schemes that provide the security level defined above. Consider a network, where the device hardware is secure and all network cards contain internal secret keys. When establishing an end-to-end connection, the operating systems of these devices specify only the physical address of the communication partner and negotiate a TLS channel using the available infrastructure. The distribution of public keys does not have to be authentic, that is, the adversary can generate key pairs, that claim to belong to some other address. Additionally, every client device has a chip card reader with a pin pad and a display. For client authentication and application data transfer, another TLS channel is created between the chip card and the server. The display shows the user of the client machine excerpts from the TLS protocol, notably the identity of the server and its certificate authority. It is easy to verify that this configuration achieves the maximal achievable security. Indeed, the display makes sure the user does not authenticate to any other party unintentionally. Due to the end-to-end physical communication channel, the adversary looses the ability to modify or listen to the communication session messages on withdrawal. However, such a configuration is surely infeasible because of the organizational costs.

Let us see how we can achieve the security goals by improving on the technologies used today. First, we need to eliminate the possibility of the attacks with a substitute chip card library. This can be achieved by protecting the link between the browser and the library by administrative rights. The second step is to transfer the client-side mutual TLS protocol handling functionality to the driver level. This means giving up the modularity of the software stack and engaging in browser- level protocol based software proxying, but this way the TLS challenge is guaranteed to belong to the correct session, as it is computed by the driver based on its own communication. As the graphical user interface of the operating system can be manipulated by locally installed malicious applications, its authenticity generally cannot be verified, so an appropriate security measure for securing PIN input and assurance of the peer's identity in the TLS protocol could be a physical pin pad reader with a display from the previous example. The display would again show the user the appropriate parts of TLS messages, notably the domain name of the peer and the certificate authority by whom its certificate was signed.

Such changes can be implemented with reasonable means: some developing work on the current software solutions and engineering of a custom pin pad reader. Purchasing the latter would translate into additional costs to a single user, but surely make up in the gained security.

References

[1] The ID-card support site. http://id.ee

[2] ID Süsteemide AS. EstEID Turvakiibi rakenduse kasutusjuhend, 2007. http://id.ee/public/EstEID_kaardi_kasutusjuhend.pdf

[3] ID Süsteemide AS. EstEID Turvakiibi rakendus ja liides, V2.01. http://id.ee/public/EstEID_Spetsifikatsioon_v2.01.pdf

[4] AS Sertifitseerimiskeskus. Sertifikaadid Eesti Vabariigi isikutunnistusel, 2004. http://www.sk.ee/file.php?id=364

[5] Isikut tõendavate dokumentide seadus, RT I 1999, 25, 365. Up-to-date version available at http://www.riigiteataja.ee/ert/act.jsp?id=742623.

[6] Digitaal-allkirja seadus, RT I 2000, 26, 150. Up-to-date-version available at http://www.riigiteataja.ee/ert/act.jsp?id=694375

[7] Sagem Orga GmbH. Micardo 2.1 Chip Card Operating System Manual.

[8] RSA Laboratories. PKCS#11 Cryptographic Token Interface Standard.

[9] ISO/IEC 7816 standard for electronic identification cards with contacts.

[10] The OpenSC smart card driver framework, version 0.11.8, with libopensc 2.0.0. http://www.opensc-project.org/opensc/

[11] The OpenCT smart card reader driver framework, version 0.6.14. http://www.opensc-project.org/openct/

[12] PC/SC Lite http://pcsclite.alioth.debian.org/

[13] RFC 5246. The Transport Layer Security (TLS) Protocol Version 1.2, 2008. http://tools.ietf.org/html/rfc5246

[14] A. O. Freier, P. Karlton, P. C. Kocher. The SSL Protocol Version 3.0, 1996. http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt

[15] Finjan security. How a cybergang operates a network of 1.9 million infected computers. http://www.finjan.com/mcrcblog.aspx?entryid=2237

[16] CA / Browser forum. http://www.cabforum.org

[17] The PhishTank phishing report collator. http://www.phishtank.com

[18] C. Jackson, D.R. Simon, D.S. Tan, A.Barth. An Evaluation of Extended Validation and Picture-in-Picture Phishing Attacks. *In proceedings of the Workshop on Usable Security*, 2007.

[19] T. Moore, R. Clayton. An empirical analysis of the current state of phishing attack and defense, *In Proceedings of the 2007 Workshop on the Economics of Information Security*, 2007.

[20] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, 2004.

[21] A. Sotirov; M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, B. de Weger. MD5 considered harmful today, 2008. Available online at `http://www.win.tue.nl/hashclash/rogue-ca/`

[22] B. Schneier. The failure of two-factor authentication. `http://www.schneier.com/blog/archives/2005/03/the_failure_of.html`

[23] The tlslite python TLS library. `http://trevp.net/tlslite/`

[24] M. Perry. Automated HTTPS cookie hijacking. `http://fscked.org/blog/fully-automated-active-https-cookie-hijacking`

[25] L. R. Knudsen, J. E. Mathiassen. Preimage and collision attacks on MD2, *Lecture notes in computer science*, Volume 3557, 2005.

Kiipkaardipõhise Autentimise Analüüs

Aleksei Gornõi
Bakalaureusetöö (6 EAP)
Kokkuvõte

Enamik kiipkaardiga töötavatest rakendusest eeldab vaikimisi, et liidestus kaardi ajuriga ei ole pealtkuulatav ning et sellele saadetavad andmed pole muudetavad. Selline eeldus pole üldjuhul mõistlik, sest tavakasutajal puuduvad vastava turvataseme tagamiseks vajalikud tehnilised teadmised. Antud lõputöö käsitleb kiipkaartide autentimisfunktsionaalsuse turvalist kasutamist olukorras, kus pahatahtlikul kolmandal osapoolel on olemas ajutine kasutajaõigustega ligipääs kaardi omaniku masinale.

Selleks uurime kõigepealt võimalikke ründeid teoreetiliselt, lähtudes standardsetest lahendustest rakenduste ja operatsioonisüsteemide arhitektuuris. Seejuures loeme igasuguse kiipkaardi-vastase ründe edukaks, kui see ei nõua ründajalt jätkuvat aktiivset osalust ning kui ründe käigus kasutatakse kaarti selle omaniku tahte vastaselt.

Teiseks, implementeerime vastavad ründed konkreetsel platvormil, mis koosneb Ubuntu Linux operatsioonisüsteemist, OpenSC kiipkaardi-ajurist ning veebilehitsejast Mozilla Firefox.

Kolmandaks, määrame, millised muutused mainitud platvormi komponentides tagavad maksimaalse turvalisuse kasutaja-privileegidega ründaja vastu.