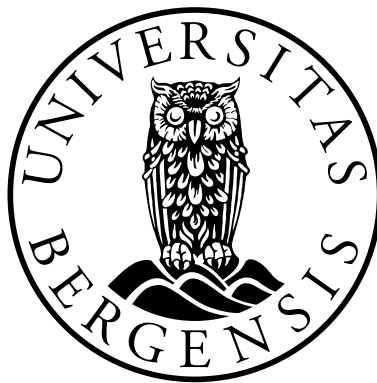


Vulnerabilities in Distributed Computer Systems

Vebjørn Moen
The degree philosophiae doctor (PhD)
University of Bergen, Norway
2006



Universitet i Bergen

Practical Security

Practical Security

Vebjørn Moen

1 Introduction

Distributed computer systems play an important role in our daily lives. Online banking, e-commerce, and e-governments are examples of distributed systems that offer more and more services and, thus, grow larger and more complex every day.

We benefit from all these new services, but they can also affect us in negative ways. Privacy concerns with making large databases available on the Internet and increased risk as e-commerce and online banking replace shops and bank branches have to be addressed by building security in. By nature, computer security is a very broad field. It is used to defend a lot of different data on many different platforms. Human interaction adds to the complexity and for most applications both national and international laws and regulations play an important role.

We can make some general observations about security: Since it is easier to attack a system than to defend it, the attacker has an advantage. To properly defend an asset, one has to defend it against all possible attacks. However, to break a system it is enough to find one exploitable vulnerability, and it does not help if a system is unbreakable at one level, if it is insecure at another level. Often one vulnerability also makes it easier to break the rest of the security of a product. These observations lead to the axiom that security is not stronger than the weakest link.

To make sure that we do not create systems that provide a false sense of security, security experts invest a lot of effort into finding vulnerabilities [1, 2], which is also the main focus of the papers in this thesis.

The next sections are organized as follows: Section 2 gives an introduction to what security is and how to understand the security of a system, Section 3 discusses how security should be managed—including a discussion of what creates vulnerabilities, Section 4 gives an overview of the papers included in this dissertation, and Section 5 concludes the first part of the thesis. The second and main part of the dissertation consists of a collection of papers.

2 What is security?

The main information security elements are *confidentiality*, *integrity* and *availability* [3, p. 5]. Confidentiality ensures that an attacker cannot read the protected information. In a system with integrity, any unauthorized modifications of the content will be detected. The availability goal is to keep services up and running and to make sure that all users can access the services.

Computer security can be defined as the protection of computer systems against actions which violate the desired security elements. To be able to provide computer security, we would like to specify and implement a security policy, defining actions as legal or illegal. Protection against attacks is then reduced to access control, of actions such as read, write, and delete. The access control requires an entity authentication or identification, i.e. corroboration of the identity of a person, a computer terminal, or a credit card [4]. Since some actions can be considered hostile in one computer system, and legal and normal in another, we need to know the details of the application/system before we create the security policy. Therefore, one definition or policy cannot fit every application, and we have no absolute definition of what security is.

Cryptographic algorithms lay the foundation for most of the technologies used to solve computer security problems. A good understanding of cryptology is needed to correctly design and implement security technologies. From the end of the 1970s and up till approximately year 2000 the common view reflected in many books such as [5] and [6], was that *when* we got the cryptography right every computer security problem would be solved.

2.1 Security problems

Attacks on computer systems are exploits of an imperfect specification or implementation of a security policy. We call security vulnerabilities in the design for *flaws*, and vulnerabilities in the implementation for *bugs* [7].

Practical experience during the last few years has showed that the problems with computer security are much broader than the underlying algorithms. Examples include: buffer overflow, Structured Query Language (SQL) injection, phishing, side channel attacks, virus, trojans, and so on. All these security problems that cost the society large amounts of money each year [8], cannot be solved by cryptography alone. The insecurities are caused by flaws within the design and bugs in the implementation, and broken abstractions when ideas and designs travel between different realms. Software applications are made to simplify laborious tasks, and solutions have to solve problems in different fields. The design and implementation have to consider the economic bottom line, the legal aspects, and the viewpoint of the end user. Developing an application that can perform the intended task can be difficult enough, but it gets even more difficult when the whole system has to be able to handle any kind of malicious input.

The engineering of building bridges has a lot in common with computer security. A secure bridge has to be able to handle the wear and tear of normal use, and external influence such as wind and rain. If the bridge is badly engineered it can lead to a bridge failure, e.g. caused by resonance [9]. Therefore, the engineering principles are similar—such as investigate, design and implement good defences against the threats. However, the threats against a computer system are of a different nature. The main difference can be found in the characteristic of the assets, a bridge is a physical object and is only vulnerable to physical attacks, but a computer system consists of both physical and nonphysical assets. Nonphysical assets are vulnerable to a whole new range of attacks, especially systems connected to the Internet, which can be attacked from any computer connected to the Internet. Very few people try to bring down a bridge for fun, but online resources have to sustain daily attacks from malicious hackers who

are trying to break the security for fun or profit.

2.2 Improving security

In cryptology, attacks are considered to be a reasonable way of evolving the security. Full-scale computer security solutions are easier to attack than cryptographic primitives, since they are bigger, more complex, and can be attacked on any of the many layers of abstractions.

The common perception in cryptology is that anyone who are designing their own cryptosystem, should consider all known attacks against such a system, but also try to find new attacks. We argue that the methodology is suitable also for computer security systems.

We study real systems, and the evaluation of such systems often starts with finding vulnerabilities. If you only come up with theoretical results, it is usually difficult to convince the owner of the system that there is a problem, but one simple attack quickly and clearly illustrates the problem in the design or implementation. Looking for attacks is a good starting point both when designing a new system and analysing an existing system.

From Sun-Tzu's *Art of War*: *"If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle."* When building a secure computer system Sun-Tzu's wisdom tells us that without knowing our enemy and the attacks our enemy can launch against us, the attacker will succeed with his attacks on our systems. We need to know and understand what we are defending against. We need to understand the mindset and the toolbox of the attacker, and when we analyze security systems we need to think as the attacker. When we have the knowledge of our attackers, we are more capable to decide which defences we have to spend more resources on.

2.3 Computer security—academic discipline or business secret

Many companies consider secrecy about their system to be an important part of the security. We have seen how the banking industry in Norway keep even their most basic description of security systems confidential, and disclosure of vulnerabilities are regarded as very harmful. Not because of the vulnerabilities, but because of bad press and loss of reputation. *Security by obscurity* can be regarded in different ways. Either the systems are more secure because the owners keep everything secret, or the systems are more insecure. A system where no details are public, can have bugs that nobody knows about. It can therefore run for several years without these bugs being a problem, and therefore the company can save money on not fixing the bugs and flaws. It can also be in production several years with a bug that only a few people know about and are able to misuse. Basing security on secrecy is a flaw in the design, since information will leak about the system over the years. Disgruntled employees can also become malicious attackers.

In addition, there are also other problems when companies refuse to share information about their systems. It creates a closed developer community where there will be fewer new ideas and solutions, since there can be few ideas traded

to and from a closed group. This creates a mode of thought known as group-think. Quality assurance will also be less efficient in a closed group, especially if testing and evaluation are done by the same people who created the system. The solution is probably not to tell the world the details of security routines and implementation details, but having an open discussion about development methods and design with external experts to improve security.

Industry's security by obscurity prevents the academic world from learning about good research problems. Not only does the policy harm research, but also the education of students. When students learn nothing about the common industry problems at the university, the industry is left to educate their own employees. As a result, they tend to maintain a limited and static view of vulnerabilities and security techniques.

We have also seen how problems arise when companies that keep their system secret are allowed to stand in court and claim that their systems are secure. When the court believes such a claim, it is easy to argue that any incident must be caused by wrong behaviour by the users.

3 Measure and manage security

In 1883, Lord Kelvin stated "*I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of Science, whatever the matter may be.*" [10]

In cryptology there are some cryptosystems with a proof of security, e.g. Rabin public-key encryption [4, p. 292], but as Lars Knudsen has said "*If it is provably secure, it is probably not.*" The Rabin public-key cryptosystem is provably secure against one type of attack, but is easily broken if an attacker is allowed to choose ciphertexts and get them decrypted. Proving that cryptographic primitives are secure against every attack is difficult, and provable secure cryptosystems get broken by nullifying the assumptions. When we consider an application, then proving anything about the security is very difficult, and perhaps impossible. Too many assumptions need to be made, and these assumptions are the attack points.

Provable security is probably impossible, but how about rating the security of a product on a scale from 1 to 10? The challenge is to be able to say more about the security of an application other than that the security is broken (proved by a practical attack), the security is untested, or no attacks are found after an analysis. Any measured security of an application will be meaningless if an exploitable flaw or bug is found.

The problem of making provable secure systems are linked to the lack of a good definition of security. For one application we need to decide what we want to be secure against, and defences against one attack might make the application vulnerable to other attacks. One example is the protection against brute-force attacks on online banks described in [11], where the attacked customer accounts are closed down after a small number of unsuccessful login attempts. A protection against brute-force attacks ends up as a vulnerability to a Denial-of-Service (DoS) attack.

Since we cannot measure or prove what a secure application is, we try to say what it is not. Karl Popper discussed the concept of falsification [12], where an expression such as “All swans are white” cannot be proved, because it implies observing every swan in the present, past, and future. However, it is easy to falsify; that only requires finding one swan that is e.g. black. The same idea can be applied to security. Claiming that a program is secure requires analysing it for every possible security bug and flaw, both known and unknown. However, we only need to find one vulnerability to say that a program or system is not secure. Following the argument, a security evaluation of a product cannot look for reasons to conclude that the product is secure. A security analysis must consider possible attacks, both old and new, and how to protect against these attacks.

3.1 What creates vulnerabilities?

According to [7] there is a trinity of trouble in computer software security: complexity, connectivity, and extensibility. The more complex a system is, the more difficult it is to analyze it, and therefore it is more likely that bugs and flaws exist but are missed in a security analysis.

Distributed and connected software are challenging to design and implement properly. The system designer must consider completely new attacks that were not a problem when developing applications for stand-alone computers. The connectivity therefore adds extra complexity to the application.

To incrementally evolve the system functionality, modern applications are often extensible with plug-ins, scripting, and updates. System designers and users like extensible systems since they are dynamic and flexible and can be adapted to do new tasks. Attackers like extensible systems for the same reasons. Extensibility is closely related to connectivity, and the extensibility feature certainly adds to the complexity of the software.

In [13], we show that many e-government projects are vulnerable to simple, well-known Web application attacks, and that industrialized countries have more vulnerable Web pages than under-developed countries. We further show that industrialized countries have more pages created with more types of technologies. The results support the idea of the trinity of trouble in computer software security.

If we buy the argument that complexity, connectivity, and extensibility are the cause of vulnerabilities, what can we do about it? Create less complex, not connected software with no possibility to extend the application? In some cases simpler solutions make sense, but for most applications these features are exactly what are needed to perform the intended tasks.

3.2 Who is the attacker?

As previously mentioned, there exist cryptosystems that are provable secure. Despite the proof of security there exist practical attacks against some of these systems. The question in a threat analysis is not whether the system is secure or not, it is what and who the system is secure against.

Consider a banking system. It is quite clear what such a system has to protect, it is the account holders’ assets. In early banking systems it was also clear who the customers had to trust, namely the local bank that held and

protected their money. There were two kinds of attackers, the bank robbers and the rogue employee, and they were bound by physical limitations. To steal money, the perpetrator had to be in the same location as the money, and if money was missing from the bank vault, and there were no signs of a break-in, then an employee had to be responsible.

There are still two kinds of attackers in modern banking, but now the bank robbers or rogue employees do not have to be in the same place as the money. A break-in to a computer system does not necessarily leave any sign, and it can be facilitated by vulnerabilities in the software.

To defend against insider attacks encryption and authentication keys are created and kept in crypto-modules, and strong cryptography is used to protect transactions. However, [14] shows that the banking industry has a history of phantom withdrawals, break-ins, and documented [15] vulnerabilities in the APIs of the crypto-modules. Thomas Whiteside [16] writes about a supposedly real attack where an embezzler transferred small amounts from many accounts into his own account. There is also a legend of an automated variation of the same attack, where the attacker collected the roundoff of pennies in calculation of e.g. interests, and transferred the money into an account under his control. The history of documented vulnerabilities shows that banks and any owner of a distributed computer system need to have a good understanding of which risks that are worth taking, and which risks that need to be mitigated.

3.3 Mitigating risk

The management of a company is responsible for the bottom line, maximizing income and minimizing expenses. Security is for most companies only a necessary expense, and management will therefore influence how discovered vulnerabilities and risks will be handled. It is bad business to fix a vulnerability if it is cheaper to take the risk of leaving it unfixed, and there are also other methods of mitigating risk than to fix the vulnerabilities. Transferring the risk to different entities through outsourcing, contracts, and terms of use can be effective business “solutions” to difficult computer security problems.

We have seen how online banking systems, even though they save the banks a lot of money on handling accounts, have transferred more risk to the customer. Ross Anderson [17] describes why letting the banks transfer risk to the customer is a bad idea, and in [14] he claims that making the banks responsible for the risk associated new technologies helps promoting security technologies. Risk is an inherent part of every system, and especially distributed software applications. Changes to the system, and introduction of new technology changes the risk. It can also change who have to take the risk. Anderson *et al.* [18] have also shown how the introduction of chip and pin to replace a system of magnetic stripe cards and signatures can be used to transfer more of the risk that banks traditionally have carried over to the customers.

4 Summary of the Thesis

All the papers in this thesis consider security in client-server systems—from cryptologic attacks to design processes and how to develop secure software. Three of the papers discuss banking systems, three papers explore wireless sys-

tems with resource-constrained devices, and one paper consider e-government systems. Five of the papers [11, 19, 20, 21, 13] have been published in refereed journals and conferences, while the last two papers [22, 23] have been submitted for publication.

The main focus of the thesis is analysis of security in distributed computer systems, and mainly on how to attack such systems. However, the papers also suggests how one could fix such vulnerabilities.

4.1 Vulnerabilities in online banks [11]

Online banks have become very popular. The banks save money by letting the customers manage their own transactions, and the customer has the benefit of easy access to updated information about their accounts.

The Norwegian banks tend to claim that Internet banking is absolutely secure if used correctly by the customers. We show that login schemes based on identifiers with a structure and Personal Identification Numbers (PINs) are insecure against a simple brute-force attack. The idea behind the brute-force attack is not to guess the PIN of one customer, but to guess the PIN belonging to a random customer. One example of the attack is to try a few PINs for each customer in a given bank. To have a high probability of success, the attack requires that the bank has more customers than possible PIN values, and that the identifiers are possible to generate. Social Security Numbers (SSNs) and bank account numbers are good examples of such generable values. The attack works even if the online banks use one-time PIN values from lists or PIN calculators if the length of the PINs is small enough.

4.2 Case study: online banking security [19]

We wanted to find out if other solutions were affected by the brute-force attack described in [11]. Several banks use PIN calculators to calculate one-time PINs. In general, PIN calculators are used to create a two-factor authentication scheme, something you have and something you know. A two-factor scheme should be more secure than the static 4 digit PIN solution we analyzed in [11]. However, we found out that because of difficulty in clock synchronizing between the calculator and the server, the server accepts more than one PIN. One of the banks accepted any PIN in a window of 19 consecutive PINs, which drastically increases the efficiency of the brute-force attack compared to a system with a static PIN of same length.

By entering a wrong PIN for an account more than, say, 3 or 4 times, the account will be locked. An attacker can thus combine DoS and brute-force attacks to maximize damage, and increase the likelihood of getting away with the money in the confusion.

Despite claims from Norwegian banks that online banking is completely secure, we found simple and practical brute-force and DoS attacks on several banks. It is likely that the systems had the weaknesses because the banks based their solution on the experience they gained from ATM systems, where these attacks are not practical. We believe that these problems could have been avoided with a proper design phase including evaluation by independent experts.

When trying to inform the banks about the weaknesses, we discovered that they are not interested in improving the security, only in maintaining an im-

pression of secure products. We also learned that the employees of banks are not allowed to discuss security solutions with external experts.

4.3 Lessons from the Norwegian ATM system [22]

Every year around 400 complaints about misused ATM cards are considered by an institution in Norway called “Bankklagenemda”. Many cases are dismissed with the argument that the ATM system is secure and that the victim must have kept the PIN together with the card, or in some other way disclosed it. According to “Bankklagenemda” there are no other explanations for how the PIN was entered correctly by an attacker.

One of these cases went on to court, where the representatives from the bank claimed that their system is secure and that disclosing details about it will make it less secure. The most important claim was that it is impossible to brute-force the PIN, based on the information available on the card, in the period of time from the card was stolen to the card was misused in an ATM. We describe an attack scenario where finding the PIN is done in a few seconds, given that the attackers have carried out pre-calculations to find the bank specific cryptographic key used to calculate the PIN verification value found on the magnetic strip of the card. Today the key used to calculate the PIN verification value is a 3-DES 112-bit key, but until 2000 many banks used 56-bit single-DES keys. The 112-bit keys are too strong, but the 56-bit keys are possible to find using sufficient computational power in a brute-force attack.

The described security issues associated with the ATM system, led us to question how to create stronger systems. We suggest that new and open development processes will improve the security in banking systems, and make it more clear who is responsible for which risk factors.

4.4 Weaknesses in the Temporal Key Hash of WPA [20]

The security scheme Wired Equivalent Privacy (WEP) is a part of the IEEE 802.11 standard for wireless communication. WEP has been shown to be weak [24], and practical attacks exist on all its security features.

To address the weaknesses of WEP, Wi-Fi Protected Access (WPA) was put forward to secure the communication in 802.11 networks. We show that the WPA protocol also has weaknesses, one of them being that it is possible to find the 128-bit master key in WPA with the work equivalent to 2^{105} RC4 encryptions, reducing the effective key size from 128 to 105 bits.

The master key is used to derive per-packet RC4 keys. Assuming we can find some sequential packet keys, we can recover the master key by attacking the key generation algorithm. In essence the attack is an exhaustive search, but we can split the search into separate searches for parts of the master key. Given four RC4 keys the master key can be recovered in 6-7 minutes on an Intel Pentium 4 2.53 GHz with a workload equivalent to $\mathcal{O}(2^{32})$ temporal key hash operations. Given only two RC4 keys the workload is $\mathcal{O}(2^{38})$ and master key recovery takes approximately 15 hours.

4.5 Attack on Sun's MIDP reference implementation of SSL [21]

It seems that every time applications are developed for new platforms, old security problems appear again. This time it is the generation of random data used to create symmetric keys. Just as Goldberg and Wagner [25] showed that the early version of SSL in Netscape was seeded with time, Sun's Mobile Information Device Profile (MIDP) reference implementation of SSL use time as seed to create keys, which drastically reduce the effective key size.

We show how to implement a practical attack against Sun's SSL implementation, which can find the SSL premaster secret based on the transmitted handshake messages. The premaster secret generated by the emulator in Sun Java 2, Micro Edition (J2ME) Wireless Toolkit 2.1 was recovered within one second. However, it is unclear if the attack is also a problem in implementations of MIDP in mobile phones, because we did not recover an SSL key from any of the mobile phones we tried the attack on. We can only speculate how mobile phone manufacturers creates pseudo-random numbers on the devices, since very little information is available.

4.6 Secure networked J2ME applications [23]

The article [23] is based on the experiences we made during the design phase of a commercial application [26] for smart-phones. The application should be able to receive, store and show medical information. In addition, it should run on as many smart-phones as possible. Due to legal considerations, the medical information had to be encrypted during transport, preferably stored encrypted on the phone, and it could only be stored on the server for a short time. Based on market information, we chose J2ME as the development platform. We did not have the desire or resources to implement our own encryption routines, so we targeted MIDP 2.0 devices since they offer HTTPS.

During the project we tested several smart-phones from Nokia, Motorola, Samsung, and SonyEricsson. We were surprised to see that several of the phones had serious bugs affecting how HTTPS connections and certificates were handled, making it impossible to support all MIDP 2.0 phones with MIDP 2.0 compliant applications.

Implementations of MIDP 2.0 contain too many bugs, and do not offer enough security functionality to create applications with the same security as applications for Java 2, Standard Edition. Therefore, we also discuss the forthcoming Security And Trust Services API (SATSA), which will be shipped with future mobile phones, that offers encryption and signing functions that can be used to create even more secure applications for smart-phones.

4.7 Vulnerabilities in e-governments [13]

To get an impression of the maturity of e-governments we checked the Web pages of governments around the world for simple Web application attacks. The attacks we looked at were SQL injection and Cross Site Scripting (XSS). We chose these attacks since they are well known and understood, and relatively easy to defend against.

In [13] we show that more than 80% of the e-governments in the world are vulnerable to these attacks, and that industrialized countries are more vulnerable than under-developed countries.

In addition, the paper also describes some malicious data mining possibilities in the Norwegian e-government. Several Web applications allow an attacker to filter SSNs. Some pages also make it possible to find the name of a person with known or guessed SSN. The information gathered on the e-government pages can be used for identity theft.

5 Conclusion

In this thesis we have explored security in real life systems, and we have discovered how challenging it is for independent researchers in the security field to present results in a constructive and fruitful manner. Disclosed vulnerabilities increase the risk of attacks, and can potentially cost companies a lot of money. Still, if no independent research is done on the security of real life systems the knowledge of the systems' security is completely controlled by companies. Since our work is funded by tax payers, we feel an obligation to evaluate national systems.

Based on how vulnerabilities have been managed in Norwegian banking systems, we believe that independent research can balance the knowledge between customers and banks, and help customers to take more educated decisions and avoid becoming victims of the banks' security-by-obscurity policy.

In [22], we looked at a court case where a Norwegian bank was sued by a customer because of stolen and misused credit cards. In the court, the bank claimed that it could not disclose any details about their system, since that would make the system more insecure. However, they were prepared to put some of their top people on the stand to testify that their system is secure. We claim that this security-by-obscurity attitude does not make bank systems more secure, and that it is harmful to the customers. It is not unlikely that similar cases to the one described in [22] will appear for online bank systems in the future.

Perfect security does not exist, and it is important to consider the risks we take when we develop large and complex systems for new platforms, e.g. wireless thin clients for banking systems. Without a proper infrastructure for identification and authentication, supporting all platforms, online banking and e-government services will continue to be vulnerable to simple attacks against the authentication schemes. The solutions to the computer security problems are not only technical, since security solutions are also influenced by laws, regulations, psychology and human-computer interaction. Perhaps we will witness a paradigm shift in computer security, where researchers have to focus on economy and law in addition to computers and algorithms.

References

- [1] E. Rescorla, "Is finding security holes a good idea?" *IEEE Security & Privacy*, vol. 3, no. 1, pp. 14–19, 2005.

- [2] D. Farmer and W. Z. Venema, "Improving the security of your site by breaking into it," Posted to Usenet, 1993.
- [3] D. Gollmann, *Computer Security*. John Wiley and Sons Ltd, 1998.
- [4] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.
- [5] S. Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Doubleday, 1999.
- [6] B. Schneier, *Applied Cryptography, Second Edition*. John Wiley & Sons, 1996.
- [7] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*. Addison-Wesley, 2004.
- [8] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson, "2005 CSI/FBI computer crime and security survey," 2005. [Online]. Available: <http://www.gocsi.com/>
- [9] K. Billah and R. Scanlan, "Resonance, Tacoma Narrows bridge failure, and undergraduate physics textbooks," *American Journal of Physics*, vol. 59, no. 2, pp. 118–124, 1991.
- [10] W. Thomson, *Popular Lectures and Addresses Vol. 1*. MacMillan, 1891.
- [11] T. Tjøstheim and V. Moen, "Vulnerabilities in online banks," in *10th Nordic Workshop on Secure IT-systems (Nordsec 2005)*, 2005.
- [12] K. Popper, *The Logic of Scientific Discovery*. Basic Books, 1959.
- [13] V. Moen, A. N. Klingsheim, K. I. F. Simonsen, and K. J. Hole, "Vulnerabilities in e-governments," in *2nd International Conference on Global E-Security (ICGeS-06)*, 2006.
- [14] R. Anderson, "Why cryptosystems fail, from communications of the ACM, november, 1994," in *William Stallings, Practical Cryptography for Data Internetworks*. IEEE Computer Society Press, 1996. [Online]. Available: citeseer.ist.psu.edu/anderson94why.html
- [15] M. Bond and R. Anderson, "API-level attacks on embedded systems," *Computer*, vol. 34, no. 10, pp. 67–75, 2001. [Online]. Available: citeseer.ist.psu.edu/bond01apilevel.html
- [16] T. Whiteside, *Computer Capers: Tales of Electronic Thievery, Embezzlement, and Fraud*. Ty Crowell Co, 1978.
- [17] R. Anderson, "Why information security is hard - an economic perspective," 2001. [Online]. Available: citeseer.ist.psu.edu/anderson01why.html
- [18] R. Anderson, M. Bond, and S. J. Murdoch, "Chip and spin," 2006. [Online]. Available: www.chipandspin.co.uk/
- [19] K. J. Hole, V. Moen, and T. Tjøstheim, "Case study: Online banking security." *IEEE Security & Privacy*, vol. 4, no. 2, pp. 14–20, 2006.

- [20] V. Moen, H. Raddum, and K. J. Hole, "Weaknesses in the temporal key hash of WPA," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, no. 2, pp. 76–83, 2004.
- [21] K. I. F. Simonsen, V. Moen, and K. J. Hole, "Attack on sun's MIDP reference implementation of SSL," in *10th Nordic Workshop on Secure IT-systems (Nordsec 2005)*, 2005.
- [22] K. J. Hole, V. Moen, and A. N. Klingsheim, "Lessons from the Norwegian ATM system," 2006, submitted to IEEE Security & Privacy.
- [23] A. N. Klingsheim, V. Moen, and K. J. Hole, "Secure networked J2ME applications," 2006, submitted to IEEE Computer.
- [24] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting mobile communications: The insecurity of 802.11," in *MOBICOM 2001*, 2001.
- [25] I. Goldberg and D. Wagner, "Randomness and the Netscape browser," *Dr. Dobbs's Journal*, pp. 66–70, January 1996.
- [26] W. M. Center. [Online]. Available: <http://www.world-medical-center/>

Paper I: Vulnerabilities in Online Banks

Vulnerabilities in Online Banks

Thomas Tjøstheim and Vebjørn Moen

Abstract

This paper describes some attacks on online banks that authenticate each customer through the use of a unique user identifier and a Personal Identification Number (PIN). Many user identifiers contain structure which make them easy to generate on a computer. Given a generated set of identifiers it is possible to do a brute-force attack on the PINs. A general attack model is described and some example attacks against a Scandinavian online bank are discussed.

1 Introduction

Online banks have thrived with the explosive growth and availability of the Internet. A wide variety of services are offered to the customers. Paying a bill, checking the account balance, or applying for a loan can now be comfortably done from one's own home or office. However, the new possibilities introduced with Internet banking have also resulted in new security challenges. It is difficult to create both user friendly and secure Internet banking solutions. Can customers really trust that an attacker will not be able to break into their accounts?

Online banks claim that they are secure as they have many security features like firewalls, Public Key Infrastructure (PKI), Intrusion Detection Systems (IDSs), money auditing systems, and Secure Socket Layer (SSL). The average customer seems to be satisfied with the level of security in Internet banks. However, security is complex and not some magic potion that you add to your system to make it secure. Security should be considered from the start of the system development phase. A careful analysis of the environment is necessary to determine the needed security services and to determine how to implement these services correctly. Analysis of a security protocol is very difficult, due to the many ways an attacker can take advantage of the protocol environment.

In this paper we show that online banks authenticating each customer through an N -digit PIN in combination with a structured user identifier are vulnerable to both brute force and Denial of Service (DoS) attacks. There are at least three online banks in Norway that use or have used this form of customer authentication. However, the authors have decided not to explicitly name any banks, as this has been requested from one of the banks, and the fact that some of the banks are still vulnerable to the attacks described in this paper.

The rest of this paper is organized as follows: Section 2 presents the general attack model, Section 3 describes structure and generation strategies when Social Security Numbers (SSNs) and account numbers are used as unique identifiers, Section 4 discusses some example attacks on a real Internet bank in Scandinavia, and Section 5 concludes the paper.

2 Attack model

A common way of authenticating customers in online banks is to require a unique identifier for each customer together with a secret that only the customer knows. This section describes a general attack model against online banks where each customer has a unique user ID and an N -digit PIN as their secret. The PIN can be either static or dynamic. A static PIN stays the same while a dynamic PIN is changed for each login; it can for example be generated by a PIN calculator.

A customer gains access to an account only by entering a valid user ID and PIN combination. Typically, there will be a limit on how many times (often three or five) a wrong PIN can be entered for a given customer's user ID. The objective of this limit is to prevent a brute-force attack against the customer's PIN. A customer will temporarily lose access to the account if the limit is exceeded and must contact the bank in order to receive a new PIN.

2.1 Brute-force attack

Figure 1 depicts the attack model for the brute-force attack. The generated set of user IDs will contain a subset of the user IDs belonging to the customers of an online bank (we will see examples of how to generate such sets in Section 3). A program logs into the online bank's web pages and automatically enters different user ID and PIN pairs. One can observe that it does not matter if the PIN is dynamic or not. The only difference is that if a PIN is dynamic the attacker would have to attack the account at the moment a valid PIN and user ID combination is found.

Running the attack from only one host is not realistic, as this most likely will be detected by the bank's IDS. A distributed attack could be done by for example spreading a virus that contains the brute-force program in addition to having a control program that gets feedback from the zombie machines. Furthermore, it is possible to avoid the IDS by spreading the attack over several days in order to hide the number of tried logins in the anticipated natural traffic from legitimate users accessing the bank.

The probability of accessing at least one account is

$$\begin{aligned} P(\text{At least one}) &= 1 - P(\text{none}) \\ &= 1 - (1 - P_{\text{success}})^Y \end{aligned} \quad (1)$$

where Y is the number of online bank customers in the generated set of user IDs. The anticipated number of cracks is

$$\mu = Y \times P_{\text{success}}. \quad (2)$$

The probability that an attacker cracks a random customer's account is

$$P_{\text{success}} = \frac{X}{10^N}. \quad (3)$$

Here, X is the maximum number of allowed wrong entries for a PIN, and N is the number of digits in the PIN.

Given a generated user ID, an attacker must first consider whether it is valid and belongs to an online bank user, or not. An attacker would ideally like to maximize the amount of customer user IDs in the initially generated set. This would give the most effective and *silent* attack.

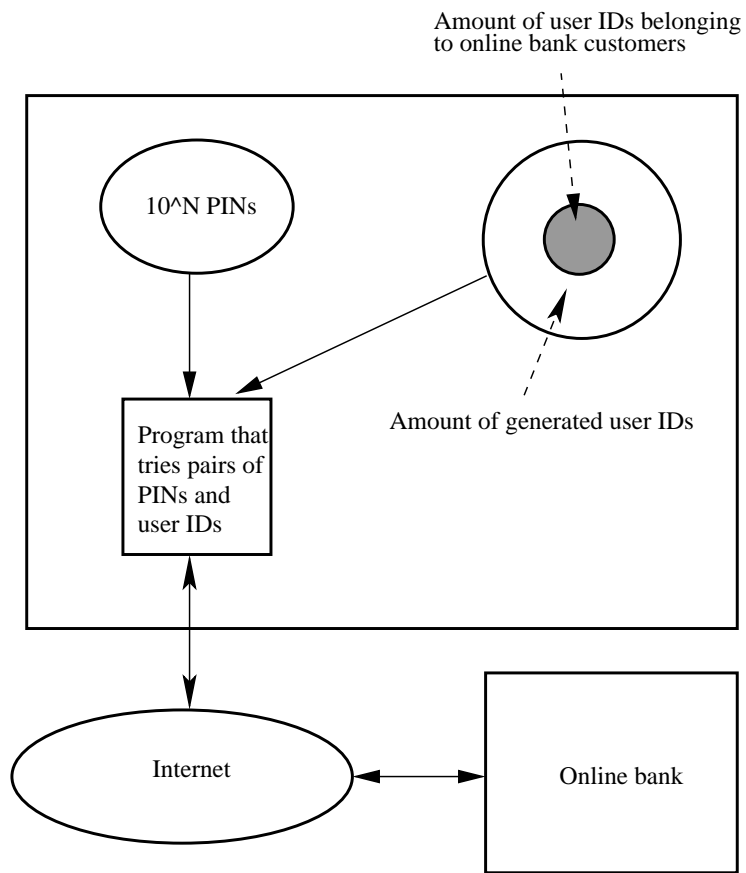


Figure 1: Attack model

2.2 DoS attack

If an attacker can acquire *many* user IDs, there is also a possibility of doing a distributed DoS attack against the bank's customers. The login scheme of online banks simplifies an application layer based DoS attack. An attacker can temporarily shut down access to accounts by entering X incorrect PINs for each valid user ID.

2.3 Combined DoS and brute-force attack

The probability in (3) assumes a combined DoS and brute-force attack. It can be discussed if the attacker is better off guessing $X - 1$ times since customers are not denied access this way, and it might take longer before the bank's IDS detects the attack. However, if the attacker controls a network of "zombie" machines, it is very difficult to both identify the attacker and stop the attack from all the machines in the controlled network. The chaos created by the combined attack could also be an advantage for the attacker. For more information on how to execute a distributed DoS attack please consult [1].

3 Generating user IDs

This section describes two real cases of user IDs being used in online banks. We will study structure and generation strategies for Norwegian SSNs and account numbers. The arguments that apply to the Norwegian user IDs are similar for other countries. In particular, we have verified that the same generation strategies, with minor modifications, can be applied to both Swedish and Danish user IDs.

3.1 Norwegian SSNs

Norwegian SSNs are not confidential. Given a reasonable documented need, the SSNs can be requested from a national register. Many public institutions like hospitals, banks, and tax authorities have legal access to people's SSNs, but it can be difficult for private persons to argue the need for many SSNs. Therefore it might be better for an attacker to generate a set of SSNs. Most SSNs have a specific structure which make them easy to generate.

3.1.1 Structure of Norwegian SSNs

The Norwegian SSN [2] consists of 11 digits: $x_1x_2x_3x_4x_5x_6i_1i_2i_3c_1c_2$.

$x_1x_2x_3x_4x_5x_6$ is the birth date of the individual on the form *ddmmyy*.

$i_1i_2i_3$ is called the individual number and is used to separate people born on the same date. The national register distributes SSNs in the order they receive birth messages. They start with the highest available valid individual number for that day and proceeds downwards for each new birth message. The individual number is based on the century the person is born in, as shown in Table 1. It is also possible to distinguish boys from

girls by looking at i_3 , which is odd for boys and even for girls.

c_1c_2 are control digits that are calculated as weighted sums of the first 9 and 10 digits, respectively.

$$\begin{aligned} c_1 &= 11 - (3x_1 + 7x_2 + 6x_3 + x_4 + 8x_5 + 9x_6 + 4i_1 + 5i_2 + 2i_3 \pmod{11}) \\ c_2 &= 11 - (5x_1 + 4x_2 + 3x_3 + 2x_4 + 7x_5 + 6x_6 + 5i_1 + 4i_2 + 3i_3 + 2c_1 \pmod{11}) \end{aligned}$$

If either c_1 or c_2 is calculated to be $10 \pmod{11}$ the SSN is discarded, and if c_1 or c_2 is equal to 11 then it is set equal to 0. Let's assume that c_1 and c_2 are approximately independent, then an SSN will be discarded with the following probability:¹

$$\begin{aligned} p(c_1 \cup c_2) &= p(c_1) + p(c_2) - p(c_1 \cap c_2) \\ &= \frac{1}{11} + \frac{1}{11} - \left(\frac{1}{11}\right)^2 = \frac{21}{121}. \end{aligned}$$

Individual number	Year in birth date	Born
500–749	>54	1855–1899
000–499		1900–1999
500–999	<55	2000–2054

Table 1: Correspondence between individual number and birth date

3.1.2 SSN generation strategies

How can an attacker maximize the ratio of customer SSNs in the initially generated set? Four different strategies will be discussed.

Strategy 1: The simplest strategy is to generate SSNs in such a way that all of the online bank's customers are covered. If for example the online bank only has customers in the age group 18–100, one could generate all possible SSNs for this group. With this scenario, all customers are born in the 20th century and are therefore given individual numbers in the range 000–499. Hence, for each day in the year we get 500 possible SSNs, but an estimate of 21/121 will be invalid numbers. The number of possible SSNs for people that are 18–100 is $500 \times 365 \times 83 \times (100/121) \approx 12.5$ million. Let Z denote the total number of customers in an online bank. The proportion of customer SSNs would then be

$$R_{customers} = \frac{Z}{12.5 \text{ million}}. \quad (4)$$

The drawback, from the attacker's point of view, is of course the huge number of SSNs that has to be checked. A large portion of the SSNs will neither belong to *real* people nor to online bank users. There is also a high probability for the bank's IDS detecting the attack because of the big workload.

¹To control the assumption of stochastic independence, a computer program was written that generated all the possible SSNs for the 20th century and counted the number of discarded SSNs. The results from the computer program gave the probability estimate 21/121 down to the 5th decimal.

Strategy 2: A strategy for increasing the concentration of customer SSNs is to focus on a specific age group that has a high percentage of online bank customers. For example, 34 % of customers in pure online banks are males in the age group 26–35 [3]. The number of generated SSNs for this particular group would be $250 \times 365 \times 10 \times (100/121) \approx 754,132$. The ratio of customer SSNs would then be

$$R_{customers} = \frac{Z \times 0.34}{754,132}. \quad (5)$$

Strategy 3: However, one still has to generate a lot of SSNs belonging to non-existing people. This problem can be avoided by taking advantage of the chronological assignment of SSNs to newborn and immigrants. Instead of generating all valid SSNs for one day, it is possible to use population statistics to reduce the amount of generated SSNs. As an example, one can look at the period corresponding to males in the age group 26–35. There is an average of about 33,343 SSNs assigned per year for this group [4]. This gives an average of $33,343/365 \approx 91.4$ people per day. Let S be the number of assigned SSNs for a particular day. This number will of course vary from day to day. To get an idea of how S varies one can make the simplifying, but only approximately true assumption, that each day in the year is equally probable for the assignment of an SSN. This gives a binomial probability distribution with $n = 33,343$ (the average for a year) and the probability $p = 1/365$ (ignoring leap years) that a person is assigned an SSN for a particular day. The standard deviation for a random variable V having a binomial distribution is

$$Sd(V) = \sqrt{np(1-p)}. \quad (6)$$

From (6) we have $Sd(S) \approx 9.5$. To get an estimate of how S varies for each day one can construct an interval with ± 3 standard deviations. The probability that S lies in the interval is

$$\begin{aligned} P(91.4 - 3Sd(X) \leq S \leq 91.4 + 3Sd(X)) \\ \approx P(62 \leq S \leq 120) = 0.9974, \end{aligned}$$

since a binomial distribution can be approximated with a normal distribution.

If the attacker generates 120 SSNs for each day, then:

$$P(120 \leq S \leq 120 + m)$$

is the probability that the attacker loses between 0 and m SSNs for that day. This probability is limited to ≤ 0.0013 since ± 3 standard deviations are used. The number of generated SSNs would be $120 \times 365 \times 10 = 438,000$. We can assume that almost all of the online bank's customers in the age group 26–35 are covered. An approximated ratio of customer SSNs is then:

$$R_{customers} = \frac{Z \times 0.34}{438,000}. \quad (7)$$

SSNs that do not belong to *real* persons can be minimized if the attacker for example generates 62 SSNs for each day. This way the attacker will lose some SSNs belonging to real people, but will with probability $P(S \leq 62 - m)$ which is ≤ 0.0013 generate between 0 and m too many SSNs for a particular day.

The number of generated SSNs would then be $62 \times 365 \times 10 = 226,300$. This will correspond to approximately generating 0.68 ($226,300/333,430$) of the total number of SSNs for the age group 26–35. If we assume a uniform distribution of online customers among the assigned SSNs, an estimate of the customer ratio is then:

$$R_{customers} = \frac{Z \times 0.34 \times 0.68}{226,300}. \quad (8)$$

Strategy 4: Another possibility is to filter out the SSNs belonging to online bank customers by trying to exploit response information from the bank’s web pages. Two filtering examples are shown in Section 4.2.

3.2 Norwegian account numbers

An account number is a unique identifier for a customer’s account, and can be generated in much the same way as an SSN. This is not hard when the structure is known.

3.2.1 Structure of Norwegian account numbers

A Norwegian account number [5] consists of 11 digits: $b_1b_2b_3b_4a_1a_2a_3a_4a_5a_6c_1$. $b_1b_2b_3b_4$ indicates which bank the account belongs to. Each bank has a set of serial numbers that identify the particular bank.

a_1a_2 is the type of account, e.g. a salary account or a high interest account. There is no standard for which numbers have to be used, each bank can define its own system.

$a_3a_4a_5a_6$ are digits that uniquely identify a customer’s account. When a new account is created the smallest available 4-digit number is chosen.

c_1 is a control digit that is calculated as a weighted sum of the first 10 digits:

$$c_1 = 11 - (5b_1 + 4b_2 + 3b_3 + 2b_4 + 7a_1 + 6a_2 + 5a_3 + 4a_4 + 3a_5 + 2a_6) \pmod{11}$$

However, if c_1 is calculated to be $10 \pmod{11}$, the account number is discarded.

3.2.2 Account number generation strategies

The strategies for generating account numbers are easier than for SSNs. An attacker can find out which serial and account type numbers a particular bank uses. Given one of the bank’s serial numbers and an account type, an attacker can generate the next four digits $a_3a_4a_5a_6$ in such a way that it gives a valid account number. Note that there are only $10,000 \times 10/11 \approx 9,090$ valid combinations.

An attacker can also take advantage of the fact that the smallest available account number is always chosen. It is also likely that an attacker can filter out valid account numbers by guessing incorrectly X times for a given account number and observing the response. Given this, it is possible to generate an interval of account numbers that the attacker knows belongs to customers.

4 Example attacks on a real Internet bank

Let Bank B denote the Norwegian branch of a Scandinavian bank that specializes in online banking services. The security solution was changed in 2004. In this section we will look at some theoretical example attacks on the Norwegian bank B, both before and after the change of security solution.

4.1 Bank B before 2004

A customer in bank B is supposedly authenticated by having a valid SSN, a $N = 4$ digit PIN, and a *personal* certificate. A new certificate must be downloaded for each new host used to connect to the bank. Before 2004 a customer downloaded a new certificate by entering a valid PIN and SSN pair. With this scenario an attacker could try to brute force the PIN by attempting to download a new certificate. An attacker had ($X = 3$) tries at guessing the correct PIN.

4.2 Brute-force attack

How does the brute-force attack described in Section 2.1 apply to bank B? Given the first strategy in Section 3.1.2, all SSNs for people in the age group 18–100 are generated. It is realistic to assume that nearly all of B's customers are covered in this SSN generation. In Norway, bank B had more than $Y = 220,000$ customers in 2003. From (1), the following probability can then be obtained

$$P(\text{At least one crack}) = 1 - \left(1 - \frac{3}{10^4}\right)^{220,000} \approx 1,$$

and from (2), the anticipated number of cracks are

$$\mu = 220,000 \times \frac{3}{10^4} = 66.$$

The ratio of customer SSNs is from (4) $220,000/12.5$ million ≈ 0.018 .

The second strategy was to only generate SSNs belonging to males in the age group 26–35. The expected number of B's customers in this age group is $Y = 220,000 \times 0.34 = 74,800$. This gives the following probability:

$$P(\text{At least one crack}) = 1 - \left(1 - \frac{3}{10^4}\right)^{74,800} \approx 1.$$

The anticipated number of cracks when checking all SSNs one time is

$$\mu = 74,800 \times \frac{3}{10^4} \approx 22.$$

From (5), the ratio of customer SSNs is $74,800/754,132 \approx 0.099$.

The third strategy was to exploit the chronological ordering of SSNs combined with the use of population statistics. We apply the strategy to both the age group of 26–35 and 18–100. Table 2 shows some different results when the average number of SSNs ± 3 standard deviations is generated per day for the two groups. In particular, the table lists the total number of SSNs generated and the approximated number of B's customers covered.

In Section 3.1.2 a binomial probability distribution was assumed, and the two cases of generating 120 SSNs and 62 SSNs each day for the group 26–35 were

Strategy 3						
Variation	SSNs per day	Age	Total SSNs	\approx B SSNs	Ratio SSNs	Cracks
1	148	18–100	4,483,660	220,000	0.049	66
2	84	18–100	2,544,780	160,180	0.063	48
3	120	26–35	438,000	74,800	0.171	22
4	62	26–35	226,300	50,864	0.225	15

Table 2: Overview of results for strategy 3, when the average number of SSNs ± 3 standard deviations is generated each day for the two age groups

considered. In the first case one can expect to almost cover all of the 74,800 customers in B and obtain about the same probabilities as when we generated all the valid SSNs for the same age group. The ratio (7) of customer SSNs would be $74,800/438,000 \approx 0.171$. With 62 SSNs generated each day the following estimate of the ratio of B SSNs is obtained from (8) to be $50,864/226,300 \approx 0.225$.

The birth statistics [4] for men and women in the age group 18–100 show that there is a total of 3,495,131 people (SSNs) and this gives an average of $3,495,131/83 \times 365 \approx 115.4$ per day. The standard deviation is calculated from (6) to be ≈ 10.7 .

From Table 2 we see that the anticipated number of accounts cracked is dependent on the number of SSNs generated. There are different attack variations depending on how the bank will react to the attack. For instance, the most effective attack would be to try and verify the 226,300 SSNs generated with variation 4 in Table 2. Depending on how B would react to the first attack, the same attack could be repeated with about 15 anticipated cracks each time. On the other hand, if the attacker only gets one chance at verifying the SSNs and the number of SSNs does not matter, then variation 1 in Table 2 yields the best attack.

The fourth strategy was to try to filter out the SSNs belonging to online bank customers. Two different approaches were discovered for the case of bank B:

1. When a valid SSN is combined with a wrong PIN the following error message is returned: "You have entered the wrong SSN or PIN." After three incorrectly entered PINs, and only if this is an SSN belonging to a customer, will the bank respond that the customer has been denied access to the bank. This means that an attacker can guess three times for each SSN, and not only find valid PIN and SSN combinations, but also filter out which of the SSNs belong to B's customers.
2. It is also possible for an attacker to filter an SSN by trying to register a new customer. When registering, bank B only verifies the correspondence between the SSN and the name. Fake email, phone numbers and so on can be entered. Only when entering an SSN that belongs to a customer will a specific error report be sent: "There was an error with registration. Please contact. . ." Otherwise the person with this SSN will be registered as a new customer. A disadvantage is that an attacker will register a *large* amount of new customers and this will probably be detected. The

advantage of this method compared to the first is that an attacker can filter the SSNs before executing the brute-force attack.

4.3 Bank B in 2004

The certificate downloading scheme in bank B was altered in 2004. In addition to entering a valid PIN and SSN combination, a customer must also enter a valid *one-time password* that is sent either to the customers' mobile phone or mailbox. If the password is sent as an SMS it has 15 minutes validity, while a password in the regular mail is valid for 14 days.

The brute-force attack described in Section 4.1 is still possible. An attacker that finds valid PIN and SSN combinations can decide how the one-time password shall be delivered. If the password is delivered by SMS, the attacker could try to *sniff* the password with an interceptor [6]. However, it is much easier and cheaper for the attacker to get the password from the mailbox. Given a valid SSN it was possible to find the matching name and the address. This could for instance be done by using a Norwegian pension fund web site [7]. This site authenticates members using only SSNs. When a member logs in, the site displays the name and address corresponding to the SSN. If the attacker chooses password delivery by mail, he decides when the password shall be delivered, and will have a good indication of when to steal the mail.

5 Conclusions

This paper shows that online banks authenticating customers through the use of a PIN in combination with an SSN or an account number are vulnerable to both brute-force and DoS attacks at the application level. The degree of exposure to brute-force attacks depends on the number of digits in the PIN. Whether the PIN is static or is changed for each login makes no difference in this case.

In Section 4 it was shown that bank B is vulnerable to a brute-force attack as the PIN only has 4 digits. The PKI solution in bank B is of limited value, since a new certificate and corresponding private key can be downloaded given a valid PIN and SSN combination and the one-time password.

An easy countermeasure against the attacks described in this paper would be to use user IDs that do not contain any structure, so that they would be difficult to generate automatically. The reason the banks have not done this, might be that it simplifies customer handling to use SSNs or account numbers as unique customer identifiers. Another suggestion is a fully functional PKI solution that require customers to meet in person with the Registration Authorities (RAs) when opening an account. This would enable stronger user authentication and possibly solve many of the vulnerabilities in online banks.

Much of the security in online banks relies on IDSs and money auditing systems. The problem with this approach is that it deals more with detection than attack prevention. A combined brute-force and DoS attack is hard to protect against with the current security schemes. The online banks have little other protection than temporarily closing down service for customers. The potential damage to the bank's reputation and the loss in revenue could be substantial.

References

- [1] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service, Attack and Defense Mechanisms*. Pearson Education, 2005.
- [2] E. Selmer, "Personnummerering i Norge: Litt anvendt tallteori og psykologi," *Nordisk matematisk tidskrift*, 1964, (in Norwegian).
- [3] M. Hjorth, "En kartlegging av nettb@nkkunders holdninger," Master's thesis, University of Bergen, 2002, (in Norwegian).
- [4] Statistics Norway, "Statistikkbanken," last visited: August 29th 2005. [Online]. Available: <http://statbank.ssb.no/statistikkbanken/>
- [5] European Committee for Banking Standards, "Norway - domestic account number," 2003. [Online]. Available: <http://www.ecbs.org/Download/TR201/norway.pdf>
- [6] Spylife.com, "Cellular telephone interceptor," last visited: August 29th 2005. [Online]. Available: http://www.spylife.com/digital_interceptor.html
- [7] S. Pensjonskasse, "Elektronisk lånesøknad," last visited: August 30th 2005. [Online]. Available: <http://www.pensjonskassa.no/elaan/Elaan>

Paper II: Case Study: Online Banking Security

Case Study: Online Banking Security

Kjell Jørgen Hole, Vebjørn Moen, and Thomas Tjøstheim

Abstract

This article argues that Norwegian Internet banking was vulnerable to computer attack in the period 2003–2004 by describing scenarios for potential attacks.

1 Introduction

Many Norwegians turned to the Internet for personal banking services during 2003 and 2004. Because of the Norwegian banks' security-by-obscurity policy, the online customers knew very little about the true level of security offered by the Internet banking systems; the banks naturally maintained that their systems were very secure.

In this article, we discuss simple—but powerful—strategies for attack on selected Norwegian online banks, and discuss why these attacks were theoretically possible during much of 2003 and 2004. The scenarios are based only on publicly available information found on the Internet. None of the attacks were carried out in practice. Instead we presented our findings to the Norwegian government agency overseeing the national banking industry. We also made a sustained effort to directly inform the banks most vulnerable to the attacks.

Our main reason for making public this account is to contribute to the development of more secure Internet banking systems. For the same reason, we also speculate why the banks developed Internet banking solutions with bad security in the first place. Finally, we suggest ways in which universities can teach students to design more secure alternatives.

Many Norwegian Internet banks have long required their customers to log into online accounts using a Social Security Number (SSN) or an account number, as well as a Personal Identification Number (PIN). While the SSN and the account number aren't considered secrets, the PIN is only known to the customer. If a customer tries to log into an account with the correct SSN (or account number) and a wrong PIN enough times (usually between 3 and 5), the bank will temporarily deny the customer any further access to the account.

As we'll see, it was possible for a cracker to launch an attack against an Internet bank by combining a simple brute-force attack with a Distributed-Denial-of-Service (DDoS) attack [1] that exploits the bank's login procedure. A successful combined attack gains access to a small number of accounts and—at the same time—prevents a large number of legitimate customers from accessing their accounts. We'll show that the combined attack is also effective when the bank's customers use PIN calculators, also called PIN cards or (hardware) tokens, to generate dynamic PINs.

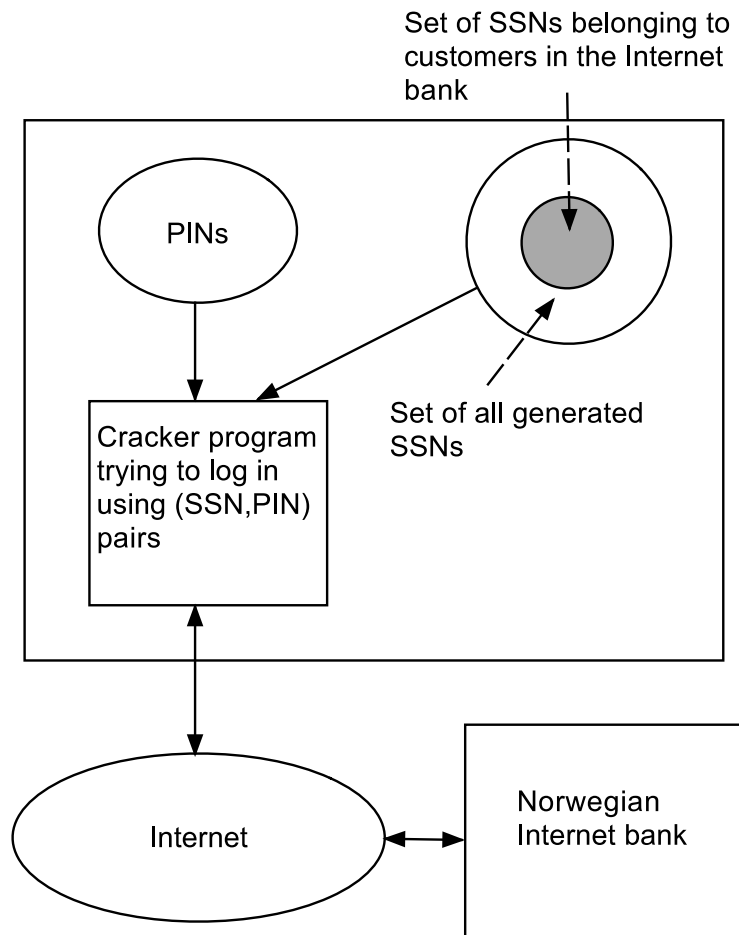


Figure 1: Model of a simple brute-force attack on a Norwegian Internet bank.

2 A simple attack

A possible brute-force attack against a Norwegian Internet bank in 2003 and 2004 is illustrated in Fig. 1. For simplicity, we consider only an attack utilizing SSNs. The single computer utilizes a large set of SSNs containing all the SSNs of the bank's online customers. (We'll explain how to determine this set shortly.) Note that the set will contain some SSNs not belonging to customers. The computer also needs the set of all possible PINs. If each PIN has n digits, then the set contains 10^n values.

When the attack starts, the computer picks an SSN from the set of SSNs and tries to log into the Internet bank using a randomly chosen PIN from the set of PIN values. Assuming that the SSN belongs to a customer, the probability of success is only 10^{-n} where $n \geq 4$ for the Internet banks we have studied. If the computer fails to log in, it tries again. Using the same SSN, the computer chooses a new PIN uniformly at random and repeats the login procedure. Since the bank closes access to the account after $T (> 1)$ trials with correct SSN and

wrong PIN, the probability of success is $p = T/10^n$.

The computer repeats the above procedure, once for each SSN in the determined set. Since this set contains the SSNs of all the bank's customers, a cracker is able to access at least one account with probability

$$\begin{aligned} \text{P}(\text{access at least one account}) &= 1 - \text{P}(\text{access no accounts}) \\ &= 1 - (1 - p)^Q, \end{aligned}$$

where Q is the number of customers in the Internet bank. The expected number of accounts a cracker gets into is $Q \cdot p$.

Note that the probability p was determined under the reasonable assumption that a bank generates customer PINs with uniform distribution. If the distribution is skewed, such that some PIN values are significantly more probable than others, then the described attack can be improved. As an example, the PINs may have a skewed distribution when the customers are allowed to choose their own PINs. According to Ross Anderson, about a third of the customers will use a birth date as a PIN in this case [2].

2.1 Norwegian SSNs

For a real attack to succeed, the cracker would've had to write computer code to generate a set of SSNs containing the SSNs associated with the Q online accounts. (As indicated earlier, the size of this set may be larger than Q , i.e., the set may contain SSNs not corresponding to any real accounts.)

The Norwegian population was about 4.5 million during 2003 and 2004. Each citizen is identified by a unique SSN consisting of 11 digits divided into three groups:

$$x_1x_2x_3x_4x_5x_6i_1i_2i_3c_1c_2.$$

Here, $x_1x_2x_3x_4x_5x_6$ is the date of birth (*ddmmyy*), $i_1i_2i_3$ is an individual identification number, and c_1c_2 are control digits:

$$\begin{aligned} c_1 &= 11 - \\ &\quad ((3x_1 + 7x_2 + 6x_3 + x_4 + 8x_5 + 9x_6 + 4i_1 + 5i_2 + 2i_3) \bmod 11), \\ c_2 &= 11 - \\ &\quad ((5x_1 + 4x_2 + 3x_3 + 2x_4 + 7x_5 + 6x_6 + 5i_1 + 4i_2 + 3i_3 + 2c_1) \bmod 11). \end{aligned}$$

All children born on the same day are assigned a unique $i_1i_2i_3$ number. The individual numbers are assigned chronologically and in such a way that i_3 is even for a girl and odd for a boy. If c_1 or c_2 is equal to 11, then the value is changed to 0. When c_1 or c_2 is equal to 10, the resulting 12-digit number isn't used because an SSN can only have 11 digits. To avoid confusion, we remark that a Norwegian SSN is called a "birth number" in Norway.

It should be clear that it is possible to generate efficiently a set of SSNs containing the SSNs of all Norwegian people between, say, the ages of 16 and 75. This set contains (nearly) all the SSNs of the customers in *any* Norwegian Internet bank. The cracker could've used this set during an attack. It was also possible for the cracker to reduce the size of the set by utilizing available statistical information on both the distribution of births in Norway and the usage of Norwegian Internet banks.

Norwegian account numbers also have a well-defined structure. A cracker could therefore run the same type of brute-force attack when a bank customer used an account number to log on instead of his or her SSN. More information on the structure of Norwegian account numbers is available in [3].

3 A distributed attack

Most likely, a bank's Intrusion-Detection System (IDS) will discover brute-force attacks similar to the attack described in the previous section, because no attempt is made by the cracker to hide the attack by e.g. spreading it over many days. Since the attack is run from a single computer, the bank can simply deny the computer access to its network to stop the attack. A cracker is of course well aware of this fact, and uses a so-called *botnet* to run the attack.

A botnet is a large network of PCs, called zombie PCs, that's controlled by a server. Worms such as MyDoom and Bagle are used to take over vulnerable PCs and add them to the expanding botnet. Often, the compromised PCs are controlled across Internet Relay Chat (IRC) channels [1].

Botnets can be large. A network of more than 10,000 zombie PCs was dismantled after security staff at the Norwegian telco Telenor located and shut down its controlling server [4]. It's also known that there are many botnets on the Internet. On September 20, 2004, The New York Times reported that the number of botnets monitored by Symantec increased from less than 2,000 to more than 30,000 during the six first months of that year. Symantec also saw a dramatic increase in electronic commerce attacks during the same period.

Let us once more consider the situation in Norway in 2003 and 2004. A cracker controlling a large botnet could divide a set of SSNs over all the zombie PCs in the network. Each PC could then try to log into an Internet bank using only the SSNs it was assigned. When the bank's IDS discovers the attack, the bank is not able to stop all the traffic from the zombie PCs because there are so many of them. Furthermore, since the attack is on the application layer of the network, it's difficult for the bank to discriminate between legitimate customers and zombie PCs trying to log on.

Note that the described attack is a combined brute-force/DDoS attack since the cracker could expect to get access to a small number of accounts $Q \cdot p$ while all the remaining $Q \cdot (1 - p)$ accounts would be closed to the customers.

3.1 Example attack

During 2003, new customers in a particular Norwegian Internet bank downloaded a certificate after providing an SSN and a fixed PIN with $n = 4$ digits. Unfortunately, the certificate could also be downloaded by a cracker knowing the SSN and PIN. Hence, the introduction of the certificate didn't really enhance the security.

The access to an account was closed after $T = 3$ unsuccessful login attempts. The probability that a cracker was able to log into at least one account was $P(\text{access at least one account}) \approx 1$ for $Q = 220,000$ customers. The expected number of cracked accounts was 66.

A cracker could possibly run the attack several times. Since the attack is stochastic in nature, he could expect to gain access to 66 new accounts each

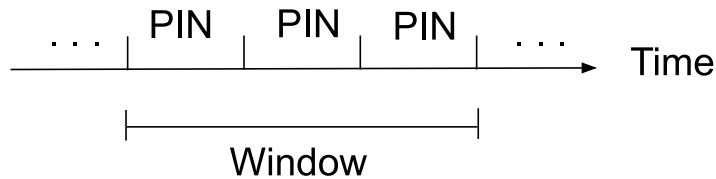


Figure 2: Window containing 3 PINs.

time. Fortunately, this Internet bank changed its login procedure in 2004.

4 Exploiting PIN calculators

A PIN calculator is supposed to provide two-factor authentication by requiring that the customer has something (the PIN calculator) and that he or she knows something (the secret PIN needed to activate the calculator). Often, the customer enters a *fixed* 4-digit PIN into the calculator to get a *dynamic* 6-digit PIN. This dynamic PIN is also called a one-time password.

The manufacturer initiates a new PIN calculator with data needed to generate a sequence of PINs. The same data is also stored in a database controlled by the Internet bank. The customer uses the calculator to generate a new PIN each time he wants to log into his account. The PIN is entered into the online bank's Graphical User Interface (GUI) and sent to the bank's security server. The server accesses the database to get the data for the given PIN calculator. It then calculates the current PIN and determines whether or not it's equal to the PIN generated by the calculator.

Many PIN calculators are programmed to generate a new (dynamic) PIN after a certain time period. It's therefore possible to divide a time line into equally long intervals and associate one PIN with each interval (see Fig. 2). Observe that both the calculator and the security server must have a clock to synchronize generation of the PINs in this case.

A problem occurs because of *clock drift*, i.e., the calculator and the server have different estimates of the time. To correct for the inevitable clock drift, a security server not only calculates the current PIN, but also the PIN in the previous time slot, as well as the PIN in the next time slot. The security server then compares the received PIN from the calculator with all three locally calculated PINs. In other words, the security server has a window of PINs that it will accept. Fig. 2 depicts a window of three PINs.

Several manufacturers allow larger windows of PINs. Provided a customer logs on regularly, an authentication server from RSA security will keep an estimate of the PIN calculator's time so that the dynamic PIN always falls within a window of three PINs. However, if a customer doesn't log on for an extended period, the PIN calculator's time could drift outside the window. In this case, the server tests against PINs in the 20 previous and the 20 next time slots, giving a window of 41 PINs [5].

Experimentation with two Vasco PIN calculators in 2004 indicated that the security server at a Norwegian Internet bank had a window of 19 PINs. The

large size was maintained even when the customer logged on regularly. The existence of a window was confirmed in [6].

Let L denote the size of a security server's window of acceptable PINs. The window makes it easier to access an account. The attacker now has to guess an arbitrary PIN in a set of L PINs instead of the fixed PIN a customer enters into the bank's GUI when a PIN calculator isn't used. The fact that the calculator generates dynamic PINs only means that the cracker must empty an account the first time he gets access because he needs a new (unknown) PIN to log on a second time.

The probability that the cracker is able to access a particular account is $q = L \cdot p$ for an SSN belonging to a bank customer. The probability that the cracker is able to access at least one account is

$$P(\text{access at least one account}) = 1 - (1 - q)^Q,$$

where Q again is the number of customers. The expected number of cracked accounts is $Q \cdot q$.

4.1 Example attack

Another Norwegian Internet bank had about $Q = 400,000$ customers during 2004, all using PIN calculators to generate dynamic PINs with $n = 6$ digits (in addition the calculators produced 2 control digits used to authenticate the bank's web site). The customers logged into their accounts using SSNs and dynamic PINs. Experimentation indicated that the bank closed the access to an account after $T = 5$ failed attempts to log on.

Assuming a minimum window of $L = 3$ PINs, the probability of getting access to at least one account is ≈ 0.998 , and the expected number of cracked accounts is 6. Increasing the window size to its maximum $L = 19$, determined by experimenting with two real PIN calculators received from the bank, gives an average of 38 cracked accounts. It was possible to repeat the attack to gain access to new accounts each time.

The bank received a written report from us outlining our concerns. We also met with representatives from the bank's top-level management. They asked us to withhold the name of the bank, but had no objection to us publishing our findings. At the time of writing, the bank has not changed the login procedure for its Internet banking system, however, we understand that the bank has developed an alternative login procedure.

5 Attack generalizations

In the following we discuss a few possible attack variations during 2003 and 2004, introduce an SSN-filtering technique, and explain why our results apply to many current client-server systems.

5.1 Simple attack variations

During 2003 and 2004 it was also possible for a cracker to launch a pure DDoS attack on the application layer against many Norwegian Internet banks. Since a bank closed down the access to an account after T login trials with correct SSN

and wrong PIN, all the customers' accounts could be closed down after $T \cdot S$ trials where S is the size of the set of SSNs. Note that while it was necessary to transmit a huge number of packets for a long time to launch a successful DDoS attack at a lower layer in the network, a DDoS attack utilizing an Internet bank's login procedure at the application layer could be carried out much more quickly. This made it hard to stop a real attack.

Assume that one bank was actually attacked. Once the bank had reopened the access to all accounts after the attack, it was possible for the cracker to start the DDoS attack again. Hence, the cracker could prevent (nearly) all bank customers from accessing their online accounts for a long time.

It was also possible for a cracker controlling a large botnet to launch DDoS attacks against several of the major Internet banks in Norway at the same time. Close to two million customers would be denied access to their online accounts in this case.

Several stealthy versions of both the simple attack launched from a single PC and the distributed attack were possible during 2003 and 2004. If a cracker had knowledge about a bank's IDS, he could try to design an attack to avoid detection. It was of course important to make the set of SSNs as small as possible. As an example, the cracker could reduce the set of SSNs by concentrating the attack on a particular age group, say, people between 35 and 50. This group was likely to have more money in the bank than younger people. The cracker could also spread the attack over many days to avoid too many unsuccessful logins on a single day.

One of the anonymous reviewers pointed out that a cracker could write an alternative attack program which held the PIN fixed as it ran through all the SSNs. This alternative attack was advantageous to the cracker when some PIN values were more likely than others.

The cracker could first run this attack using one of the most likely PINs. He could then repeat the attack several times using other PINs that occurred with high probabilities. Depending on the actual distribution of the PINs, the cracker could expect to get access to a significantly larger number of accounts than in the case of uniform PIN distribution.

5.2 SSN filtering

We have discovered that it's sometimes possible to determine SSNs belonging to customers in an online bank by exploiting the bank's own web site. A script can be written that tries to log into the site using different SSNs. For each SSN, the script receives a message from the site. Often, there are several types of messages depending on whether an SSN is valid and whether it belongs to a customer. The script can use these messages to determine customer SSNs. The described "filtering" of SSNs can be very helpful when constructing a small set of SSNs.

Other web sites—not belonging to Norwegian banks—may also be exploited to determine SSNs in use. In the following we report our experience with a site created by a Norwegian Public Service Pension Fund (NPSPF). A large percentage of the Norwegian population are members of this pension fund. In particular, government employees, including university employees, in Norway are members.

Any member of NPSPF could apply for a housing loan on the web during 2004. To apply for a loan, the applicant first entered his Norwegian SSN. A small script could also carry out the same operation. If the SSN was valid and it belonged to a member of the pension fund, then the script received a message containing the person's name and address, including the zip code; else the script received an error message.

Since the Norwegian SSNs have a well-defined structure, a script can generate a large number of SSNs that may or may not be in use. Using the NPSPF's web site during 2004, such a script could build a database containing the name, address, and SSN of a large number of Norwegian citizens. Because the database contained a zip code for each person, a cracker could easily create a set of SSNs belonging to people in a particular geographical area. This again enabled an attacker to go after the customers in smaller local banks.

One of our colleagues informed both NPSPF and the *Data Inspectorate*, an independent administrative body under the Norwegian Ministry of Labour and Government Administration, that it was possible to determine valid SSNs using NPSPF's web page for loan applications. In a letter to the Data Inspectorate, NPSPF promised to change the web page for loan applications within 14 days. However, when we checked after 2 weeks it was still possible to filter out a large number of SSNs. Our test script was able to access NPSPF's web site for nearly 24 hours. NPSPF and the Data Inspectorate were contacted once again, this time we included the SSNs and addresses belonging to the Norwegian Prime Minister and the Director of the Data Inspectorate. We also explained that an attacker could create chaos by applying for a large number of loans using other people's SSNs obtained from NPSPF's own site.

During 2003 and 2004 it was also possible to filter out SSNs from other Norwegian web sites. In fact, by combining SSNs from several sites, an attacker could build a database with the names, addresses, and SSNs of nearly everybody in Norway.

5.3 Generalization to other systems

It's important to realize that our insights apply to any client-server system on the Internet as long as the authentication of each client is based on a structured user ID and a short PIN (or password). Examples of IDs with a well-defined structure are SSNs, account numbers, patient IDs, and e-mail addresses. Whether a customer's PIN is static or dynamic makes little difference in our case.

A combined DDoS/brute-force attack represents a potential serious problem for a client-server system owned by a business enterprise. The attack is hard to stop, it closes down many user accounts, and it allows a cracker to access some of the accounts. An attack can also result in other problems for the enterprise. The venture can lose income if it closes down its servers to stop the cracker from accessing accounts. Bad press can result in loss of current and future customers, as well as reduce the level of trust afforded by the remaining customers. The value of the brand can be reduced, especially if it takes the enterprise a long time to reactivate all the accounts closed down by the attack. Consequently, designers of commercial client-server systems using structured IDs and fixed or dynamic PINs should verify that their systems are not vulnerable to an attack.

6 Countermeasures

This section discusses how a system can fend off combined DDoS/brute-force attacks at the application layer of the network.

6.1 Brute-force countermeasures

The degree of exposure to brute-force attacks depends on the number of digits n in the PINs. Consider one client-server system using fixed PINs and another system using dynamic PINs, and assume that all PINs in both systems have n digits. Let us compare the two systems' exposure to brute-force attacks. If the dynamic PINs are generated by PIN calculators and the security server has a window of length $L = 10$, then it is 10 times more likely that an attack program guesses the correct PIN for a given customer compared to the case when the PINs are fixed.

The system using dynamic PINs of length n has the same level of security as a system using fixed PINs of length $n - 1$. In other words, the use of PIN calculators can cause a system to become significantly more vulnerable to brute-force attacks. Hence, it may be necessary to use longer PINs in systems utilizing PIN calculators than in systems using fixed PINs.

As the number of users grows, a client-server system using PINs of a set length n becomes more vulnerable to brute-force attacks. A designer of a new system must therefore estimate the number of future users before she can determine the number of digits n needed to be safe from brute-force attacks. The simple calculations introduced earlier in the paper show how to determine whether or not the PINs have enough digits.

Brute-force attacks can be further hampered if large random numbers are used as user IDs, making it difficult for an attack program to generate these IDs.

6.2 DDoS countermeasures

There exist various DDoS attacks targeted at different network layers. No general defense against these attacks exists. We refer the interested reader to [1] for a comprehensive introduction to DDoS attacks and the different techniques used to limit the negative consequences of the attacks.

We've seen that if a client-server system closes down a customer's access to the server after a few login trials with correct user ID and wrong PIN, then a simple and efficient application-layer DDoS attack can prevent all customers from getting access to the server. Clearly, well-designed client-server systems should not utilize ID/PIN authentication techniques that reduce the amount of resources a cracker needs to carry out a DDoS attack at the application layer.

One possible solution to this particular DDoS attack is to base a client-server system on a Public-Key Infrastructure (PKI) that requires new users to show up in person at the registration authority before getting access to the system [7]. In this case it is no longer necessary to transmit IDs and PINs from the clients to the server. Instead, the server can verify that a client has the (long random) private key corresponding to the public key in the client's certificate.

A PKI-based solution may still need a PIN to unlock a client's private key. However, this PIN need not be transmitted to the server. On the other hand,

there exist PKI solutions where user IDs and PINs are sent from the clients to the server. These solutions may be vulnerable to DDoS attacks.

7 Discussion

To help foster development and maintenance of distributed systems with better security, we first discuss why the combined DDoS/brute-force attack was possible during much of 2003 and 2004. Next, we consider the dangers associated with the Norwegian banks' security-by-obscurity policy, before finally discussing how universities can teach computer science students to develop more secure systems.

7.1 Bad security

A high level of expertise isn't needed to carry out the presented attacks; they're only based on well-known brute forcing and DDoS techniques. A well-designed system should of course not be vulnerable to brute forcing in practice. In fact, this is one of the first things a designer of a new system should verify. Why then were many Internet banking systems in Norway with a total of more than one million customers vulnerable to the combined DDoS/brute-force attack during 2003 and 2004? Our answer is based on discussions with representatives from Norwegian banks and a report [8] from *Kredittilsynet*, the Norwegian government agency overseeing the banks.

Many of the banks' security experts and software developers had prior experience with Automatic Teller Machine (ATM) systems. The (mental) models developed during the ATM work have—to a large degree—influenced the design of the Internet banking systems. Since it's difficult to access customer accounts in an ATM system by brute-force attack, it seems that the banks paid insufficient attention to the comparative ease with which a cracker can assail an Internet banking system by brute force.

Because no Norwegian bank was instructed by Kredittilsynet to develop a separate risk analysis for its IT systems before August 2003, few banks had in place mechanisms to determine an acceptable level of risk for these systems [8]. Consequently, the security of a system could very well deteriorate over time without a bank discovering the fact. Of course a successful brute-force attack requires that an Internet bank has a large number of customers. Initially, the number of customers was relatively small in Norwegian Internet banks. As the number of customers grew, not all banks realized that it was necessary to increase the number of digits in the PINs to avoid being vulnerable to brute-force attacks.

Many banks had outsourced the daily operation of their Internet banking systems during 2003 and 2004. Kredittilsynet [8] has pointed out that it was difficult for these banks to maintain the needed security expertise when no longer they could learn from their own systems. A single corporation operated most of the Internet banking systems during the discussed time period. Since this company had nearly a monopoly, and many banks no longer had the ability to evaluate the security of their outsourced systems, there were no external mechanisms outside the corporation to ensure that the Internet banking systems maintained a high level of security over time.

Taken together, these reasons led to much of the bad security in Norwegian Internet banking systems. In the following, we argue that the banks' security-by-obscurity policy was also a contributing factor.

7.2 Security by obscurity

Quoting Bruce Schneier [9], "there is a considerable confusion between the concept of secrecy and the concept of security, and it is causing a lot of bad security." Like many foreign banks [10], Norwegian banks have long practiced security by obscurity. No technical information about the banking systems' security protocols and use of cryptographic primitives is made available to independent security researchers or customers complaining about debits on their accounts for which they were not responsible. The banks have for many years simply stated that their systems are very secure.

Our analysis shows that the security in Norwegian Internet banking systems may not be very high after all. We believe that the banks' security-by-obscurity policy has led to a false feeling of security instead of real security, making the Internet banking systems vulnerable to rather trivial attacks during 2003 and 2004. It's well known that it's much more difficult to design a new secure system than to find vulnerabilities in an existing system. After a designer has invested much time, effort, and prestige on a new design, he or she may not be motivated to find weaknesses in the same design. It's therefore important to hire outside experts to evaluate all new security designs. It's equally important to evaluate the implementation on a regular basis. As an example, we have seen how a Norwegian Internet bank has several times been vulnerable to cross site scripting and, hence, "phishing" e-mail scams after it made changes to its web site.

In Norway, the banks' upper-level management is much to blame for the current practice of security by obscurity. Management typically has little understanding of real security, and has a tendency to assume that a system is secure if all information about it is kept secret. Consequently, all employees responsible for the security must sign nondisclosure agreements making them ill-prepared to discuss security problems with anybody outside the banking industry. This was amply demonstrated to us when we tried to inform selected banks about our findings. In our opinion the security-by-obscurity policy creates unnecessary friction, prevents us from learning, and, hence, causes the same mistakes to be made over and over again.

7.3 Teaching security

It's desirable to develop better security courses for tomorrow's computer science students. Initially, such a course should evaluate the security in some existing systems, preferably systems that the students already use. An 'holistic' approach should be taken, covering the main security techniques implemented in the evaluated systems; difficult technical details should be avoided because they will only cloud the important issues at this early stage.

A course should cover banking systems, the next biggest application of cryptography after government systems. We believe that future Internet banking systems must be based on PKIs with client certificates [7] to strengthen the customer authentication. Consequently, it's important to analyze at least one real

PKI system during a course.

The new Norwegian BankID standard for Internet banking has a PKI. Unfortunately, the Norwegian banks have (once again) decided to keep the complete standard a secret, and not allow an independent evaluation of the standard. Clearly, it's important to teach the students the difference between what information needs to be kept secret and what information may be shared.

We're of the opinion that it's necessary to study real attacks in a security course. A good understanding of attacks helps students analyze the security through the eyes of a cracker; exposing weaknesses and determining the most serious risks. Furthermore, demonstrations of "real" attacks make wonders when it comes to motivating students to incorporate security in the initial system design.

In particular, DDoS attacks should be studied. As we have seen, it isn't a good idea to design a login procedure that simplifies a DDoS attack by closing down access to an account after a few login attempts. The discussed security course should contain an introduction to different types of DDoS attacks and techniques to mitigate such attacks.

The decision to teach attack techniques should not be taken without some serious thought to the potential consequences. It's irresponsible to study attacks without also including some discussion about what constitutes ethical behavior. It may be a good idea to style this part of the course as an introduction to penetration testing to emphasize that the attacks are taught to discover vulnerabilities in systems, not to attack systems for personal financial reasons.

An understanding of vulnerabilities and attacks alone isn't enough to develop secure systems that let us escape from the endless cycle of penetration and patch. We recommend [11] for a discussion on how to teach constructive security. Information on how to build secure software may be found in [12].

8 Final remarks

Internet banking is increasingly popular both in Norway and in many other countries. The banks have actively encouraged this trend by persuading customers to sign up as a cost-saving measure. The online banking customers appear to be driven by convenience and aren't much concerned about identity theft and "phishing" e-mail scams. In fact, most customers seem to believe that Internet banking is very safe, simply because their banks told them so. In reality, the customers may well have a false sense of security. We believe more online banking systems must be evaluated by independent security researchers to determine the true level of security. Our own investigation shows that the authentication of many Norwegian online customers was too weak in 2003 and 2004.

Like many other security researchers, we have experienced how hard it is to alert banks (and other large institutions) to security weaknesses. The press on the other hand is very interested in the results of our research. In one instance we informed the press about our findings after the bank in question had made changes to its system. We got a lot of media attention, but also some very negative comments from mostly anonymous sources. In a few cases it was even suggested to us that we had personal financial motives for our investigation of the banks' security procedures. Personal attacks like these only show how

important it is to partake in the public debate about security issues in an open and straight forward manner—and how vital it is to create more and better security courses in our universities.

References

- [1] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service Attack and Defense Mechanisms*. Prentice Hall, 2005.
- [2] R. Anderson, *Security Engineering*. Wiley, 2001.
- [3] IBAN, “Domestic account number.” [Online]. Available: <http://www.ecbs.org/Download/TR201/norway.pdf>
- [4] The Register, “Telenor takes down massive botnet.” [Online]. Available: http://www.theregister.co.uk/2004/09/09/telenor_botnet_dismantled/
- [5] RSA Security, “The power behind RSA SecurID two-factor user authentication: RSA ACE/server,” 2001, white paper.
- [6] Vasco, “VACMAN controller integration,” technical white paper.
- [7] C. Adams and S. Lloyd, *Understanding PKI*, 2nd ed. Addison-Wesley, 2003.
- [8] The Financial Supervisory Authority of Norway, “Risk and vulnerability analysis 2003,” 2003, (in Norwegian).
- [9] B. Schneier, “Keeping network outages secret.” [Online]. Available: <http://www.schneier.com/crypto-gram-0410.html#2>
- [10] R. Anderson, “Why cryptosystems fail, from communications of the ACM, november, 1994,” in *William Stallings, Practical Cryptography for Data Internetworks*. IEEE Computer Society Press, 1996. [Online]. Available: <http://citeseer.ist.psu.edu/anderson94why.html>
- [11] C. E. Irvine, “Teaching constructive security.” *IEEE Security & Privacy*, vol. 1, no. 6, pp. 59–61, 2003.
- [12] J. Viega and G. McGraw, *Building Secure Software*. Addison-Wesley, 2002.

Paper III: Lessons from the Norwegian ATM system

Lessons from the Norwegian ATM system

Kjell J. Hole, Vebjørn Moen, and André N. Klingsheim

Abstract

This paper first analyzes a Norwegian court case involving a stolen ATM card misused by unknown person(s) who somehow knew the card's PIN. A fictitious attack scenario shows that as long as the Norwegian ATM system used DES to verify a PIN, it was possible for a skilled cracker to determine the PIN belonging to any Norwegian ATM card. It's then discussed how new and open development processes can lead to improved security and usability in future banking systems, as well as better legal protection for the bank customers.

1 Introduction

The authors study a court case from 2004 concerning a Norwegian citizen whose ATM (Automatic Teller Machine) card was stolen and later misused by unknown person(s) [1]. These thieves had somehow obtained the correct PIN (Personal Identification Number) associated with the card.

According to the judge's verdict [1], the Norwegian ATM system employed *single* DES (Data Encryption Standard) [2], or simply DES, to verify a PIN at the time the card was stolen. Despite this assumption, the bank responsible for issuing the card claimed it was impossible for the thieves to ascertain the PIN from the information on the card's magnetic strip during the hour it took from the card was stolen until it was first misused.

We'll introduce a simple model of an ATM system that uses DES to verify PINs and describe a theoretical attack scenario utilizing a two-step attack strategy. The first step is time-consuming and can be carried out before an ATM card is stolen. The second step can then ascertain the PIN belonging to the ATM card in a matter of seconds. In fact, after the first step is completed, the second step can determine the PIN belonging to *any* card issued by the bank in our model.

The Norwegian card owner lost his case because the judge decided it was impossible to establish the PIN during the available time. The attack scenario shows that the judge based his decision on wrong information.

After citing additional information concerning the Norwegian Internet banking systems during 2003 and 2004 [3], the authors assert that the Norwegian bank community's refusal to provide adequate security information is a threat to the citizens' legal protection. We also explain why this secrecy causes the security of banking systems to deteriorate over time. Finally, we make clear why new and open development processes can lead to both improved security and better legal protection in the future. These processes are discussed in some detail.

While we only study Norwegian banking systems, our findings are applicable to many other commercial systems. We leave it to the reader to apply our insights to other systems. The reader should note that the fictitious attack scenario is no longer a threat since today's Norwegian ATM system is based on triple DES.

2 The court case

In 2001, unknown person(s) stole two shoulder bags from a Norwegian citizen—we'll call him Mr. A—at an airport in Spain [1]. Mr. A lost his wallet with six Norwegian payment cards, while his wife lost four cards. Unidentified thieves later misused two of the cards.

Most payment cards issued by Norwegian banks are ATM ready and consist of two parts, one Visa/MasterCard part used abroad and one BankAxept part used in Norway. Since the *same* PIN is associated with both parts, it's theoretically possible to calculate the PIN from the BankAxept part and misuse the Visa/MasterCard part and vice versa.

We'll concentrate on the court case concerning the misuse of one of Mr. A's stolen cards. Since the court didn't consider the Spanish ATM system at all during the trial, we'll only consider the Norwegian ATM system. The stolen card was a MasterCard/BankAxept card issued by a particular Norwegian bank. Using this ATM card, unknown criminals were able to withdraw more than 9,000 Norwegian kroner (NOK) about an hour after Mr. A's bags were stolen. The ATM card was misused four times during a period of about 6 minutes. Each time, the right PIN was entered on the first attempt according to the verdict [1].

In Norway there is a national committee called "Bankklagenemda", established to solve disputes between Norwegian banks and their private customers. "Bankklagenemda" didn't believe the unidentified criminals had obtained the correct PIN by looking over Mr. A's shoulder, because the last time he had used the card was at the airport in Norway before leaving for Spain. This argument is strengthened by the fact that one of his wife's stolen cards with a different PIN—not used at the Norwegian airport—was also misused in Spain.

While Mr. A claimed the only written copy of his PIN was kept in a safe at home, "Bankklagenemda" ruled that he must have kept the PIN together with the card in the stolen wallet. With reference to the relevant Norwegian law, "Bankklagenemda" then made Mr. A responsible for 8,000 NOK of the loss.

Mr. A didn't accept the ruling and took the case to court in 2004. The defendant was the bank responsible for issuing the credit card. According to the scenario favored by the court, Mr. A's PIN was first encrypted with DES and then stored on the card's magnetic strip during the production of the card. The bank's two expert witnesses claimed it was impossible to determine the PIN from the information on the magnetic strip during the hour it took from the card being stolen to the first time the card was misused.

The expert witness for the plaintiff on the other hand, explained how most of the cracking could be done in advance if the criminals had prior access to a small number of cards issued from the bank. The judge chose to believe the bank's experts and concluded that the plaintiff most likely had kept a copy of his PIN in the stolen wallet. In the three following sections we'll explain why

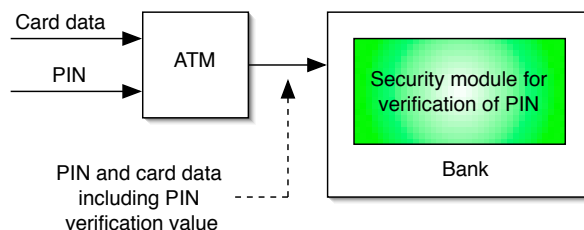


Figure 1: Simplified model of ATM and bank.

the plaintiff's expert witness was correct.

3 Model of ATM system

Figure 1 depicts a simplified model of the Norwegian ATM system during the period it employed DES to verify a PIN. To withdraw cash from the ATM in the model, a customer places her card in the card reader and types the PIN on the keypad. Information on the card's magnetic strip, including a value denoted the *PIN verification value*, is first read and then transmitted over a secure channel to the bank. As we shall see later, the PIN verification value is of particular interest to us.

The bank in Figure 1 employs a (hardware) *security module* [4] to verify the PIN. The verification process is shown in Figure 2. The security module uses DES encryption with a 56-bit secret key protected within the module. The 64-bit block of input data to the DES encryption consists of the customer's PIN and data from the ATM card's magnetic strip. Note that the PIN verification value from the card is not encrypted. Instead, the 64-bit output block from the DES encryption is transformed and compared to this PIN verification value. If the two values are equal then the PIN is accepted by the bank and the customer is allowed to withdraw cash from the ATM.

The transformation in Figure 2 is not the same in all real ATM systems. The exact function used in the Norwegian system is not publicly known. In our simplified model we only assume that the transformation produces a 16-bit result. For simplicity, all possible 16-bit values are assumed to be equally likely.

The same type of security module is used both during the production of ATM cards and the real-time verification of PINs in our model. The part of Figure 2 starting with the input to the DES encryption and ending with the output from the transformation defines how a 16-bit PIN verification value is generated. Clearly, to obtain a match between the pre-calculated PIN verification value and the value generated during a bank's real-time PIN verification, both values must be based on the same DES key. In our model, this DES key is used to verify

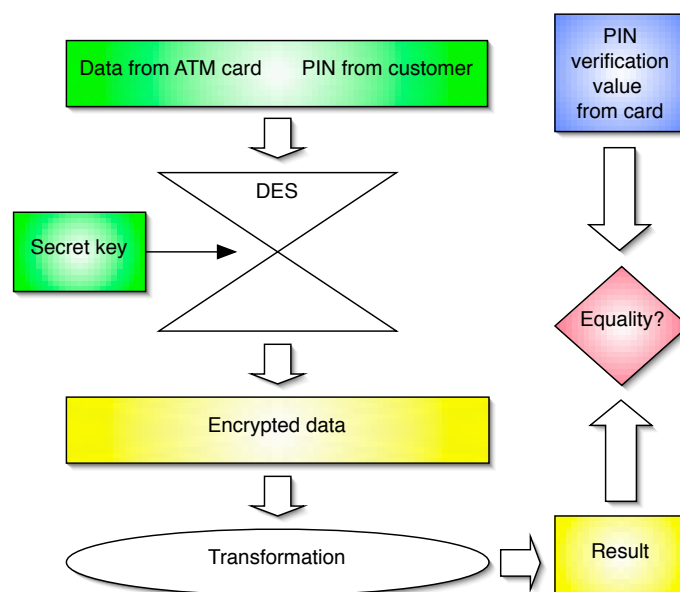


Figure 2: General outline of the PIN verification inside the security module.

the PINs belonging to *all* cards issued by a given bank. Different banks have different DES keys.

For a given 64-bit input block to DES, observe that many candidate keys result in the correct 16-bit PIN verification value. Hence, the PIN verification value partly determines the correct key, or, in more formal terms, the PIN verification value constitutes a 16-bit condition on the secret key.

4 How to determine a PIN

Using the ATM system model, we'll show how a fictitious cracker could establish the PIN belonging to any ATM card issued by a particular Norwegian bank during the time DES was used to verify PINs. To do this, the cracker had to be able to execute the internal operations of the security module (see Figure 2). Even if he didn't succeed in buying a module, it's still reasonable to assume he could determine its operations.

Several publicly available ISO (International Organization for Standardization) documents describe the techniques and data formats employed in security modules during the time DES was utilized [5, 6]. Furthermore, it's likely that, throughout the many years the modules were employed, a few of the manufacturers or some of the many banks leaked documents describing the modules' internal operations. Disgruntled former bank employees with intimate knowledge of the ATM system also had the necessary knowledge. Hence, an attacker could obtain the needed information to write a program to carry out all operations of a module. He could also buy a card reader to obtain the information on the magnetic strip of an arbitrary ATM card. The card reader could be connected to a computer to efficiently feed the data from a card into the program

mimicking the operations of the security module.

Suppose an attacker wanted to learn the PIN belonging to a particular ATM card. For now we assume he knew the DES key in Figure 2. (we'll show how to determine this key later on.) The attacker simply had to let the program try different PINs until the transformed value of an encrypted PIN was equal to the PIN verification value (see Figure 2). Note how this very simple technique gives the correct PIN because the PIN verification value is available on the ATM card itself!

Four-digit PINs were employed together with the ATM cards. If all 10,000 possible PIN values were used equally often, then the discussed program had only to try 5,000 PINs on average to determine the correct PIN belonging to a card.

5 How to determine a DES key

Our fictitious cracker had to discover the secret DES key stored in a bank's security module before he could run the program to establish PINs belonging to ATM cards issued by the bank. We'll now study how to determine such DES keys.

5.1 DES crackers

DES [2] is a block cipher encrypting 64-bit blocks of input data. The uppermost block in Figure 2 represents one of these input blocks. The encryption of each new input block is determined by the same secret 56-bit DES key, resulting in a stream of 64-bit blocks of encrypted output data.

EFF (Electronic Frontier Foundation) needed 18 months to design and build a computer to crack DES keys [7]. On July 17 in 1998 EFF determined the DES key used by RSA Data Security to encrypt a secret message. EFF's DES cracker simply tried a large number of different keys during a time period of 56 hours to obtain the correct DES key. Early in 1999 a large network of computers calculated a correct key in 22 hours and 15 minutes [8]. Using programmable logic arrays technology, Bond and Clayton later implemented an efficient DES cracker for 995 US dollars [9].

Cryptographers had already asserted for many years that 56-bit DES keys were too short to offer a high level of security [10]. EFF's DES cracker proved this assertion once and for all, and showed at the same time that many security systems based on DES didn't have the needed level of security. The DES-based security modules in the Norwegian ATM system were examples of such unsafe systems.

5.2 Access to ATM cards

It has been possible for a skilled attacker to crack DES keys since the early '90s. In the following we'll describe how the attacker could learn the DES key belonging to a bank's security module, using a DES cracker and a small number of ATM cards with known PINs. We first introduce a simple attack to establish how many ATM cards the cracker needed to determine a DES key. A more efficient attack is then described.

Assume that the attacker had one ATM card with known PIN, e.g., his own card. He then knew the complete content of the 64-bit input block to the DES encryption in Figure 2. However, the attacker had a problem because he didn't know the corresponding 64-bit output block—only the PIN verification value assumed to consist of 16 bits in this description. The problem for the attacker was that a large number of keys caused the transformation in Figure 2 to produce a value equal to the PIN verification value.

During the attack it was therefore necessary to try all 2^{56} keys and collect the 2^{40} keys resulting in equality in Figure 2. The number 2^{40} follows from the observation that the cracker had a 16-bit condition on the key. Consequently, there are $2^{56}/2^{16} = 2^{40}$ keys giving the correct PIN verification value. However, only one key can result in the correct determination of PINs belonging to all ATM cards issued by the bank.

The correct key was contained in the set of the 2^{40} remaining keys. This set could be reduced to a set of 2^{24} keys by trying all the 2^{40} keys together with a *new* ATM card with a different PIN and PIN verification value. To obtain the new card the attacker could open another account, have another person open an account, or simply steal the card and PIN from one of the bank's customers.

The set of 2^{24} keys could be further reduced to 2^8 keys using a third card. The correct DES key was then determined utilizing a fourth card.

The outlined attack made it necessary to store 2^{40} keys. If the attacker had access to four different ATM cards with known PINs before the attack, he then could test each of the 2^{56} keys against the four cards right away and, thus, remove the need to store a large number of keys. This modification also reduced the number of candidate keys needed to be tried to 2^{55} on average.

It's possible to further simplify the attack when a card owner is allowed to change the PIN. An attacker can then obtain the needed pairs of PIN and PIN verification values from a single card by repeatedly changing the PIN and reading the corresponding PIN verification value from the card's magnetic strip.

5.3 The main point

Even today it takes some time to calculate the key belonging to a security module employing DES. The cracking of a key also requires purpose built hardware or a large collection of regular PCs. The important point is, however, that it's only necessary to determine *one* DES key per bank.

After a DES key is known, the simple program described earlier can ascertain the PIN belonging to any stolen ATM card issued by the bank by simply trying 5,000 PIN values on average. This program can be run on a small laptop today. The same was clearly true during the '90s. In other words, after a skilled attacker had determined the DES key belonging to a security module in a particular bank, an unskilled operator of a simple PC program could establish the PIN belonging to any stolen ATM card from this bank in a matter of seconds.

5.4 Did an attack take place?

According to one line of thought it's unlikely that real attacks took place because this would've lead to a massive number of unexplainable withdrawals from Norwegian bank accounts. Centers would've popped up where criminals could come to determine PINs belonging to stolen cards. Alternatively, criminals could've

“rented out” laptops running phase 2 of the PIN cracking to thieves stealing cards from a particular bank.

On the other hand, it’s possible to argue that real attacks would not have lead to massive fraud. According to our model a thief must physically steal each ATM card. Most card owners report card theft to their bank right away. The bank then closes down the account making it impossible to withdraw any cash. The rapid closing of accounts forces a thief to steal no more than a few cards before he tries to withdraw cash from an ATM, which limits how many cards he can misuse during a given time period.

It follows that a group of thieves was needed to steal and successfully misuse, say, five hundreds cards per year. Because the ATM system has had, and still has, limits to how much cash a customer can withdraw from an account during a single week, the group members would’ve had to operate in different locations to steal enough cards to make the operation profitable. Attacks on different banks would’ve lead to a distributed geographical pattern of ATMs processing stolen cards, much like the pattern we’ve seen during the last decade.

The number of misused cards and the geographical pattern of abused ATMs depend on the number of criminal groups. The level of expertise and resources needed to determine one DES key per targeted bank, as well as the actual number of stolen cards each year, make it unlikely that more than a very small number of groups can have existed in Norway.

It’s hard both for banks and outside experts to discover whether or not real attacks occurred since there is no simple way to determine if a DES key has been cracked. An uncertainty about how a thief could obtain a PIN remains, making it debatable whether the plaintiff in the cited court case actually kept the PIN together with the stolen ATM card.

6 Too much secrecy is counterproductive

In this section, we’ll discuss why too much secrecy causes the security of a banking system to deteriorate over time, and why a bank’s refusal to share technical information is a threat to a customer’s right to legal protection during a conflict.

6.1 Court case revisited

The bank’s expert witnesses seemed not to know about the described attack scenario, or if they did, they didn’t acknowledge this during the trial. Their claim that it was impossible to determine a PIN belonging to a stolen ATM card in less than an hour shows a limited understanding of the level of security provided by the DES-based ATM system. The same lack of understanding was observed when we analyzed the Internet banking systems in Norway. Several banks were completely unaware that they were vulnerable to distributed attacks during 2003 and 2004 [3].

Notice how DES cracking only became a threat as time passed and computer technologies improved. Similarly, the attack scenarios against the Norwegian Internet banking systems were not a problem when the systems were new and had few users, but as the number of users grew the systems became more and more vulnerable.

Because Norwegian banks keep all system information secret and don't allow independent experts to analyze their systems, there is a limited number of security experts with an intimate understanding of the Norwegian banking systems. In the long run, the banks' own experts have a tendency to think alike, especially since they cannot freely discuss the banking systems with outside experts. As a result, they have a propensity to overlook slowly developing system vulnerabilities.

As long as the true level of security is hidden from the Norwegian courts, it's difficult for bank customers to win cases against the banks. The case discussed in this paper demonstrates the problem. The international research community had known for many years that DES was unsafe. Still it was difficult for the plaintiff's lawyer to refute the assertion from the bank's experts because the bank didn't have to provide him with any information about the ATM system.

The plaintiff appealed the verdict to a higher court. According to the judge's ruling after the first trial, the ATM system used DES to verify a PIN when the card was stolen. During the appeal process the defendant's lawyer tried to show that the ATM system had utilized triple DES, and not DES, to verify PINs. The fact that this very important information was not established during the first trial only underscores how important it is to have access to correct technical information.

During the appeal process the plaintiff's lawyer asked for more information, but very little was given. In particular, the bank argued that an encryption algorithm developed to do PIN verifications for MasterCard transactions must be kept secret. According to modern security thinking there is no need to keep cryptographic algorithms secret. In fact, it's considered very bad practice. Unfortunately, economical and personal reasons led the plaintiff to withdraw the case before it could be considered by the higher court.

6.2 "Bankklagenemda" revisited

During the last decade the committee has considered a large number of cases involving stolen ATM cards. In nearly all instances the committee concluded that the card owner must have stored the PIN together with the card. The owner typically didn't agree. This state of affairs continues even today. During 2004 there were around 500 cases involving misused ATM cards.

The committee has all along based its decisions on the assumption that the ATM system has had, and still has, a high degree of security. As evidence they initially referred to a note from 1993 penned by the Norwegian Central Bank. According to the Central Bank it was not possible to crack the PIN using the information on the magnetic strip. Our fictitious attack scenario shows the opposite, it was indeed possible to crack PINs during the '90s.

Later the committee started to refer to a letter from 2002 written by The Financial Supervisory Authority of Norway, a government agency supervising the Norwegian banks. The letter cites a security report first completed in 1997 and then re-evaluated in late 2001. This report written by representatives from the Norwegian bank community isn't available to the public.

Even though some card owners simply forgot they wrote down the PIN, and others lied, it's unfortunate how "Bankklagenemda" has branded a large number of bank customers as liars without a thorough review of the security in the ATM system. Unlike the earlier secret self-evaluations from the Norwegian

bank community, a new review should be carried out by independent security experts and made available to the public.

Even after the upgrade to triple DES there are indications that the level of security may be lower than advertised by the banks. R. Anderson *et al.* [10] have described numerous physical and logical attacks on security modules, including powerful remote attacks on a module's application programming interface. A clever insider attack determining triple-DES keys is described by Bond and Clayton [9].

6.3 Weak authentications

Authentication is the process of establishing confidence in the truth of some claim [11, Ch. 2]. In particular, an authentication process does not *prove* that a particular individual is who she claims to be. The process can only provide a level of confidence in the claim. Unfortunately, a high level of confidence in an authentication method may well be undeserved. The security of a system suffers when developers have high confidence in an authentication technique that turns out to be vulnerable to attacks by crackers.

We define an authentication method to be *weak* if it's susceptible to a practical attack scenario. An attack scenario is practical if it's reasonable to believe it can be carried out by skilled crackers. The fictitious attack scenario in this paper reveals that the customer authentication in the Norwegian ATM system was weak during the last years DES was used. The attack scenarios reported in [3] show the customer authentication in several Norwegian Internet banking systems to be weak during 2003 and 2004.

7 Toward better development processes

The *architecture* of a banking system defines its conceptual structure and logical organization. The *design* specifies how to create the system and includes the description of communication protocols and cryptographic primitives. Our investigations of the ATM system and Internet banking systems [3] strongly indicate that Norwegian banks didn't perform thorough periodic reviews of the underlying architectural and design assumptions. In fact, according to The Financial Supervisory Authority of Norway ("Kredittilsynet"), no Norwegian bank was instructed to carry out a separate risk analysis of its IT systems before August 2003 [12]. This helps explain why the customer authentications were allowed to become weak over time. Discussions with bank experts have led us to conclude that improved architectural and design processes—better incorporating security and usability aspects—are needed to create more secure banking systems. These processes must produce architectural and design documents understandable not only to the developers of the system, but also to external security experts. The lack of adequate documentation makes it difficult to carry out periodical security reviews of the architecture and design, reducing the likelihood that slowly developing security problems are discovered before they become serious.

In the following, we first outline an alternative to a hierarchical organized development team. We then describe how the team can incorporate security and usability in the architectural and design processes and produce useful documentation. Finally, we discuss how this documentation can also be used during

a conflict between a customer and a bank.

7.1 Development team

A hierarchical organization tends to promote selfish behavior [13]. Leaders wanting to strengthen their own positions in the hierarchy discourage the rank-and-file members from asking critical questions or pointing out mistakes. Groups in an organization where critical thinking is discouraged are unlikely to perform well in the long run. Members only do what they are told, and take little or no personal responsibility for the final outcome of their work. In particular, development teams are unlikely to produce secure systems in environments where critical thinking is discouraged.

A team developing a banking system should be organized as a *heterarchy*—not a hierarchy. Heterarchy means 'multiple rule', a balance of powers rather than the single rule of hierarchy. In a heterarchy, authority is determined by knowledge and function. As the development process progresses, different team members take charge of the process [13].

Members of a successful heterarchy share knowledge and help solve each other's problems. To ensure efficient cooperation, it's important to take the time in the beginning of a project to make sure that all members communicate well and, thus, develop similar mental models. During the project it's vital to build an open culture allowing the team members to utilize all their talents.

In practice, the development team is a small heterarchy in a hierarchical bank organization. Because of the overall hierarchical organization, the team may very well have a formal leader. To ensure that all team members collaborate, the team leader should facilitate free interaction between members and only exercise control during prolonged conflicts.

Open-source collaborative principles [14] capture the most important aspects of a heterarchy. An open dialog between the members is encouraged as well as critical evaluations of ideas. A critical assessment of architectural and design concepts requires the (core) design team to work closely with different kinds of external experts. This is a problem for Norwegian banks since they are used to keep all technical details secret. Unfortunately, secrecy leads to 'groupthink' that discourages creativity and individual responsibility. Bank employees need to learn about openness. See [13, 15] for more information.

7.2 Initial development process

Figure 3 depicts the *initial* stages of a development process producing a system design. (See [16] for a description of the complete process.) Note that we do not assume a particular software development methodology (such as the Rational Unified Process or the waterfall method). If the development team utilizes an iterative process, then it will cycle through the diagram multiple times.

The first step is to describe the functionality of the new banking system. Often the functionality is described in terms of use cases. Similarly, abuse cases are developed to describe the system's behavior under attack. Together, the use and abuse cases form the basis for the security and usability requirements. The architecture is then developed from these requirements.

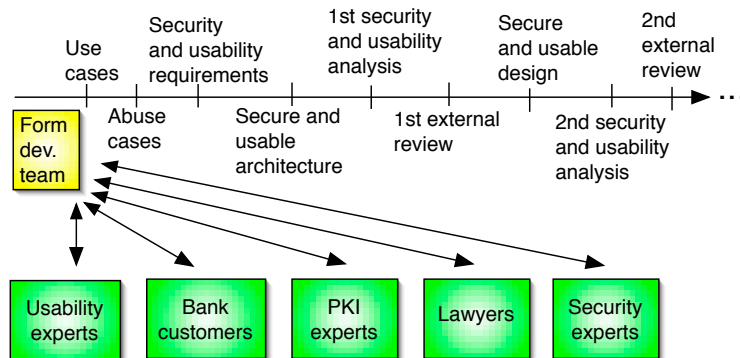


Figure 3: Initial stages of development process for (PKI-based) banking applications where a core development team collaborates with external groups.

7.3 Collaborative architectural process

Norwegian banking systems with weak authentication have been described. To significantly strengthen the authentication in a future banking system, the authentication should be based on a Public-Key Infrastructure (PKI) [17].

We assume that a PKI is needed to fulfill the functionality description since this helps us illustrate the need for collaboration. While it's not difficult to understand the principles of public-key cryptography, it's very hard to design a good PKI. The PKI literature is large and it can be both time consuming and hard for a development team, especially a small team, to obtain a good overview of all critical PKI issues. It's therefore advantageous for the team to collaborate with PKI experts (as indicated in Figure 3). Even if the team decides to buy a PKI solution, it's valuable to have independent PKI experts evaluate the different PKI offers.

A PKI can offer the service of digital signatures, a service analogous to handwritten signatures [17]. A PKI user may deny that a digital signature came from him. This denial is referred to as *repudiation* of the signature. Outside PKI experts are particularly useful if the banking system is to offer digital signatures with a high degree of *non-repudiation*. The goal of the non-repudiation service is to be able to provide credible evidence to a third party that a the signature came from a particular person. This third party is often a judge, jury, or independent arbitrator [18].

Non-repudiation cannot be achieved by technical means alone. To provide a third party with convincing evidence, the development team must collaborate with a team of lawyers to understand what constitutes good evidence according to the relevant local and/or international laws. The architecture should be designed to facilitate the presentation of non-repudiation evidence in court. This evidence must be understandable to people with absolutely no prior knowledge of security. The lawyers can also draw up legally binding contracts to further support the needed degree of non-repudiation.

If people are unable to use a banking system in a secure way, they'll use it in an insecure way. Security must therefore be combined with usability [15]. In fact, usability is king in a banking system. If the usability is poor then many

customers will not use the system at all. Usability goals should therefore be described right after the use cases are developed. The usability goals are particularly important when developing a new banking system for a large number of customers whose computer skills vary widely. Many developers believe there is an inherent trade-off between security and usability. However, this isn't necessarily true when both security and usability are designed into the system from the very start.

The development team should develop the usability goals together with usability experts and bank customers. Important goals in our PKI example is to make certificate generation, installation, and revocation painless for users, and make it easy for users to inspect digital signatures and verify the validity of certificates.

Once the first version of the architecture is finished, the security and usability must be analyzed. Most designers have problems "attacking" their own work. Of course independent outside experts have no such problems. An external review of outside experts (not involved in the development of the system) can therefore be highly advantageous. Note that it's much better to discover serious problems with the architecture at an early stage when it's still possible to remove the problems without incurring large costs.

If a security vulnerability or a usability problem is found, then the architecture must be modified and the altered architecture must be analyzed for new problems. This process must be repeated until no serious problems are found. Any remaining hidden security or usability flaws may lead to an expensive re-design some years into the future.

7.4 Collaborative design process

To see why the design process should also be a collaborative undertaking, we return to our PKI example. The development team should again collaborate with the PKI experts to avoid a design which violates well established PKI principles [17, 18]. The experts make sure that different pairs of public-private keys are used for authentication and digital signatures. If a high degree of non-repudiation is needed, then they suggest a design where a user's signature key is both generated and stored locally on the user's own device.

The usability aspects must be developed further. Unfortunately, users don't understand the difference between a public key and a private key. They also have problems understanding the role of public-key certificates. Finally, users don't understand the connection between the PKI and the goal they are trying to achieve by using a system [15, Ch. 16]. The usability experts can help the development team overcome these issues. Together they can design a banking specific PKI, where most of the PKI functionality is transparent to the users.

Once the initial design is finished, the security and usability must be evaluated. The development team must continue the development until no more serious problems are found.

7.5 Documents

To further encourage all team members to take part in the decision process and make them feel responsible for the complete system, the architectural and design documents should be the property of the development team rather than

individual members. To emphasize that security and usability must be an integral part of the development process, team members should not develop separate security and usability documents.

The architectural and design documents of future banking systems should be made available to the public. This will further encourage good development processes and, thus, reduce the risk of creating systems with serious security flaws.

During a conflict a customer can easily employ outside security experts to carry out an independent security analysis of a banking system. A good design should of course manage to withstand scrutiny from hostile experts.

8 Final remarks

It's difficult to develop banking systems that remain secure and usable over time. A good architecture and design require the (core) development team to collaborate with outside lawyers, security experts, usability specialists, and customers. Such collaborative development is hampered when the company policy dictates that the development team must keep all information about the system secret.

An open collaborative development process can produce banking (and other commercial) systems with better security and usability than in many current closed systems. The publicly available architectural and design documents produced during the development period can improve the customers' legal protection during conflicts involving the systems.

More work is needed to establish open development processes combining security and usability. Ideas from the open source community on how to collaborate [14] and results from security and usability research [15] provide good starting points.

We've only considered the Norwegian DES-based ATM system because our main goal was to analyze a Norwegian court case. However, we believe that much of our analysis applies to other ATM systems based on DES. It's difficult to verify a security analysis of a closed system. While we've had many discussions with banking experts to try to double-check our work, the authors remain solely responsible for the content of this paper.

References

- [1] "Verdict from Trondheim Tingrett, case number 04-016794TVI-TRON," (in Norwegian).
- [2] NIST, "Data Encryption Standard (DES)." [Online]. Available: <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>
- [3] K. J. Hole, V. Moen, and T. Tjøstheim, "Case study: Online banking security." *IEEE Security & Privacy*, vol. 4, no. 2, pp. 14–20, 2006.
- [4] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov, "Cryptographic processors—a survey," University of Cambridge, Tech. Rep. 641, 2005. [Online]. Available: <http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-641.pdf>

- [5] “ISO 9564–1 Banking—Personal Identification Number (PIN) management and security—part 1: Basic principles and requirements for online PIN handling in ATM and POS systems, 1st ed.” 1991.
- [6] “ISO 9564–2 Banking—Personal Identification Number (PIN) management and security—part 2: Approved algorithm(s) for PIN encipherment,” 1991.
- [7] EFF, “Cracking DES,” 1998. [Online]. Available: http://www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/
- [8] “RSA DES challenge III,” 1999. [Online]. Available: http://www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19990119_deschallenge3.html
- [9] M. Bond and R. Clayton, 2001. [Online]. Available: <http://www.cl.cam.ac.uk/~rnc1/descrack/>
- [10] R. Anderson, “Why cryptosystems fail, from communications of the ACM, November, 1994,” in *William Stallings, Practical Cryptography for Data Internetworks*. IEEE Computer Society Press, 1996. [Online]. Available: <http://citeseer.ist.psu.edu/anderson94why.html>
- [11] N. R. Council, *Who Goes There?* National Academies Press, 2003.
- [12] The Financial Supervisory Authority of Norway, “Risk and vulnerability analysis 2003,” 2003, (in Norwegian).
- [13] G. Fairtlough, *The Three Ways of Getting Things Done*. Triarchy Press, 2005.
- [14] C. DiBona, D. Cooper, and M. Stone, Eds., *Open Sources 2.0*. O’Reilly, 2006.
- [15] L. F. Cranor and S. Garfinkel, Eds., *Security and Usability*. O’Reilly, 2005.
- [16] G. McGraw, “Software security,” *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, 2004.
- [17] C. Adams and S. Lloyd, *Understanding PKI*, 2nd ed. Addison-Wesley, 2003.
- [18] W. Ford and M. S. Baum, *Secure Electronic Commerce*, 2nd ed. Prentice Hall, 2001.

Paper IV: Weaknesses in the Temporal Key Hash of WPA

Weaknesses in the Temporal Key Hash of WPA

Vebjørn Moen, Håvard Raddum, and Kjell J. Hole

This article describes some weaknesses in the key scheduling in Wi-Fi Protected Access (WPA) put forward to secure the IEEE standard 802.11-1999. Given a few RC4 packet keys in WPA it is possible to find the Temporal Key (TK) and the Message Integrity Check (MIC) key. This is not a practical attack on WPA, but it shows that parts of WPA are weak on their own. Using this attack it is possible to do a TK recovery attack on WPA with complexity $\mathcal{O}(2^{105})$ compared to a brute force attack with complexity $\mathcal{O}(2^{128})$.

1 Introduction

The IEEE standard 802.11-1999 [1] is a set of protocols defining a communication channel inspired by Ethernet, but using unlicensed radio spectrum bands instead of wires. Since radio is used to communicate, eavesdropping can be done by anyone with a radio receiver, and anyone with a radio transmitter can write to the channel. This shows the need for built-in security in the WLAN design. The 1999 standard includes a security protocol called Wired Equivalent Privacy, or WEP. The goal was to achieve the same level of security as wired Ethernet.

It has been shown that the WEP design has many basic flaws and does not fulfill the design goals. It does not defend properly against packet forgery or replay, which allows an attacker to use the 802.11 infrastructure to launch attacks on the WEP encryption key. In addition, WEP uses the RC4 encryption algorithm in a way that makes it possible to mount plaintext recovery attacks and key recovery attack using public domain software, *e.g.* AirSnort [2].

To correct these design flaws, the 802.11 Working Group (WG) has chartered work to find a new security protocol. First the WG has defined WPA, which is a WEP wrapper design, to fix all the known problems with WEP. It has been established that WPA cannot fulfill the original WEP design goals, because the available CPUs on existing hardware are too limited. Therefore, the WG is also working on a new protocol based on the Advanced Encryption Standard (AES) that can meet the original design goals, but it will require new hardware.

The rest of this paper is organized as follows: Section 2 describes the algorithms that are relevant to the security in WPA, Section 3 describes the attack, Section 4 discusses the practicality and impact of the results, and Section 5 gives a summary of this paper.

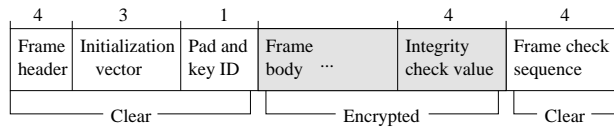


Figure 1: WEP frame. Length of fields measured in bytes.

2 Algorithms

2.1 WEP

WEP was suggested in the IEEE standard 802.11-1999 to provide security equivalent with that of a wired Ethernet. The WEP algorithm should insure confidentiality and integrity of the frames on the wireless network. A Cyclic Redundancy Check (CRC) is used to compute an Integrity Check Value (ICV) on the message. The ICV is then concatenated on the message before encrypting with the stream cipher RC4. The WEP-frame is illustrated in Figure 1.

RC4 is a symmetric cipher, *i.e.*, the same key encrypts and decrypts the data. The encryption key is a per-packet key which is obtained by concatenating an *Initialization Vector* (IV) with the user key. Because of export regulations the standard specifies 64-bit keys where 24 bits are the IV, but many vendors have also implemented 128-bit keys where 24 bits are the IV.

2.1.1 Security issues

The CRC used for the ICV can catch single-bit alterations with high probability, but it is not cryptographically secure. The CRC is a linear function of the message, and Borisov *et al.* [3] showed that it is possible to make controlled modifications to a ciphertext without disrupting the checksum.

The standard ignores the issue of key management. Most vendors do not implement any key distribution mechanism, this means that keys must be statically entered into either the driver software or firmware. All the mobile stations accessing the same access point use the same pre-shared key and can therefore decrypt each others packets. Since the key needs to be manually distributed and typed into a device, it is not likely that the key will be changed very often. The IV is only 24 bits long, which implies that the same key and IV will be reused, this is known as the two-time pad [3].

Fluhrer *et al.* [4] also found a correlation between the combination of the IV and user key with the first RC4 key stream byte, which leads to a practical key recovery attack.

2.2 Wi-Fi Protected Access

Because WEP has been shown to be totally insecure, the 802.11 WG has suggested a new security protocol. The protocol is called Wi-Fi Protected Access (WPA). The goal for this protocol is to fix all known security flaws in WEP and it was designed to be deployed as a software patch on existing hardware.

WPA includes a key hash function [5] to defend against the Fluhrer *et al.* [4] attack, a Message Integrity Code (MIC) [6] and a key management scheme based

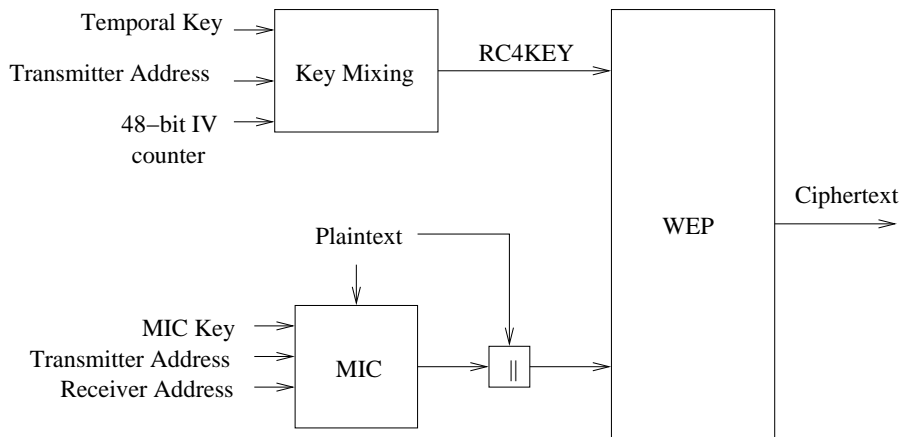


Figure 2: WPA encapsulation process

on 802.1X [7] to avoid key reuse and to ease the key distribution. Figure 2 shows the encapsulation process.

The 16-byte *Temporal Key* (TK) is obtained from the key management scheme during the authentication, and goes into the key hash function together with the 6-byte *Transmitter Address* (TA) and a 48-bit IV, often called the TKIP sequence counter. The key hash function outputs a 16-byte RC4 key where the three first bytes are derived from the IV. This key is used only for one WEP frame, since the IV is implemented as a counter which increases after each package, and the key is therefore often called a per-package key. The IV counter also works as a defense against replay attacks, the receiver will not accept packets with smaller or equal IV to previously received packets.

Integrity of the message is insured by the MIC. This function takes as input a MIC key, TA, receiver address, and the message, and outputs the message concatenated with a MIC-tag. If necessary this output is fragmented before it enters WEP.

This means that WPA is a wrapper for WEP insuring that a $\langle \text{TK}, \text{IV} \rangle$ pair is only used once by a sender, and improving the integrity of WEP frames by applying a non-linear message integrity function. More details about the MIC can be found in [6].

2.2.1 Key mixing

The key mixing function is described by Housley *et al.* [5], this function is also called a *temporal key hash*. As shown on Figure 2, this function takes as input the TK, the TA and the 48-bit IV, and outputs a 128-bit WEP key where 24 bits are derived from the IV. The least significant 16 bits of the 48-bit IV are denoted IV16, and 32 most significant bits are denoted IV32. The key mixing is a two-phase process which may be summarized as:

$$\begin{aligned}
 P1K &= \text{Phase1}(TK, TA, IV32) \\
 RC4KEY &= \text{Phase2}(P1K, TK, IV16)
 \end{aligned}$$

```

PHASE1_STEP1:
P1K[0] = Lo16(IV32)
P1K[1] = Hi16(IV32)
P1K[2] = Mk16(TA[1],TA[0])
P1K[3] = Mk16(TA[3],TA[2])
P1K[4] = Mk16(TA[5],TA[4])

PHASE1_STEP2:
FOR i = 0 to 7
BEGIN
  j = 2*(i & 1)
  P1K[0] = P1K[0] + S[P1K[4]  $\oplus$  Mk16(TK[ 1+j],TK[ 0+j])]
  P1K[1] = P1K[1] + S[P1K[0]  $\oplus$  Mk16(TK[ 5+j],TK[ 4+j])]
  P1K[2] = P1K[2] + S[P1K[1]  $\oplus$  Mk16(TK[ 9+j],TK[ 8+j])]
  P1K[3] = P1K[3] + S[P1K[2]  $\oplus$  Mk16(TK[13+j],TK[12+j])]
  P1K[4] = P1K[4] + S[P1K[3]  $\oplus$  Mk16(TK[ 1+j],TK[ 0+j])] + i
END

```

Algorithm 1: Phase1 of Temporal Key hash

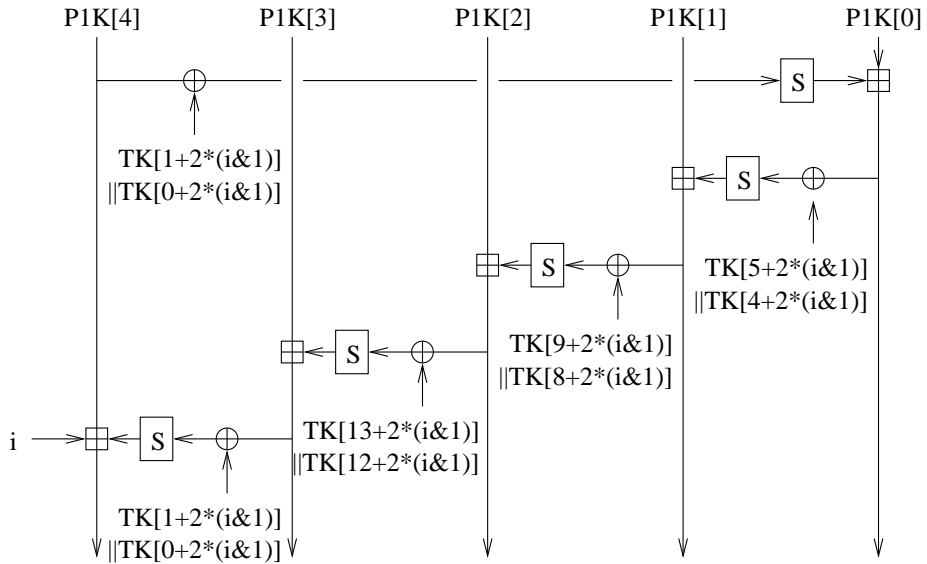


Figure 3: One of 8 rounds of the first phase of temporal key hash. This round is repeated for $i = 0..7$.

Phase 1 is shown in Algorithm 1 and in Figure 3. This part is usually only done once every 2^{16} packets and cached. It takes TK, TA and IV32 as input and outputs P1K used as input in Phase 2.

```

PHASE2_STEP1:
PPK[0] = P1K[0]
PPK[1] = P1K[1]
PPK[2] = P1K[2]
PPK[3] = P1K[3]
PPK[4] = P1K[4]
PPK[5] = P1K[4] + IV16

PHASE2_STEP2:
PPK[0] = PPK[0] + S[PPK[5]  $\oplus$  Mk16(TK[ 1],TK[ 0])]
PPK[1] = PPK[1] + S[PPK[0]  $\oplus$  Mk16(TK[ 3],TK[ 2])]
PPK[2] = PPK[2] + S[PPK[1]  $\oplus$  Mk16(TK[ 5],TK[ 4])]
PPK[3] = PPK[3] + S[PPK[2]  $\oplus$  Mk16(TK[ 7],TK[ 6])]
PPK[4] = PPK[4] + S[PPK[3]  $\oplus$  Mk16(TK[ 9],TK[ 8])]
PPK[5] = PPK[5] + S[PPK[4]  $\oplus$  Mk16(TK[11],TK[10])]

PPK[0] = PPK[0] + RotR1(PPK[5]  $\oplus$  Mk16(TK[13],TK[12]))
PPK[1] = PPK[1] + RotR1(PPK[0]  $\oplus$  Mk16(TK[15],TK[14]))
PPK[2] = PPK[2] + RotR1(PPK[1])
PPK[3] = PPK[3] + RotR1(PPK[2])
PPK[4] = PPK[4] + RotR1(PPK[3])
PPK[5] = PPK[5] + RotR1(PPK[4])

PHASE2_STEP3:
RC4KEY[0] = Hi8(IV16)
RC4KEY[1] = (Hi8(IV16) | 0x20) & 0x7F RC4KEY[2] = Lo8(IV16)
RC4KEY[3] = Lo8((PPK[5]  $\oplus$  Mk16(TK[1],TK[0])) >> 1)

FOR i = 0 to 5
BEGIN
  RC4KEY[4+(2*i)] = Lo8(PPK[i])
  RC4KEY[5+(2*i)] = Hi8(PPK[i])
END

```

Algorithm 2: Phase2 of the temporal key hash

Phase 2 takes the output from Phase1, TK and IV16 as input, and outputs the 128-bit WEP key. Phase 2 is described in Algorithm 2 and in Figure 4. Note that the TK is viewed as an array [0..15] of 8-bit bytes.

The S-box is a bijective nonlinear function defined by a table lookup in [5], it takes a 16-bit input and outputs a 16-bit value. The Mk16 function takes two 8-bit inputs and produces a 16-bit word, such that $Mk16(X,Y) = 256 * X + Y$ which is equivalent to $Mk16(X,Y) = X||Y$. Hi16 takes a 32-bit input and returns the most significant 16 bits, Lo16 takes a 32-bit input and returns the least significant 16 bits. Hi8 and Lo8 are similar but with input size 16 bits and output size 8 bits. Furthermore, & denotes bit-wise logical AND, and | represents bit-wise logical OR. Also, note that + in the Algorithms 1 and 2, and

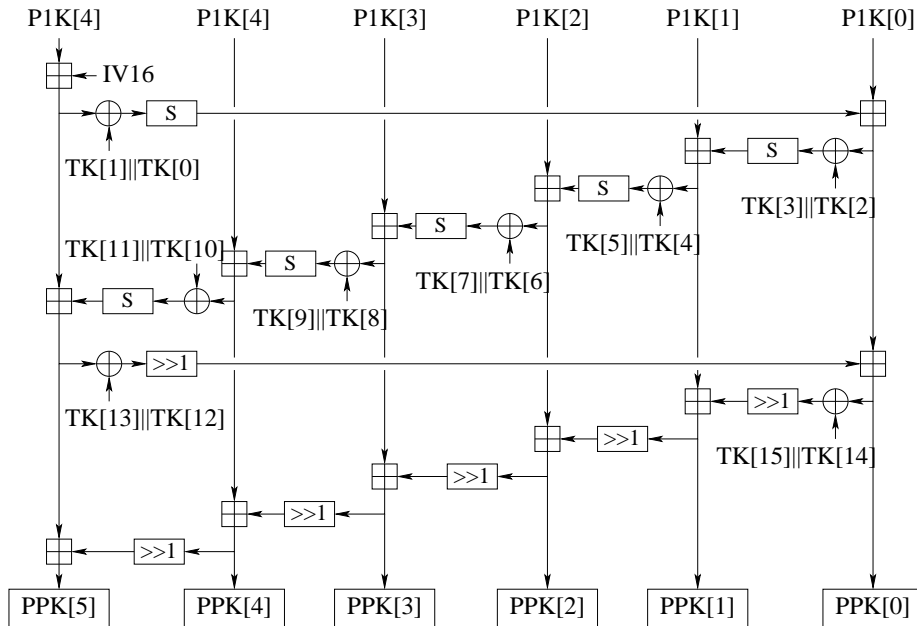


Figure 4: Phase 2 of the temporal key hash. The 96-bit PPK is used together with 16 bits of the IV and 8 bits of the TK to create the 128-bit WEP key.

\boxplus in Figures 3 and 4 are addition modulo 2^{16} ; \oplus represents bit-wise exclusive OR. Both RotR1 and $\gg 1$ denote right circular shift by 1. P1K and PPK are treated as arrays of 16 bit words and RC4KEY is treated as an array of 8-bit bytes.

3 Attacking Temporal Key Hash

This section describes an attack on the temporal key hash described above. It is assumed that the attacker has knowledge of a few (less than 10) RC4-keys computed under the same IV32. Whether this is a realistic assumption or not will be discussed in Section 4.

Under this assumption we show that the attacker can easily compute the TK, and thus decrypt any packet the same way the legitimate receiver does. The attack has a complexity of about 2^{32} simple operations, and takes a few minutes to execute on a normal modern PC. The attack is basically done by computing backwards through Phase 2, guessing on parts of the TK. We can check if a guess is right or wrong since we know that the P1K-values do not change before IV32 changes.

The attack makes use of the fact that eight bits of TK can be computed directly from an RC4-key. The PPK-values output from Phase 2 are known from the RC4-key, in particular PPK[5] is known. By looking at Step 3 of Phase 2 in Algorithm 2 we see how RC4KEY[3] is computed. It is then easy to reveal the least significant bit of TK[1], and the seven most significant bits of TK[0].

The rest of this section describes the attack in detail, showing how we can

break the rest of the TK into six parts, and guess on one part at the time. In the diagrams below, thick lines indicates that we know the values carried on them, and dotted lines indicate that there are two choices for the values on the lines.

3.1 Finding TK[10] and TK[11]

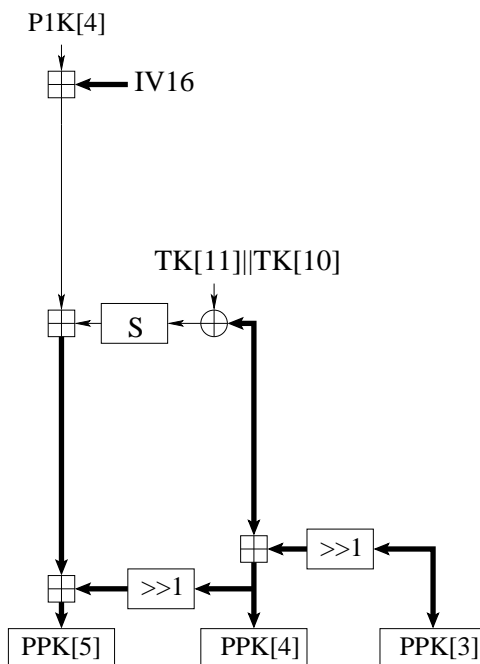


Figure 5: Part of Phase 2 needed to compute TK[10] and TK[11] .

Figure 5 is cut out from Figure 4 of Phase 2 of the key hash. The attack is based on the divide-and-conquer technique as illustrated in the figure. The idea is to find some bytes of the TK at a time, and Figure 5 shows the parts of PPK and TK which are needed to calculate backwards to P1K[4]. Since PPK[3], PPK[4] and PPK[5] are known we can start backtracking Phase 2. The arrows show what is input and output.

The first thing we need to do is to right shift PPK[3] and PPK[4] by one, and this is straight forward. The inverse of addition modulo 2^{32} is subtraction modulo 2^{32} . Using this we can compute backwards up to the point where the values depend on TK[10] and TK[11]. Now we guess on the value of TK[11]||TK[10], which allows us to backtrack through the XOR and S-box and two additions modulo 2^{32} . Remember that the IV is a known value, sent in the clear in the WPA packet. For each guess of TK[11]||TK[10] we get a suggestion for P1K[4]. We repeat the above procedure for each RC4-key that we are using in the attack, and if different RC4-keys give different P1K[4]-values, the guess was wrong. Using two or three RC4-keys should be enough to eliminate all but the correct values of TK[10] and TK[11].

3.2 Finding TK[8] and TK[9]

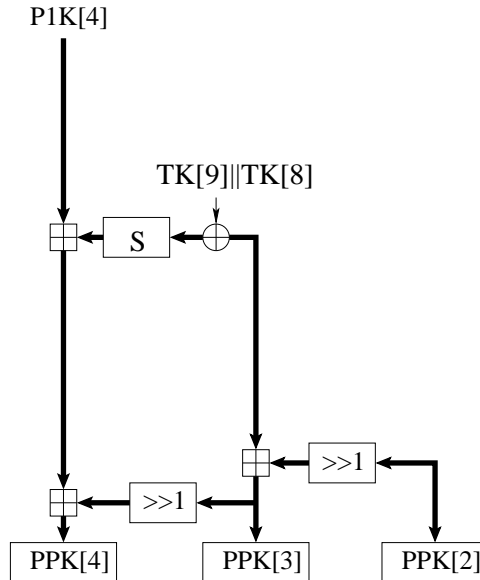


Figure 6: Part of Phase 2 needed to determine TK[8] and TK[9].

Figure 6 shows the part of Phase 2 necessary to compute TK[8] and TK[9]. Note that when we found TK[10] and TK[11] during the previous step, we also found the correct value of P1K[4]. Therefore, TK[8] and TK[9] can be computed directly, without any guessing.

3.3 Finding TK[6] and TK[7]

As Figure 7 shows, TK[6] and TK[7] can be found exactly the same way as TK[10] and TK[11] were found.

3.4 Finding TK[0], TK[1], TK[12] and TK[13]

Consider the part of Phase 2 shown in Figure 8. Here we take advantage of the fact that only eight bits of TK[0] and TK[1] are unknown. In order to compute the value of P1K[0], we can again make use of the now known value of P1K[4], but this time it is necessary to guess on TK[12], TK[13], and the eight unknown bits of TK[0] and TK[1] to reconstruct a candidate for P1K[0], a total of 24 bits. Again, if we don't get the same value of P1K[0] for all RC4-keys, we can discard the current guess as wrong.

At this stage, a subtle point comes into play. Assume we take the correct values of TK[0], TK[1], TK[12] and TK[13], but flip the least significant bit of TK[12]. This is a wrong guess, and should be discarded given sufficiently many

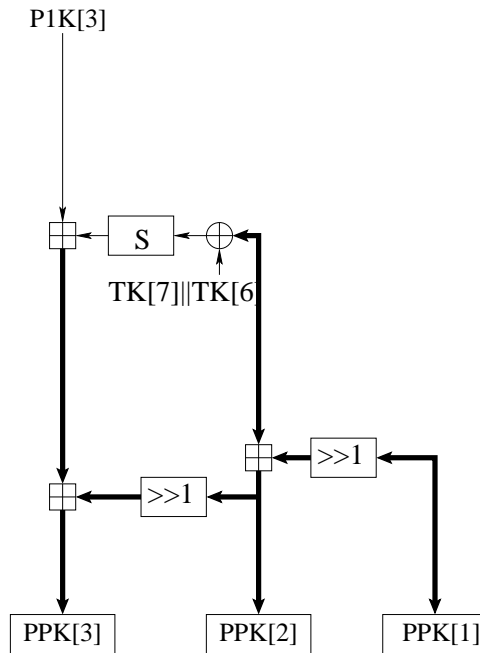


Figure 7: Part of Phase 2 needed to calculate TK[6] and TK[7]

RC4-keys. We will compare the values obtained in Figure 8 using this guess, to the values obtained with the correct guess. The guess for TK[0] and TK[1] is the correct one, so the values on the horizontal wire on the top will be equal in both cases. Going through the rotation in the bottom half, the values will differ only in the most significant bit. This is the same as saying that they differ by $2^{15} \pmod{2^{16}}$. Since the remaining operations for computing P1K[0] are subtraction mod 2^{16} , the computed values for P1K[0] with this wrong key guess will differ in the most significant bit from the correct value of P1K[0]. In particular, the P1K[0]-values computed with the wrong guess will all be equal! This means that the least significant bit of TK[12] can not be determined at this point using our method, however, it is easily determined later. It also means that P1K[0] is not determined completely, we only know the low 15 bits.

3.5 Finding TK[2], TK[3], TK[14] and TK[15]

Figure 9 shows the most expensive part of the attack. Here we will guess on TK[2], TK[3], TK[14] and TK[15] at the same time. For each guess we will compute the values of P1K[1] and check if they are equal. The matter is slightly complicated by the fact that we do not know P1K[0] completely. P1K[0] can take one of two values, so we have to do the check on P1K[1] for both values. For the correct guess of TK[14] and TK[15], there will be two values of TK[3]||TK[2] suggested, one for each P1K[0].

The problem with the least significant bit occurs here too, we can not find the least significant bit of TK[14]. This means we do not need to guess on it

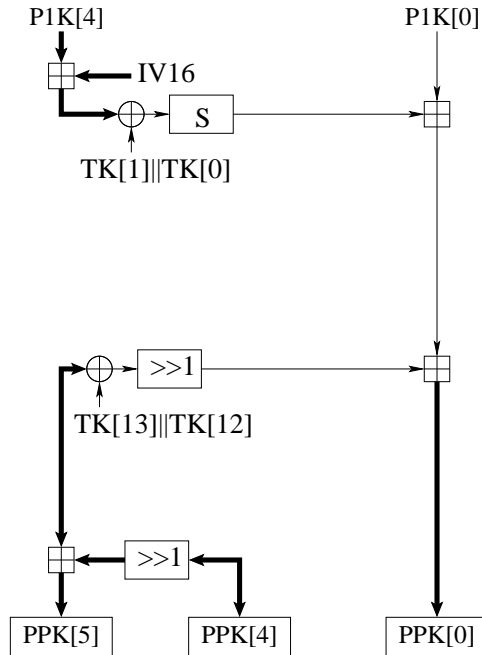


Figure 8: Part of Phase 2 needed to compute TK[0], TK[1], TK[12] and TK[13]

either, so there will be a total of 31 bits guessed. For each guess we will have to do the check on P1K[1] twice, once for each of the two possible values of P1K[0], so the overall complexity is 2^{32} checks. This step dominates the overall complexity of the attack. After this we are left with four sets of possible values of TK[2], TK[3], TK[12], and TK[14].

3.6 Finding TK[4] and TK[5]

Finally, Figure 10 shows how TK[4] and TK[5] are found by checking on the P1K[2]-values. Each guess also includes guessing on the least significant bit of TK[14]. For each of the two possibilities of TK[14] there will be one TK[5]- and TK[4]-value suggested as correct.

3.7 Finding the least significant bits of TK[12] and TK[14]

After completing all six steps described above, we are left with four possible values for the whole TK. Each possible TK has its corresponding P1K-value. The correct TK can now be found by running Phase 1 for each of the TK candidates, and see which one that gives its corresponding P1K as output. The probability that a wrong TK results in its corresponding P1K is $4 * 2^{-80} = 2^{-78}$ since P1K is 80 bits.

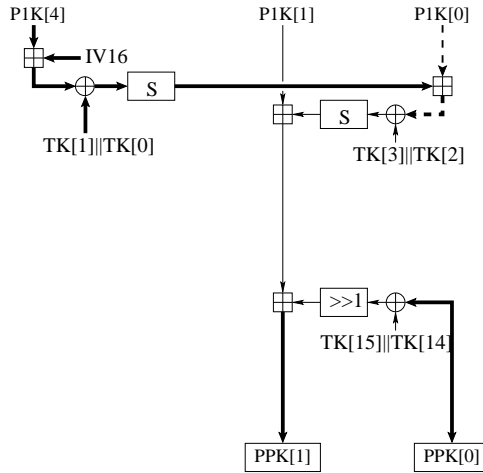


Figure 9: Part of Phase 2 needed to calculate TK[2], TK[3], TK[14] and TK[15]

3.8 Attacking with only two RC4-keys

When the attacker only has two RC4-keys, she only has a 16-bit condition for eliminating wrong guesses at each step. In the step with TK[0], TK[1], TK[12] and TK[13], we are guessing on 23 bits, so we expect to have about 2^7 candidates suggested for these four bytes. In the step with TK[2], TK[3], TK[14] and TK[15] we are guessing on 31 bits, so we expect to be left with 2^{15} candidates for this part of the TK. In the other steps we are guessing on 16 bits, so we expect to be only left with the correct TK-parts. With the four possible variations of the least significant bit of TK[12] and TK[14], this gives us $4 \cdot 2^7 \cdot 2^{15} = 2^{24}$ candidates for the whole TK. Each candidate has its corresponding set of P1K-values. Now we can run Phase 1 for each candidate and see which one has matching suggestions for the P1K value from both Phase1 and Phase2. The probability of a wrong TK to result in corresponding P1K values is $2^{24} \cdot 2^{-80} = 2^{-56}$, so with high probability only the correct TK will pass this test. Total work with only two texts is approximately $\mathcal{O}(2^{38})$, since we need to guess on 31 bits for each of the 2^7 suggestions for TK[0], TK[1], TK[12] and TK[13].

3.9 Temporal Key recovery attack on WPA

The results in this paper imply that it is possible to mount a Temporal Key recovery attack on WPA with time complexity $\mathcal{O}(2^{105})$ compared to simply brute force search on the TK, which has time complexity $\mathcal{O}(2^{128})$. The idea of this attack is simply to brute force two distinct RC4 keys with 104 unknown bits in each and then apply the attack described in this paper to recover the 128-bit Temporal Key and the 64-bit message authentication key with additional $\mathcal{O}(2^{38})$ effort. This attack has time complexity $\mathcal{O}(2^{105})$ which still is not practical, but it is a significant reduction.

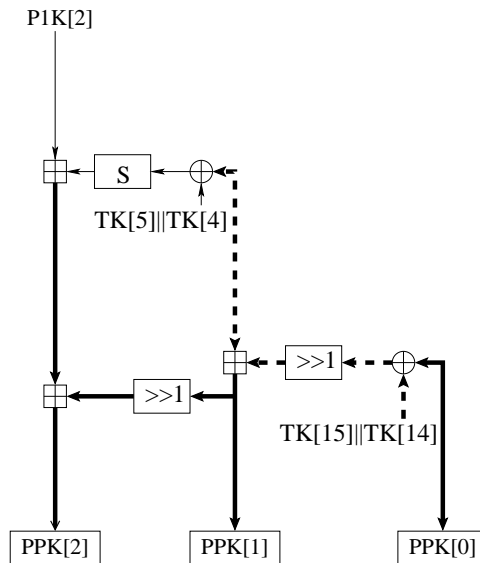


Figure 10: Part of Phase 2 needed to compute TK[4] and TK[5].

4 Discussion of the attack

The attack described in the previous section has been implemented on a computer to verify its correctness. The processor used is an Intel Pentium 4 2.53 GHz, and it takes about 6-7 minutes to recover the TK given four or more RC4-keys. Given only 2 texts the work is a factor 2^7 bigger, which gives a running time of approximately 15 hours.

This makes it a highly practical attack, but if the results of this paper shall have any impact, the important question is: “How likely is it that an attacker gets two RC4-keys generated under the same IV32?” The answer to this question depends on the implementation. The main contribution of this paper is therefore to highlight the weak spots of WPA.

4.1 Loss of RC4-key = total loss of security

The most important thing to keep in mind for an implementor is that RC4-keys used to encrypt packets, and the TK are equally important to keep secret. As this paper shows, the loss of a few RC4-keys allows the attacker to recover the long term secret TK, and not only the contents of the compromised packets.

In [6], describing the details of the MIC part of WPA, it is stressed that the integrity of a packet relies on the fact that each packet is encrypted. The author points out that the loss of one RC4-key allows the attacker to recover the MIC-key, and thus produce a valid MIC-tag to a packet of her choice. However, the attacker needs to block the receiver from receiving this and any subsequent packets until the modified packet has been inserted on the channel. Also, the attacker can only modify the particular compromised packet, since the other packets are protected by encryption using other keys. This seems to limit the threat to that of an active adversary with capabilities of blocking the receiver, being able to modify compromised packets.

This paper shows that the compromise of two or more RC4-keys is much more serious. The attacker may recover all secret keys the user has, and can therefore perform any action the user can do. In particular, WPA provides no forward secrecy since the attacker can construct earlier RC4-keys, as well as future ones, once some keys have leaked.

4.2 Cut-and-paste cryptographic primitives

It is quite common to take (parts of) existing cryptographic primitives and construct new cryptographic algorithms from them. For example, WPA uses RC4 for encryption, and parts of the AES round function in the temporal key hash. Ferguson [6] warns that the MIC function used in WPA is only secure in this particular setting. We would like to issue the same warning when it comes to temporal key hash: it is no good as a hash function, but only as a key generator.

4.3 Possible attack scenario

It is possible to imagine a situation where a leader of a group is communicating on behalf of the whole group. Some parts of the information the leader receives he wants to keep to himself, while other parts should be broadcast to all the group members. For instance, a review form for conference submissions often contains the fields “comments to program chair only” and “comments to program committee”. Since WPA is a wireless network, all members of the group can receive the packets broadcast from the access point, but only the leader holding the TK can decrypt the packets. Some of the packets are for the whole group to read. The leader can choose to broadcast the contents of these packets to the group, but since the group members can receive the encrypted versions of these packets themselves, a cheaper way would be to just broadcast the RC4-keys for the packets in question, and let each member do the decryption himself.

Someone not aware of the shortcomings of the temporal key hash might opt for this solution, not knowing this allows the whole communication to be read by everyone.

5 Summary

We have shown the whole security in WPA relies on the secrecy of all packet keys. Given one packet key it is possible to find the MIC key and given two packet keys with the same IV32 an attacker can do anything the legitimate user can, for the duration of the TK.

Since these packet keys are supposed to be kept secret, the attack in this paper does not imply that WPA is broken, but it underlines the importance of keeping each and every packet key secret.

References

- [1] IEEE 802.11 WG, *802.11b: Standards for Local and Metropolitan Area Networks: Wireless Lan Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE, 1999.

- [2] Airsnort, last visited: June 12th, 2006. [Online]. Available: airsnort.shmoo.com
- [3] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting mobile communications: The insecurity of 802.11," in *MOBICOM 2001*, 2001.
- [4] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4," *Lecture Notes in Computer Science*, vol. 2259, pp. 1–24, 2001. [Online]. Available: citeseer.ist.psu.edu/fluhrer01weaknesses.html
- [5] R. Housley, D. Whiting, and N. Ferguson, "Alternate temporal key hash," IEEE doc. 802.11-02/282r2, 2002.
- [6] N. Ferguson, "Michael: an improved MIC for 802.11 WEP," IEEE doc. 802.11-2/020r0, 2002.
- [7] IEEE 802.1 WG, *802.1x: Standards for Local and Metropolitan Area Networks: Port-Based Access Control*. IEEE, 2001.

**Paper V: Attack on Sun's
MIDP Reference
Implementation of SSL**

Attack on Sun's MIDP Reference Implementation of SSL

Kent Inge Simonsen, Vebjørn Moen, and Kjell Jørgen Hole

Abstract

Key generation on resource-constrained devices is a challenging task. This paper describes a proof-of-concept implementation of an attack on Sun's reference implementation of the Mobile Information Device Profile (MIDP). It is known that this implementation has a flaw in the generation of the premaster secret in SSL. The attack recovers the symmetric keys and plaintext from an SSL session.

1 Introduction

Running Java programs on resource-constrained devices like cellular phones and personal digital assistants require a specialized run-time environment. The Connected Limited Device Configuration (CLDC) [1] provides a set of Application Programming Interfaces (APIs) and a virtual machine for this environment. Together with a profile such as the Mobile Information Device Profile (MIDP) [2], it provides the possibility to develop Java applications to run on devices with limited memory, processing power, and graphical capabilities.

MIDP is a collection of APIs building on CLDC, providing some more advanced capabilities. Applications that comply with this standard are called MIDlets. Many companies have been involved in the development of MIDP, including Ericsson, NEC, Nokia, Palm Computing, Research In Motion (RIM), DoCoMo, LG TeleCom, Samsung, and Motorola.

MIDP has support for the Hyper Text Transfer Protocol (HTTP), where the information is sent in the clear, and secure HTTP, denoted HTTPS, which supports authentication, confidentiality, and integrity. The security of HTTPS is provided by Secure Socket Layer (SSL), or its successor Transport Layer Security (TLS).

As with many other cryptographic protocols, the security of SSL and TLS depends on generating secret key material. The randomness used in the process of generating the key material decides the strength of the resulting keys.

The first version of SSL in Netscape was shown to create key material using time [3] as input to a Pseudo-Random Number Generator (PRNG); this input is called a *seed*. Seeding with time is a common mistake, since it is difficult to get access to a good seed on a general purpose computer. Creating truly random numbers on a deterministic device such as a computer is impossible. We need to access a hardware source to get some randomness—strong sources of randomness include thermal noise and a radioactive decay source. Creating good random numbers in a constrained environment such as a cellular phone is

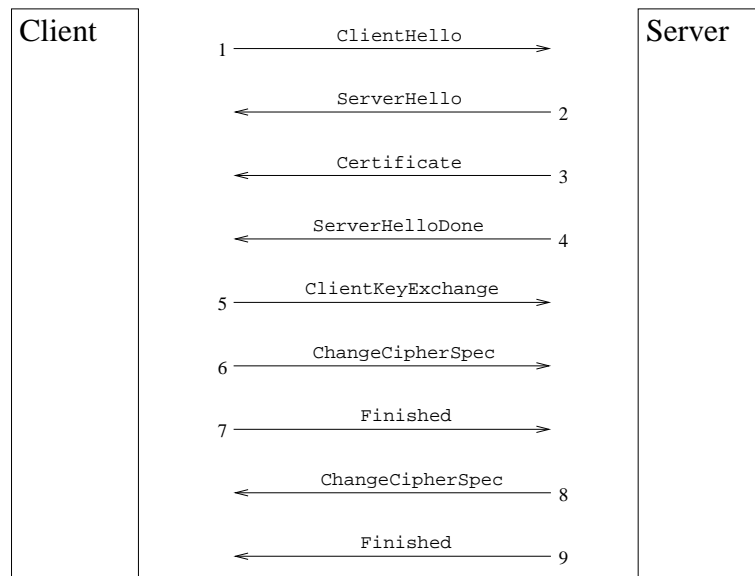


Figure 1: The 9 messages that SSL uses to establish an encrypted communication channel.

truly a challenge, but the security in SSL and most other crypto systems depend on a source for randomness.

It is known [4] that the reference implementation of MIDP provided by Sun has a flaw in the generation of the *premaster secret*, from which the message authentication and encryption keys in SSL are derived, due to seeding a PRNG with time. We describe an implementation of an attack on an SSL session between a server and a client using Sun’s MIDP reference implementation which successfully recovers the SSL premaster secret, and consequently the authentication and encryption keys used in the SSL session.

In Section 2 we give a brief introduction to SSL, Section 3 considers randomness, Section 4 describes the attack on SSL in MIDP, as well as the implementation of the attack, and Section 5 concludes the paper.

2 SSL

This section is not meant to give a complete description of the SSL protocol; for a complete description [5] is recommended. We will consider the simplest case of SSL, namely establishing an encrypted communications channel.

The situation is that a client wants to establish a secure session with a server. To do this the client and server exchange SSL messages. Figure 1 shows the SSL handshake used to establish a share secret.

1. **ClientHello**: The client asks the server to begin the negotiation of the security services used by SSL. This message contains fields for a version number (3.0 for SSLv3 and 3.1 for TLS), and a 32-byte nonce used as seed in the generation of the premaster secret. The SSL specification suggests that 4 of these 32 bytes contain the time and date to avoid client reuse

of this 32-byte random number. A session ID to identify the specific SSL session, a list of cryptographic primitives that the client can support, and some more fields not mentioned here are also a part of the `ClientHello`.

2. `ServerHello`: The server responds to the `ClientHello`. This message contains fields for a version number, a 32-byte nonce where 4 bytes are used for time and date, a session ID number, a `CipherSuite` field which determines the cryptographic parameters, such as algorithms and key sizes. The `ServerHello` also contains some more fields not discussed here.
3. `Certificate`: The server sends a certificate containing the public key information.
4. `ServerHelloDone`: Tells the client that the server is finished with the initial negotiation messages.
5. `ClientKeyExchange`: The client generates the premaster secret, encrypts it with the public key received in the server certificate and sends the result to the server.
6. `ChangeCipherSpec`: This message tells the server that from now on any message received from the client will be encrypted with the agreed algorithm and key.
7. `Finished`: This message from the client to the server allows the server to verify that the negotiation has been successful. It contains a hash of key information, and contents of all previous SSL handshake messages exchanged by the client and server. Also notice that this message is encrypted.
8. `ChangeCipherSpec`: This message tells the client that from now on all messages from the server will be encrypted.
9. `Finished`: The client can now verify that the SSL negotiation has been successful. Just as for the finished message from the client it contains a hash of key information, and contents of all previous SSL handshake messages, and it is also encrypted.

After finishing the above protocol the client and the server share symmetric keys for message authentication and encryption, and using the certificate received from the server in message 3 the client can verify that it is talking to the correct server. Note however that the described SSL negotiation does not allow the server to authenticate the client. Observe also that “`Finished`” messages can be used by the server and the client to verify that the other part has the correct key.

3 Randomness and PRNGs

The security of SSL rests on the infeasibility of testing all possible keys used for encryption. If the key space is too large, then the brute-force attack will take too much time. But if an attacker can reduce the number of keys to be tested, she might be able to crack the key.

Many applications use easily available sources of randomness to create an initial value, or seed. This seed is then used as input to a PRNG. The PRNG expands the seed into a longer, random-looking bit stream. For a non-security application the seed only needs to change every time the program runs, but when we use it to generate cryptographic keys, the seed also needs to be as unpredictable and unguessable as the key itself for an attacker.

Consider a system using 128-bit keys. A brute-force attack on such a system would need to check on average 2^{127} keys, which is a huge number and clearly infeasible on a modern computer. What happens if these 128 bits are generated with a PRNG? Assuming that all the details about the PRNG are known to the attacker, the security of the cryptographic key now depends upon the seed. In other words, the number of possible seeds gives the number of possible cryptographic keys. If the PRNG is seeded with milliseconds since midnight, January 1, 1970 in the GMT timezone, and the attacker knows which year the seed is created, she only needs to check $365 \cdot 24 \cdot 3,600 \cdot 1,000 = 31,536,000,000 \approx 2^{35}$ different keys, which is a relatively small task for a modern computer.

Using PRNGs to create cryptographic keys requires that there exists at least as many equally likely seeds as possible keys, to avoid that the PRNG reduces the effective key length.

3.1 Creating a seed

The seed is essential for the security of the system. RFC 4086 [6] gives some recommendations for security in randomness. Essentially there are two strategies: either use a reliable hardware source of randomness or use a mixing function to combine several more or less random inputs to create a “pool” of random data, e.g. Yarrow [7] and */dev/random* in GNU/Linux.

Radioactivity decay source, Gaussian white noise and spinning disks [6, 8] are all examples of hardware sources of randomness. A small addition in hardware and software to access these sources, could solve the seed problem.

The */dev/random* in GNU/Linux is an RNG which collects environmental noise from devices and other sources into an entropy pool, and keeps an estimate of the number of available bits in the entropy pool. When random numbers are requested they are created from the pool. Gutmann [9] describes some practical solutions of how to create random numbers for use in cryptographical protocols and for key material.

4 The Attack

The source code for Sun’s reference implementation of MIDP is available for download from Sun, but it does not contain the source code for SSL and the PRNG. By decompiling the SSL.jar which comes with the compiled version of MIDP we obtained the Java byte code, and from that we discovered how the seeding of the PRNG is implemented.

The PRNG is seeded with the current time in milliseconds and 16 static bytes. The PRNG also allows manual seeding, but this is not used in the reference implementation. First, we give a brief overview of how the PRNG works and what the idea of the attack is, then more details are given in the remainder of the section.

The PRNG uses the MD5 hash function to mix input and the current state, and it is reseeded with current time and the previous seed for each block of data that is generated. The entire MD5 output is used, which gives a block size of 16 bytes.

During the SSL handshake a PRNG object is constructed on the client. The PRNG object generates a 32-byte nonce sent in the clear, as well as a 48-byte premaster secret which is sent encrypted. The first two bytes of the 48 bytes used for the premaster secret are discarded to make room for some version information.

The PRNG is seeded 5 times with time in milliseconds, and one can be certain that all the time seeds come in proximity of each other. Since the nonces are sent in the clear, it seems reasonable to split the process in two parts. First, the time seeds used to create the client nonce are found so that we can synchronize our clock with the clock on the device, and then we guess the next three time seeds that lead to the premaster secret.

For each suggestion for the premaster secret we need to generate the encryption/decryption and message authentication keys, decrypt a package and check the Message Authentication Code (MAC) value.

4.1 The PRNG

The handshake procedure uses the same PRNG object to create the nonce and the premaster secret. The pseudo code version of the decompiled PRNG from Sun's reference implementation of MIDP is shown in Figure 2.

When the PRNG is constructed it initializes the MD5 digest and the `updateSeed()` method is called, where a time seed together with a constant are used to create the first state. The `updateSeed()` method feeds the current state and the current time in milliseconds in that order and calls the `doFinal()` method whose output is the next state. The digest is reset after every `doFinal()`.

The `generateData()` method writes the pseudo random data to an array (which it takes as an argument). When it runs out of random data, every 16 bytes, it digests the current state and calls the `updateSeed()` method. The data resulting from hashing the current state is said to be the pseudo random data, and is written to the array until it is full, or more data is needed. Note that `randomBytes` is a global array.

The generation of the nonce and premaster in the MIDP SSL is illustrated in Figure 3. The client generates 5 different 16-byte values with this PRNG, the first two outputs are used for the known nonce and the three next outputs are used for the unknown premaster secret. To generate the first 16-byte, a 16-byte constant and current time in milliseconds are hashed and the output is the first state, which again is hashed to yield the first 16-byte of output. At the same time the state and current time in milliseconds are digested and the output is the next state. The next four outputs needed to create the nonce and premaster secret, are generated in a similar manner; digest the state to get the output, and digest the state together with current time to get the next state.

4.2 The attack step-by-step

1. Sniff an SSL session and record the starting time.

```

constructor() {
    initialize digest;
    updateSeed();
}
updateSeed() {
    digest.update(seed);
    digest.update(currentTimeMillis);
    seed = digest.doFinal();
}
generateData(byte[] buf, int off, int len) {
    int i = 0;
    int bytesAvailable = 16;
    while(true) {
        if(bytesAvailable == 0) {
            randomBytes = digest.doFinal(seed);
            updateSeed();
            bytesAvailable = 16;
        }
        while(bytesAvailable > 0) {
            if (i == len)
                return;
            buf[off+i] = randomBytes[--bytesAvailable];
            i++;
        }
    }
}
}

```

Figure 2: Pseudo code of the PRNG from Sun's reference implementation of MIDP.

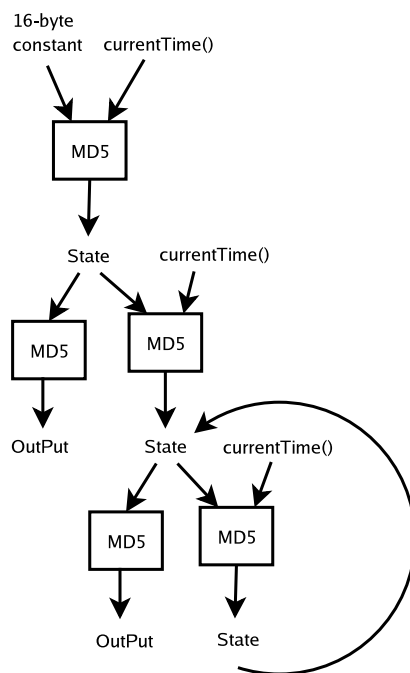


Figure 3: How MD5 is utilized to generate the pseudo random data used for nonce and premaster in the reference implementation of MIDP SSL. The 16-byte constant is known from the decompiled Java byte code.

2. Retrieve the client nonce and the server nonce. These are sent in the clear in the `ClientHello` and `ServerHello` messages.
3. Decide the start and stop time, i.e., in which time interval did the client seed the PRNG.
4. Since the client nonce is sent in clear, we know the first and second output of the PRNG. Find the value between start and stop time that was used to create the first 16 bytes of the client nonce by trying all possible values.
5. When the time seed that were used to generated the first 16 bytes is found, the PRNG can be set in the correct state. Then try all possible time seeds from the start time until the stop time, until the next 16 bytes of the nonce is found.
6. We now know exactly when the client's nonce was created according to the clients internal clock. Using this information we try to find the premaster secret which the client generates a short time after creating the nonce. Exactly how short this time is, is determined by the client device, its load, the speed of the network connection and many such factors. The amount of uncertainty about the time period in which the premaster secret is generated affects the complexity of the search for the premaster secret. Use the time seeds found in step 4 and 5 to set the state of the PRNG, then generate all possible values for the next three time seeds. Then use the suggested values together with the client nonce and server nonce to generate a candidate for the premaster secret and check if it is correct.

```

for each t1 in time interval
  for each t2 in time interval  $\geq$  t1
    for each t3 in time interval  $\geq$  t2
      premaster = generatePreMasterCandidate(
                    PRNG_state,t1,t2,t3)
      check(premaster)

```

4.3 Checking the premaster

There are several approaches to check if the suggested premaster secret is correct. One good suggestion is to create the keys used in SSL (encryption and message authentication keys) based on the premaster secret. Then we decrypt a package and attempt to verify the MAC. If the MAC verifies, we have a suggestion for the premaster secret. Any false positives can be eliminated by using more packets and MACs.

One other method is to use the `Finished` packets in the SSL handshake protocol, which contain a hash of the key material together with other known data. Yet another method could be a known plaintext attack on an SSL connection.

4.4 Time complexity

Given a start time t_{start} and finished time t_{stop} then $\Delta t = t_{stop} - t_{start}$ denotes how many milliseconds the SSL handshake takes on the device we are attacking. Using the client nonce and guessing the first time seed of the PRNG takes $\mathcal{O}(\Delta t)$ time, and guessing the second time seed also takes $\mathcal{O}(\Delta t)$ time. Notice

that this step allows us to synchronize with the device, i.e., we know the exact time on the device, which gives us an exact \hat{t}_{start} , \hat{t}_{stop} for the generation and $\Delta\hat{t} = \hat{t}_{stop} - \hat{t}_{start}$.

We need to guess three time seeds to generate a suggestion for the premaster secret, which have time complexity $\mathcal{O}\left((\Delta\hat{t})^3\right)$. However, since the time seeds are generated sequentially with approximately the same amount of work between each generation, it is possible to implement the attack so that it divides $\Delta\hat{t}$ into three time-slots and searches the first time-slot for the first time seed, and so on... Estimated time complexity for the search for the premaster secret is $\mathcal{O}\left((1/3 \cdot \Delta\hat{t})^3\right) = 1/27 \cdot \mathcal{O}\left((\Delta\hat{t})^3\right)$. Resulting in a total time complexity of:

$$2 \cdot \mathcal{O}(\Delta t) + \frac{1}{27} \cdot \mathcal{O}\left((\Delta\hat{t})^3\right).$$

4.5 Implementation

The attack was tested with a simple SSL client MIDlet written in J2ME and a simple SSL server implemented in J2SE. We used Ethereal [10] to sniff the traffic between the two programs and recover one encrypted SSL package. The attack code guessed keys and decrypted the package and checked the MAC value, utilizing methods from TinySSL [11] for key generation, decryption and MAC calculation.

The MIDlet first ran on a Nokia 6600 and a SonyEricsson P900 over GPRS. However, we were unable to recover the time from the client nonce, which led to the conclusion that these phones do not use the same implementation of the PRNG as Sun's reference implementation.

The same MIDlet was then tested on the emulator in Sun J2ME Wireless Toolkit 2.1 over the loop back interface, where the attack successfully recovered the shared premaster secret.

4.5.1 How long to find the keys?

On average an SSL handshake took approximately 20–30 seconds over GPRS with both the SonyEricsson P900 and the Nokia 6600; the timings include the time it took to enter user input requested by the phones during an SSL connection.

When we tested the attack on the emulator we measured the handshake to take less than 200 milliseconds. To further simulate a proper phone we used $\Delta t = 40s$, and recovered the premaster secret in less than a second on a laptop with an Intel Pentium M processor running at 1600MHz. It is likely that the attack on an SSL connection between a real phone and a server will take more time, since all the seeds to the PRNG were created within 25 milliseconds on the emulator.

5 Conclusion

We have shown that Sun's reference implementation of SSL in MIDP is vulnerable to a key recovery attack because of a bad choice of seed to the PRNG. There

is a solution to this problem: find a better seed. However, this might prove difficult to implement on the software layer of resource constrained devices and the manufacturers of these devices should make hardware randomness available for software developers.

It is also unclear whether or not the developers of mobile phones have solved the problem with cryptographic randomness, history have shown how easy it is to do the generation of random data in an insecure manner.

References

- [1] JSR 139 Expert Group, *Connected Limited Device Configuration, Version 1.1*. Sun microsystems, 2003.
- [2] JSR 118 Expert Group, *Mobile Information Device Profile for Java 2 Micro Edition*. Java Community Process, 2002.
- [3] I. Goldberg and D. Wagner, "Randomness and the Netscape browser," *Dr. Dobbs's Journal*, pp. 66–70, January 1996.
- [4] D. Povey, "Wireless Java security," *Java Developer's Journal*, last visited: May 26, 2006. [Online]. Available: <http://java.sys-con.com/read/37377.htm>
- [5] S. Thomas, *SSL and TLS Essentials: Securing the Web*. Wiley Computer Publishing, 2000.
- [6] D. Eastlake, J. Schiller, and S. Crocker, "Randomness Requirements for Security," RFC 4086 (Best Current Practice), June 2005. [Online]. Available: <http://www.ietf.org/rfc/rfc4086.txt>
- [7] J. Kelsey, B. Schneier, and N. Ferguson, "Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator," in *Selected Areas in Cryptography*, no. Generators, 1999, pp. 13–33. [Online]. Available: citeseer.ist.psu.edu/kelsey99yarrow.html
- [8] D. Davis, R. Ihaka, and P. Fenstermacher, "Cryptographic randomness from air turbulence in disk drives," in *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1994, pp. 114–120.
- [9] P. Gutmann, "Software generation of practical strong random numbers," in *Proceedings of the Seventh USENIX Security Symposium*, 1998, pp. 243–257.
- [10] Ethereal network protocol analyzer, last visited: June 12th, 2006. [Online]. Available: <http://www.ethereal.com/>
- [11] TinySSL, last visited: June 12th, 2006. [Online]. Available: <http://www.xwt.org/javadoc/org/xwt/TinySSL.html>

Paper VI: Secure Networked J2ME applications: Problems and Challenges

Secure Networked J2ME Applications: Problems and Challenges

André N. Klingsheim, Vebjørn Moen, and Kjell J. Hole

Abstract

An increasing number of smartphones support the Java 2, Micro Edition (J2ME) platform. The authors discuss problems and challenges of writing secure client-server applications for these phones. In particular, they explore the security of the current J2ME platform, and examine the new Security and Trust Services API for J2ME.

1 Introduction

A smartphone is a high-end mobile phone featuring a large color display and more processing power than regular mobile phones. A key feature is the ability to install additional applications. The smartphone market is growing fast [1]. The increasing availability of smartphones stimulates the market for rich content such as games, news, media (audio/video), and adult content. Games for mobile phones have been around for several years, but lately we've seen a rapid growth also in this market. Total global revenues in the mobile gaming market were around 2.6 billion USD in 2005, and is estimated to increase to 11.2 billion USD by 2010. It is also anticipated that 20.5% of the global revenues in 2010 are generated by online multi-player games [2].

Many different development platforms exist for smartphones, categorized by phone manufacturers, mobile operating systems and device capabilities. The most widespread platform is Java 2, Micro Edition (J2ME, see java.sun.com/j2me), available on nearly 80% of the smartphones currently on the market.

In this article, the authors discuss experiences gained during a commercial J2ME development project. They discuss how J2ME technologies are implemented on real devices and provide insights into the problems and challenges that occurred during the development process. Current and future J2ME security related functionality is outlined. The reader should be able to make educated decisions on how to develop secure J2ME applications after reading this paper.

The rest of the article is organized as follows. Section 2 gives an overview of the J2ME technologies, Section 3 discusses some of the problems and challenges on the mobile platform, and Section 4 outlines the current J2ME security toolbox. Section 5 discusses how a malicious J2ME client can attack the server, Section 6 gives an overview of the new Security And Trust Services API (SATSA), and Section 7 concludes the paper.

Profile
Configuration
Host Operating System

Figure 1: J2ME Architecture

2 J2ME technologies

The Java platform has several editions, the reader may be familiar with Java 2, Enterprise Edition (J2EE) for the server side and Java 2, Standard Edition (J2SE) for desktop systems. J2ME is a highly optimized Java runtime environment aimed at mobile phones, Personal Digital Assistants (PDAs), and other small devices. J2ME introduces *configurations* and *profiles*, see Figure 1. Configurations are intended for devices with similar characteristics in terms of processors and memory. Profiles, on the other hand, target devices that are similar in terms of screen type, input devices and network connectivity, and complement the low-level functionality of configurations by adding support for e.g. user interface and network connectivity.

A configuration specifies the Java Virtual Machine (JVM) features supported, the included Java programming language features, and supported Java libraries and Application Programming Interfaces (APIs). Two configurations are widely available on J2ME enabled devices, Connected Limited Device Configuration (CLDC) versions 1.0 and 1.1. Version 1.1 is deployed in newly released J2ME devices, while version 1.0 has been around for years. The most notable difference between the two is that CLDC 1.1 adds support for floating point operations. Since CLDC 1.1 is a superset of CLDC 1.0, Java applications built for CLDC 1.0 will run without problems on CLDC 1.1. Unless any of the additions in version 1.1 is strictly needed it is therefore recommended that developers build their applications for CLDC 1.0 to be compatible with as many devices as possible.

Two profiles exist that extend the functionality of CLDC, namely Mobile Information Device Profile (MIDP) versions 1.0 and 2.0 [3]. The MIDP profiles add APIs for user interface, network connectivity and persistent storage. Network connectivity is provided through HTTP connections, and persistent storage is provided through a record management system. MIDP 2.0 is a superset of MIDP 1.0, and includes support for secure HTTP (HTTPS) connections and more powerful graphics APIs for gaming.

New J2EE/J2SE/J2ME components are developed through the Java Community Process (JCP) program. Java technology specifications are created by first submitting a Java Specification Request (JSR) that must be accepted by an executive committee. Once a JSR is accepted, an expert group is formed to take responsibility for the specification development. The specification draft must first pass a community review, and then a public review. Finally, a proposed final draft is presented, and a final approval ballot decides if the specification is suitable for a final release.

An expert group typically consists of many parties. As an example, the MIDP 2.0 expert group included, amongst others, Ericsson Inc., Motorola, Nokia, Siemens, Sun Microsystems Inc., Samsung, and Symbian Ltd. The formation of an expert group enables large industrial parties to collaborate in order

to specify new functionality for the Java platform. Hence, competition between different vendor specifications is minimized, increasing the likelihood of wide adoption of a specification. All CLDC versions and MIDP versions are results of JSRs, as well as several other J2ME components such as SATSA (JSR-177), the Wireless Messaging API (JSR-120), and the Java APIs for Bluetooth Wireless Technology (JSR-82). All specifications are freely available at the JCP website (www.jcp.org).

3 Current problems and challenges

J2ME developers face several problems and challenges. Our experience with J2ME enabled smartphones is that the implementations generally have some quality issues. Developers are likely to spend time solving problems that occur because of the varying quality of the J2ME implementations. Specific phone models can have their own bugs, forcing developers to maintain several parallel versions of their source code to support as many devices as possible. This situation is not consistent with the Java philosophy of “Write Once, Run Anywhere,” and severely increases the complexity of mobile software development and maintenance. In our experience, MIDP 2.0 implementations have less bugs than MIDP 1.0 implementations. However, to cover as much of the market as possible developers have to consider all the existing MIDP 1.0 devices already out there. MIDP 2.0 devices still constitute a minority of all J2ME devices sold in recent years.

3.1 Insufficient testing

Many developers fail to recognize that J2ME devices can behave inconsistently. It is of utmost importance that J2ME applications are tested on many different devices. We have seen some businesses base their testing on 4–5 devices, only to be really surprised when their application does not run correctly on other devices. In our opinion, too many businesses neglect the testing phase, and let their customers do the beta testing. To survive with such a strategy, you need a pretty unique application and your customers have to be both enthusiastic and understanding.

In order to get a real understanding of how J2ME phones operate, several devices from each major vendor must be tested and each version of a vendor’s development platform must be represented in the set of test devices. Of course, testing 40–50 (or even more) devices will cost you. Businesses therefore need to carefully consider how much resources to spend on testing, versus the risk involved in releasing an application to customers with devices that have not been tested.

3.2 Permanent bugs

On the desktop, we’re used to download patches from Microsoft Windows Update or similar systems, to solve security issues and fix bugs in our software. How is bug fixing handled on mobile phones? The short answer is that it’s not. Mobile phone vendors release new software versions for their mobile phones, but these do not reach consumers that have already bought phones. The majority

of mobile phones must be handed in to a repair shop to perform a software upgrade. Only hardcore mobile phone geeks actually do this, hence the bugs you get when buying a mobile phone usually stay there for the phone's lifetime. However, if you buy the same phone a year later, you will probably get a newer software version. As mobile phone viruses have started to appear, we really see the need for a solution to the patching problem that enables consumers to upgrade the phone software themselves, similarly to what they hopefully do with their desktop computers.

3.3 Resource management

The mobile platform differs from regular desktop computers, by having limited amounts of memory, processing power, network bandwidth, and disk space available to application developers. Though smartphones have more resources than regular mobile phones, they should still be considered a resource constrained platform compared to the desktop. In addition to the traditional functionality focus, mobile developers must consider effective resource utilization to make user-friendly mobile applications.

Defensive programming is the key to create a well functioning application. For example, available runtime memory and the amount of storage memory on the device can be queried during program execution. The developer should always make sure that there is enough memory to carry out the operations of the program. If the device runs out of memory it will show an error message and terminate the application, giving the user the impression that an error occurred, while the real problem was that the application did not adapt to the amount of available resources.

3.4 Responsive applications

Applications must be responsive to provide a positive user experience. To achieve this goal, intimate knowledge of the inner workings of J2ME devices is very helpful, since different devices behave in different ways. One example is if the developer actively triggers a garbage collection to reclaim memory from unused objects. The garbage collector should then run in the background, with minor impact on the application. However, some devices will "freeze" for a few seconds while memory is collected, which is probably not what the developer wanted or expected.

Multi-threading is important when developing applications for mobile phones. A program's execution flow is event-driven, so the main thread must be idle and ready to handle events. Time consuming operations such as lengthy calculations or network communication should therefore take place in a separate thread to retain a responsive user interface. This should be familiar for developers with experience from developing graphical User Interface (UI) applications on the desktop, as the same considerations of UI- versus non-UI threads apply.

4 MIDP 2.0 security framework

MIDP 2.0 includes several mechanisms to secure an application and provide secure communication channels. This section gives an overview of these mech-

anisms, and explain why some of them have weaknesses.

Applications can be signed in order to obtain authenticity and integrity. Secure communication channels are realized by HTTPS connections, which in most cases rely on SSL (Secure Sockets Layer) or TLS (Transport Layer Security). In addition, MIDP 2.0 ensures that an application is not able to read other J2ME applications' persistent data unless it is explicitly allowed. However, since encrypted storage is not provided, hardware attacks exist to read application data. Even easier, on some devices you can install a file system explorer, locate the files used by J2ME, and use Bluetooth to send them to another device. It's clear that you cannot trust the storage system in J2ME with sensitive data.

Developers will probably have high hopes after reading the MIDP 2.0 specification, but sometimes things look better than they are. As we shall see, by studying the details of MIDP 2.0 one realizes that some critical security functionality is actually optional to implement on J2ME enabled devices, or can be based on insecure mechanisms. Also, once testing has commenced on real devices, developers will be disappointed to realize that mandatory security functionality is not implemented correctly on some mobile phones.

4.1 Application signing

Application signing based on an X.509 Public-Key Infrastructure (PKI) [4] is an optional part of the MIDP 2.0 specification and enables the device to verify the origin and integrity of J2ME application. Application signing is a good idea, but what should developers do when several mobile phones lack support for signed applications? Such a situation is highly unsatisfactory for m-commerce, or mobile governmental services since parts of your security architecture crumble if your application is installed on certain devices.

We tested two newly released Samsung smartphones during our project and both of them refused to install signed J2ME applications. The J2ME applications were signed with a code-signing certificate obtained from Verisign Inc. The Nokia devices we tested validated the certificate and applications without problems. This illustrates that to support all devices, both signed and unsigned versions of applications must be published to customers, effectively making the "signed application" feature of the security architecture optional. Since the validation of signed applications depends on a pre-installed list of root certificates (belonging to Certificate Authorities) other issues surface, which we'll discuss later.

4.2 Secure communication

Secure connections in MIDP 2.0 must be implemented by one or more of the following specifications:

- HTTP over TLS (RFC 2818) with TLS Protocol version 1.0 (RFC 2246)
- SSL version 3.0 [5]
- WTLS (Wireless Transport Layer Security) [6]
- WAP (Wireless Application Protocol) TLS Profile and Tunneling Specification [7]

Note that a developer cannot know which specifications are implemented on a specific device, without actually testing it. In the case where WTLS is used, end-to-end encryption is not provided. Secure connections will then exist between the phone and the WAP gateway, and further from the gateway to the final destination. Hence, the gateway has access to unencrypted data and must be fully trusted. This is not acceptable for a high security system, as the gateway is usually operated by the mobile network provider.

Observe also that secure connections in MIDP 2.0 require the server to have a valid certificate for authentication. However, there is no support for certificate based authentication of the client, hence the client must be authenticated on the application level by other means.

4.3 Certificate management and verification

All certificate verification procedures on a mobile phone rely on a set of pre-installed root certificates on the phone, equivalent to what we see in web browsers. Of course, different mobile phones may have different root certificates installed.

Self-signed certificates can be installed on smartphones, which is very useful during a test phase. We successfully installed a self-signed X.509v3 certificate on the Nokia 6600 by publishing it to a web server and then downloading it to the phone via WAP. A certain level of user interaction was needed after the certificate was installed, as all certificates in the Nokia 6600 have properties describing their area of use. A certificate installed by the user must therefore be enabled for verification of signed applications or server authentication before it can be put to use.

An important requirement when working with certificates is the ability to validate a certificate. Time limited validity is one mechanism, but support for certificate revocation is much more critical in order to establish a certificate's validity. The MIDP 2.0 specification states that "Certificate revocation can be performed if the appropriate mechanism is implemented on the device. Such mechanisms are not part of MIDP implementation and hence do not form a part of MIDP 2.0 security framework." Consequently, the validity of a certificate can only be established based on the assumption that none of the certificates in its certificate chain have been revoked.

One interesting observation we did was that the Samsung SGH-E720 had Verisign class 1, 2, 3 and 4 root certificates installed that were all reported valid from 1. October 1999 to 1. January 1970. Other installed certificates showed sensible validity intervals. We located the very same Verisign root certificates on the Nokia 6600, and they all had expiry date 01.10.2049. We give Samsung the benefit of doubt, and assume that the invalid expiry date is not the value actually stored in the certificate, but more of a presentation problem. However, since the date is not presented correctly, the expiry date might not be interpreted correctly during the certificate validation process. Unfortunately, we were not able to verify this, since it is not a trivial task to find a website that uses a certificate signed by a specific root certificate.

4.4 Secure storage

Encrypted storage is not supported in MIDP 2.0. An adversary may therefore use equipment to read the memory in your mobile phone and get access to

your data. Cryptographic libraries for Java exist, and may be a solution to the plaintext storage problem. One example is Bouncy Castle's lightweight cryptography API supporting several symmetric ciphers. However, the availability of cryptographic APIs does not automatically solve the plaintext storage problem. Encryption is a computational intensive task, and encryption implemented in Java can prove to be time consuming since low-level optimizations (e.g. for CPU architecture) is impossible. Native libraries or cryptographic hardware are likely to perform encryption more efficiently.

Another important issue is the lack of sources of randomness in J2ME implementations. A strong cipher can be used, but it is essential that the encryption key is impossible to guess. MIDP 2.0 does not provide a cryptographically strong Pseudo Random Number Generator (PRNG) similar to the `SecureRandom` in J2SE. Hence, the PRNG in J2ME is not suitable for generating encryption keys. Still, developers use the PRNG for different purposes which can have a major impact on the security of a system, especially since developers tend to seed the PRNG with the current time [8, Ch. 10]. One example is an attack on SSL in Sun's MIDP reference implementation [9].

4.4.1 Homemade crypto

Since MIDP 2.0 does not provide encrypted storage, and MIDP 1.0 does not provide HTTPS links, some J2ME development companies decide to specify their own "lightweight" crypto schemes. Well-meaning efforts to create new, cryptographic algorithms usually result in solutions that keep data hidden from average users, but the solutions are very seldom cryptographically strong. Such initiatives usually rely on the secrecy of the encryption algorithm, which is considered very bad practice [8, p. 268].

In many countries there are laws regulating how private data must be protected during transportation over a medium not controlled by the two communicating parties. These laws often require that the data is encrypted with an algorithm of strength equal to or better than 3DES. The Advanced Encryption Standard (AES) fulfill this requirement, but usually homemade crypto solutions will not have the strength of well tested encryption algorithms such as 3DES or AES.

5 Using clients to attack the server

We'll not consider well-known Internet server attacks from viruses and worms, or DDoS attacks [10]. Instead, we'll briefly discuss how a client application can attack the server application.

Client applications should be assumed to be evil by nature, even though you wrote them yourself. Several approaches can be used to attack a server application. Two examples are clients sending commands out of order or sending unexpectedly large data chunks as input to an application. We've talked to several businesses that develop client-server applications, and they all seem to completely trust their client software. Their arguments are along the line of: "Hey, we wrote it. Why shouldn't we trust it?" In our opinion, this is a dangerous approach as trust is easily misplaced unless a careful analysis of the trust model is carried out [8, Ch. 12].

The client software may be handed out to all kinds of people, including the ones that cannot resist trying to break it. Software can be reverse engineered, and the source code can be studied. Reverse engineering might not even be necessary, the binary code could just as well be tampered with. The gaming industry is experiencing this on a daily basis, as games are cracked to avoid license key checks. Another approach is to use a network sniffer, figure out the application protocol, and write your own malicious client that behaves similarly to the original client. Several measures can be taken to increase the level of security in an application, but it all starts with the attitude of the developers. For developers interested in building secure software we recommend [8].

6 J2ME security in the future

Several of the shortcomings in the MIDP 2.0 security architecture are addressed by SATSA, the new Security and Trust Services API for J2ME [11]. SATSA relies on a Security Element (SE), implemented in either software or hardware. This implies that the SE can take different forms such as a software component, dedicated hardware in the device, or a removable smart card. Several SEs can be available in one device. The exact form of the SE is transparent to the application developer, the interaction with the SE is handled by the SATSA implementation.

The support for cryptographic smart cards is of particular interest to developers writing J2ME applications for smartphones. Keys and certificates can be stored on the smart card, and data can be signed without the private key ever leaving the card. High-end smart cards are tamper resistant and provide authentication schemes, such as requiring a PIN or a password before access to the smart card is granted. This way, security is dependent on the smart card not being compromised. Private keys do not have to be stored on diverse insecure clients, enabling vendors to focus on keeping the smart card secure from physical tampering and, just as important, smart card API exploitation [12].

An interesting observation is that many banks are already giving their customers smart cards, which also have a magnetic strip in order to be compatible with old ATMs. By giving customers smart cards with cryptographic tools, a bank could have client software for mobile phones, PDAs, and desktop computers, all relying on the customer's smart card, hence giving (nearly) the same level of security for key storage on all platforms. Of course, different OSs have different levels of security, so a careful analysis of each platform must be carried out to make sure that the smart card is accessed in a controlled manner.

6.1 SATSA APIs

The SATSA specification defines four APIs, SATSA-APDU, SATSA-JCRMI, SATSA-PKI, and SATSA-CRYPTO. The first two APIs add functionality for smart card interaction. SATSA-APDU enables communication with smart cards using the Application Protocol Data Unit (APDU) protocol defined by the ISO7816-4 specification. SATSA-JCRMI enables high level communication with smart cards through the Java Card Remote Method Invocation Protocol (JCRMI).

SATSA-PKI enables applications to request digital signatures from an SE, hence providing authentication and possibly non-repudiation [4, pp. 32–33] by

using keys stored on a smart card. Client certificate management is also provided by SATSA-PKI, giving an application the opportunity to add or remove certificates from an SE. The most interesting part of the certificate management is the possibility to request generation of a new key-pair and then produce a Certificate Signing Request. The fact that the client generates its own keys is one of the key factors needed to support non-repudiation in a system. Note that key generation is dependent on the SE, the SE might not support key generation at all. Hence, the SE must be chosen with care, considering the application requirements.

SATSA-CRYPTO offers cryptographic tools like message digests, digital signature verification, and ciphers. The API enables applications to store data encrypted and signed on a mobile device, ensuring both confidentiality and integrity. Applications that require secure storage of highly sensitive information can therefore be realized. Note that it is up to the implementor to decide which ciphers and digest algorithms to include. The SATSA specification recommends DES, 3DES, and AES as symmetric ciphers, RSA as asymmetric cipher, and SHA-1 as the digest algorithm. SHA1withRSA is the recommended algorithm for digital signatures.

6.1.1 Security issues

SATSA is distributed as a part of the Java Runtime Environment (JRE) in some smartphones. The OS of the mobile phone must therefore be fully trusted, as the JRE depends on services from the OS. The SATSA specification states that both SATSA and the application using SATSA must trust the OS. The specification further states that when SATSA is taking over UI control, e.g. when asking the user for the smart card pin code, the UI must be “distinguishable from a UI generated by external sources” in order to prevent a malicious application from mimicking the SATSA UI. Another requirement is that “external sources are not able to retrieve or insert PIN data”. These requirements put a lot of responsibility on the OS. We see three scenarios that could cause trouble.

Since the PIN is entered on the device through the keypad, a keylogger application could possibly obtain the PIN. Hence, the OS must limit the distribution of key pressed events to the SATSA implementation exclusively. Keyloggers exist for Symbian OS versions prior to version 9, so this may be an issue. Another approach would be to read the PIN from memory. When a PIN is entered, it must be stored in memory somewhere before it is handed over to the smart card. It could be possible for other applications to read this memory.

J2ME applications use a per-application dedicated logical channel to communicate with the smart card. This channel could be hijacked using low-level functionality of the OS. If access to the smart card is acquired on a level below the SATSA implementation, requests could be sent to the smart card circumventing the SATSA implementation. If this functionality was included in e.g. a Trojan horse, an attacker could have full access to your smart card without your knowledge.

We do not know if these scenarios actually apply, but they illustrate some problems with the complete trust in the OS. In our opinion, an application cannot be more secure than the OS it runs in. It remains to be seen how the mobile platform copes with viral threats, as we’ve only seen the beginning of viruses for mobile phones.

The Trusted Computing Group has an initiative to make OSs on mobile devices more secure (www.trustedcomputinggroup.org/groups/mobile). In the future, we expect that mobile devices will become more trustworthy, making it easier for developers to create secure applications.

6.1.2 SATSA shortcomings

Client certificates generated by the SATSA implementation cannot be used for authentication during setup of HTTPS links using SSL or TLS. Developers must therefore handle certificate-based authentication of clients themselves. One alternative is to open an SSL connection to a server, which authenticates the server and provides a secure communication channel. Client authentication can then be carried out using a certificate provided by the client application, and having the client sign a challenge from the server. This scheme is more robust than the widely used password authentication, as dictionary or brute-force attacks are avoided. Consequently, SATSA can improve the level of security in current server-client applications by replacing old-fashioned authentication schemes. However, it would be convenient if client certificates stored by the SATSA implementation could be used by SSL or TLS implementations found in most newer smartphones.

Signature verification with SATSA is not as easy as signature creation. SATSA generates signed messages on the Cryptographic Message Syntax (CMS) format, but does not easily validate these messages. To verify a signature, the application needs to split a message into respective data and signature parts, and must supply a public key to be used for verification. Libraries exist for handling CMS messages, so developers need not implement parsing of CMS messages themselves. However, it would be easier if SATSA could verify its own signed messages.

Signature generation and signature verification is not handled in the same way in terms of how information is presented to the user. When data is signed by the user, the underlying SATSA implementation takes control of the UI and presents the user with the certificate to be used for signing, along with the data to sign. The user can then be confident that he signs the intended data, and not something else. Signature verification is just as important, but here the application must present details about the signature to the user. This means that you trust SATSA when signing data, while you trust the application to show you correct information when verifying signatures. We would like to trust SATSA on both occasions.

Note also that SATSA does not provide verification of certificates. The developer must implement the certificate verification process and public key extraction from the certificate, along with presenting the certificate and signed data to the user. SATSA provides the most basic building blocks for PKI enabled applications, but does not include any certificate validation functionality present in J2SE.

Private keys stored on the smart card cannot be used for decryption, as they are accessible only for signing. SATSA supports asymmetric crypto, and it would be convenient if a certificate (and its corresponding private key) could be selected for decryption. Data could then be decrypted on the smart card, without exposing the private key to the application nor the operating system.

7 Conclusions

The security model in J2ME has its limitations and may not be adequate for all purposes. In addition, hurdles exist because of the bugs and limitations in various J2ME implementations on real devices. A proper analysis of the security requirements for an application must be carried out considering these limitations and difficulties.

SATSA empowers the mobile platform with cryptographic capabilities and is therefore a much needed library for secure application development. Local encryption is possible, and authentication schemes can rely on public key cryptography instead of the usual username and password authentication. Storage of user certificates and support for digital signatures in smart cards open up possibilities for applications that require a high level of security. SATSA fits perfectly in a scenario where strong authentication of the client is needed, as well as digital signatures on behalf of the client. However, SATSA provides only the basic cryptographic building blocks to build a PKI. Important functionality found in J2SE such as certificate parsing, validation, and storage is left to the developer to implement. Hence, implementing signature validation and public key cryptography based on public keys in certificates can prove to be complex tasks.

Acknowledgment

We'd like to thank World Medical Center for their cooperation during the project and for giving us access to the mobile phones needed to carry out the security related tests.

References

- [1] Canalys, "Worldwide smart phone market soars in q3," last visited: March 26, 2006. [Online]. Available: <http://www.canalys.com/pr/2005/r2005102.htm>
- [2] telecoms.com, "Mobile games industry worth usd 11.2 billion by 2010," last visited: March 26, 2006. [Online]. Available: <http://www.telecoms.com/itmgcontent/tcoms/search/articles/20017303052.html>
- [3] JSR 118 Expert Group, *Mobile Information Device Profile for Java 2 Micro Edition, Version 2.0*, 2002.
- [4] C. Adams and S. Lloyd, *Understanding PKI*, 2nd ed. Addison-Wesley, 2003.
- [5] Netscape Communications, *The SSL Protocol, Version 3.0*, 1996.
- [6] Open Mobile Alliance, *Wireless Transport Layer Security Specification*. WAP 1.2.1 conformance release, 2000.
- [7] Open Mobile Alliance, *WAP TLS Profile and Tunneling Specification*. WAP 2.0 conformance release, 2001.

- [8] J. Viega and G. McGraw, *Building Secure Software*. Addison-Wesley, 2002.
- [9] K. I. F. Simonsen, V. Moen, and K. J. Hole, “Attack on sun’s MIDP reference implementation of SSL,” in *10th Nordic Workshop on Secure IT-systems (Nordsec 2005)*, 2005.
- [10] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service Attack and Defense Mechanisms*. Prentice Hall, 2005.
- [11] Sun Microsystems, “Security And Trust Services API for J2ME,” last visited: March 26, 2006. [Online]. Available: <http://java.sun.com/products/satsa/>
- [12] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov, “Cryptographic processors—a survey,” University of Cambridge, Tech. Rep. 641, 2005. [Online]. Available: <http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-641.pdf>

Paper VII: Vulnerabilities in E-Governments

Vulnerabilities in E-Governments

Vebjørn Moen, André N. Klingsheim,
Kent Inge Fagerland Simonsen, and Kjell Jørgen Hole

Abstract

This paper shows that more than 80% of the e-governments in the world are vulnerable to common Web-application attacks such as Cross Site Scripting and SQL injection. Industrialized countries were found to be more vulnerable than under-developed countries (90% vs. 50%). The paper also describes some malicious data mining possibilities on the Norwegian e-government, and how information can be combined and used to create other practical attacks.

1 Introduction

E-government is one of the buzzwords of the Internet age, referring to any government function or process that is conducted in a digital form over the Internet. Basically, one or several Web portals supply individuals and businesses with public information, government forms for download, and contact with government representatives. According to most e-government plans, providing services over the Internet will yield higher efficiency and quality, easier access, the possibility of offering individual services, and increased transparency, ultimately leading to a more efficient public sector [1].

An EU press release claims: “A new survey of e-government services prepared for the European Commission has found that EU citizens are saving 7 million hours a year on the time it takes to do their income tax returns, and EU firms are saving about €10 per transaction on their VAT returns by doing them online. Moreover, there is still huge scope for further savings” [2].

However, since e-government projects are provided over an insecure channel, namely the Internet, other important issues surface. In most countries there are no governmental infrastructure that supports authentication, confidentiality, and integrity. The Public Key Infrastructure (PKI) used in E-Commerce is not applicable to e-government without a thorough analysis of what the new trust model should be. The trust calculation for commerce is based on monetary issues, while government solutions involve important infrastructure, society, and privacy issues.

There are also other problems, related to Web solutions, that can give unforeseen consequences when e-government solutions are put forth. We have examined e-governments in the whole world for known Web application issues, and found that a vast majority of them have common vulnerabilities. As a special case study, we have also examined parts of the Norwegian effort to create an e-government and located possibilities for malicious data mining and denial-of-service attacks in the services offered.

Section 2 gives a short introduction to some common Web application vulnerabilities, Section 3 presents the results from the inquiry into the security of e-governments, Section 4 describes why the Social Security Number (SSN) should not be used for identification or authentication, and Section 5 concludes the paper.

2 Web application vulnerabilities

In this section we give a brief overview of the Web application vulnerabilities we have chosen, Cross Site Scripting (XSS) and Structured Query Language (SQL) Injection. These vulnerabilities were chosen because they have been well-known for years. XSS and SQL injection attacks are simple, yet powerful attacks against Web applications that can be carried out with nothing more than a Web browser. The implications of these attacks will be discussed in the following sections.

2.1 Cross Site Scripting

XSS [3, 4] is perhaps the most common Web application vulnerability, where the Web application reflects unvalidated user input on a dynamically generated Web page. This makes it possible to supply Javascripts and HTML code in the user input, which will then be part of the dynamically generated page. E.g., by using Javascript it is possible to steal cookies, and Cascading Style Sheets and the HTML tag *iframe* can be utilized to completely control both the content and the layout of the resulting page. Common attack vectors are search applications which reflect the search string, and parameters supplied in the URL.

Here is an example of an XSS attack string which generates a page with arbitrary content on an XSS vulnerable site:

```
http://VulnerableSite.com/search?q=<iframe style=height:100%;  
width:100%;border:none;transparent:none;position:absolute;top:0;  
left:0;z-index:100; src=http://AttackerSite.com/ />
```

The attack string can be URL encoded [5] so that the content is unreadable for the average Internet user. An attack is successful if a victim visits an URL containing the XSS attack. This can be achieved by e.g. spoofing an e-mail from a sender in which the user has trust.

2.2 SQL injection

SQL injection [3, 6] is considered to have more severe consequences than XSS, due to the fact that a successful SQL injection can compromise the integrity of a database. A Web application is vulnerable to SQL injection if unvalidated user input is used to generate SQL queries. The following example is a typical SQL query used to generate dynamic Web pages:

```
SELECT * FROM articles WHERE id='<user input>';
```

An attacker can control the user input, and, e.g., enter: `'; DROP 'articles';`. This adds a second command to the SQL query, which then becomes: `SELECT * FROM articles WHERE id="'; DROP 'articles';'`

These SQL commands will select some data, delete the table "articles" in the database, and then generate an SQL error due to the single quotation mark.

Continent	Only XSS	Only SQL	XSS and SQL	XSS or SQL	None
Africa (61)	14.75 (9)	0.00 (0)	34.43 (21)	49.18 (30)	50.82 (31)
Asia (55)	9.09 (5)	0.00 (0)	76.36 (42)	85.45 (47)	14.55 (8)
Europe (53)	7.55 (4)	0.00 (0)	83.02 (44)	90.57 (48)	9.43 (5)
North America (34)	20.59 (7)	2.94 (1)	52.94 (18)	76.47 (26)	23.53 (8)
Oceania (25)	24.00 (6)	0.00 (0)	28.00 (7)	52.00 (13)	48.00 (12)
South America (17)	17.65 (3)	0.00 (0)	52.94 (9)	70.59 (12)	29.41 (5)

Table 1: Percentages of vulnerabilities in e-governments for each continent. Number of countries enclosed in parenthesis. Note that some countries are counted for more than one continent, e.g. Russia belongs to both Europe and Asia.

In general, SQL injection gives an attacker the opportunity to manipulate the database and in special cases execute arbitrary code on the database server. It is therefore an effective attack on Web applications. Typical attack vectors are logins, search forms, and the URL of dynamically generated pages (e.g. `http://VulnerableSite.com/article?id=42` could result in a SQL query similar to the one in the example). SQL injection can be avoided through user input validation, ensuring appropriate handling of characters with a special meaning in SQL.

2.3 Combining information and data mining

Internet was created for information sharing, and e-governments will hopefully give more efficient services to the public. However, the Web is completely different from the more traditional information channels, such as radio, TV, and postal services. The fact that we have an up-channel gives rise to new attack scenarios which must be taken into account before the e-government concept can replace old government systems. In addition to the previously described attacks on e-government portals, other risks exist. Malicious data mining can be possible when huge databases containing governmental information about the public are made accessible through e-government Web applications.

3 Vulnerabilities in e-government Web portals

In order to investigate the security levels of e-government Web applications we have probed government Web sites for XSS and SQL injection vulnerabilities. An e-government Web application is considered to be all Web pages under a gov.<country code> sub domain, and/or pages from the government, ministries, and parliament.

In addition to searches on Google, the Google directory of countries [7] and Gunnar Anzinger's list of Governments on the WWW [8] were used to locate the e-government Web applications. The investigation was carried out during February and March, 2005.

3.1 Scope of vulnerabilities

In this paper, a Web application is considered vulnerable to XSS if we are able to insert script or HTML code on a dynamic Web page. Furthermore, a Web application is vulnerable to SQL injection if we are able to change the structure of an SQL query run by a dynamic Web page, causing an SQL statement error. Vulnerabilities are not examined further, since both ethical and legal issues then arise. In order to evaluate the severity of a vulnerability we would have to exploit it in the worst possible manner. Disruption of service, loss of data, or bringing the Web application to a complete halt could be the effect of such an investigation. Our mission was not to cause havoc on e-government Web sites, but rather to check the status of their Web application security. Hence, we were forced to use a simple definition of what a vulnerable Web application is.

We suggest defining a country as vulnerable if there exists one vulnerability in an e-government Web application. This simplification is done since there are millions of governmental Web pages on the Internet, so an exhaustive review of all available pages would be impossible without automated tools. Different countries of course have different numbers of vulnerabilities, and the vulnerabilities vary in their severity. In addition, many of the Web pages are subject to constant change, so vulnerabilities may come and go. Hence, an exhaustive review would never give exact results. We therefore argue that our approach is useful, confirming that there are problems in governmental Web applications and that harm may be done when armed with nothing more than a Web browser.

3.2 Methods for finding XSS and SQL vulnerabilities

There exist algorithms and programs that can check Web applications for common vulnerabilities, one example is [9], but we chose to check all the e-governments manually in order to completely control which pages and input vectors that were used.

The basic idea in finding XSS and SQL vulnerabilities is to look for input vectors that are used by some server side script. To test for XSS, an input such as `<script>alert('XSS')</script>` can be used. If this string is included in a dynamically generated page, a Javascript enabled browser will show a Javascript pop-up window containing the text XSS. To test for SQL injection vulnerabilities, a single quote (') can be used as input. The single quote is a special character in SQL, and if it is included in the SQL query it will most likely generate an SQL statement error.

To further ease the work of finding vulnerabilities, we used Google to search for file extensions we know are more likely to be vulnerable than others. Consider e.g. the Australian government, which have the gov.au domain. If we search for: `site:gov.au inurl:php` in Google, we will find pages under the gov.au domain generated using PHP. A common practice was to search for ASP or PHP pages, and then other dynamic pages if no vulnerable ASP or PHP pages were found.

3.3 Results

There are 244 entries on the ISO 3166 list [10], but some of these entries are for countries/territories governed by another country. In total, we were able to locate e-government sites for 212 countries, and 173 of these countries were

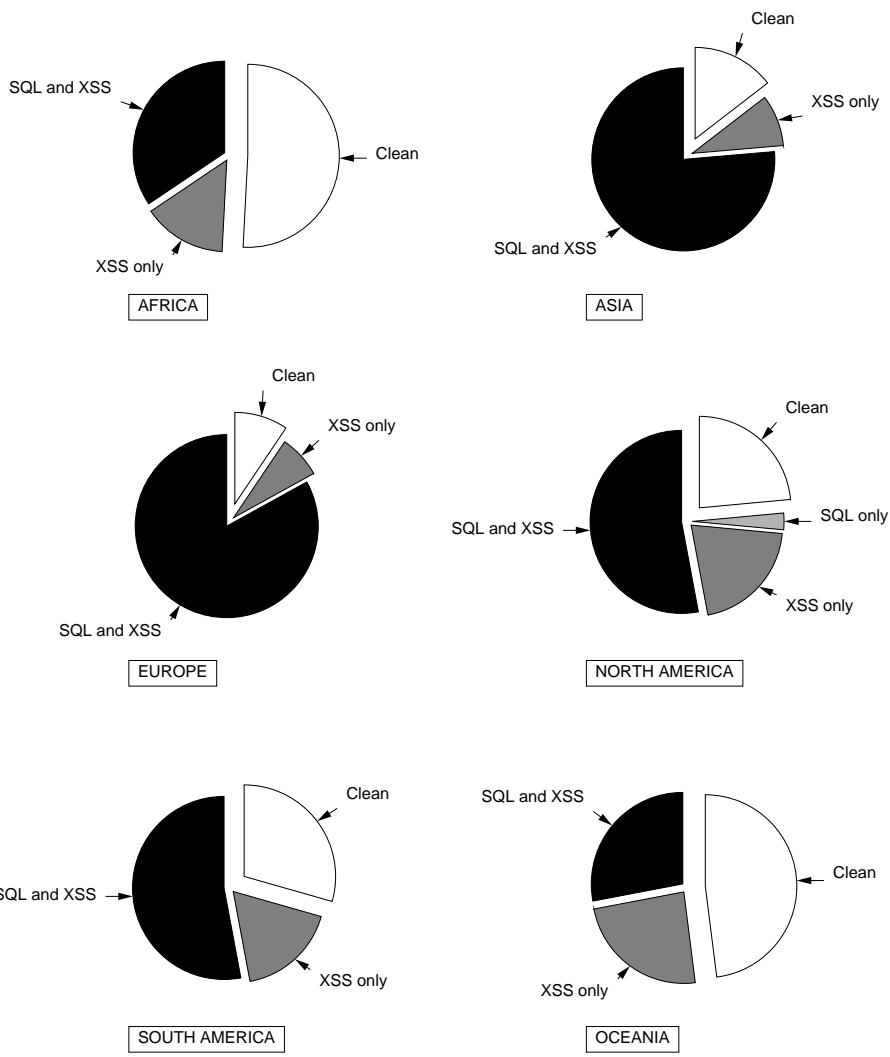


Figure 1: Pie charts showing vulnerabilities for each continent.

Country category	Only XSS	Only SQL	XSS and SQL	XSS or SQL	None
1st World (32)	6.25 (2)	0.00 (0)	90.63 (29)	96.88 (31)	3.12 (1)
2nd World (31)	9.68 (3)	0.00 (0)	80.64 (25)	90.32 (28)	9.68 (3)
3rd World (50)	18.00 (9)	0.00 (0)	32.00 (16)	50.00 (25)	50.00 (25)
G8 (8)	0.00 (0)	0.00 (0)	100.00 (8)	100.00 (8)	0.00 (0)

Table 2: Percentages of vulnerabilities in e-governments for different country categories. Number of countries enclosed in parenthesis. Note that not all of the 244 countries are included in the statistic for 1st, 2nd, and 3rd World, we used the countries listed on [11].

vulnerable to either XSS or SQL injection. That is; 81.6% of the countries with a Web portal were vulnerable to these simple attacks.

Table 1 gives the percentages of vulnerable countries for each continent. Europe have the highest ratio of countries vulnerable to either XSS or SQL injection, closely followed by Asia. More than 90% of European e-governments have a vulnerability, while slightly less than 50% of African e-governments are vulnerable. Figure 1 illustrates the numbers with pie charts.

The same tendency can be seen in Table 2, showing the percentages of vulnerable 1st, 2nd, and 3rd World countries, as well as the G8 countries. More than 90% of 1st and 2nd World countries are vulnerable, along with all of the G8 countries. However, we found SQL injection or XSS for “only” 50% of the 3rd World countries. These results are also shown as pie charts in Figure 2.

While testing the e-government portals we noticed that almost all of the industrialized countries had dynamic pages, based on technologies such as PHP, ASP, or JSP. We also noted that many of the dynamic pages retrieved parts of their content from databases. To our “disappointment” we only located static HTML pages for many African and 3rd World countries. Figure 3 shows the distribution of Web server software based on vulnerability type. Figure 4 shows which technologies were used to generate the vulnerable pages. However, the results are affected by the methods we used to find vulnerable pages, and they give a better picture of which technologies were most frequently used than of the security of any of the technologies.

3.4 “Invulnerable” sites

Some sites did not seem to have any vulnerabilities so we had to look at the characteristics of these sites. According to [12] there is a trinity of trouble which makes software difficult to control: complexity, extensibility, and connectivity. Therefore; our assumption is that sites with no discovered vulnerabilities must be smaller and less complex, and also use fewer technologies.

The 39 “invulnerable” countries had on average 8,126 pages indexed by Google, and 2.75 different technologies had been used to create the pages. In comparison; the G8 countries had an average of more than 12 million pages, and had pages created by 14–15 different technologies.

Our conclusion from these observations is that the “invulnerable” sites are not vulnerable because their complexity is low. The big industrial countries on the other hand use many different technologies to build enormous Web sites, hence increasing the possibility of introducing vulnerabilities in their Web applications.

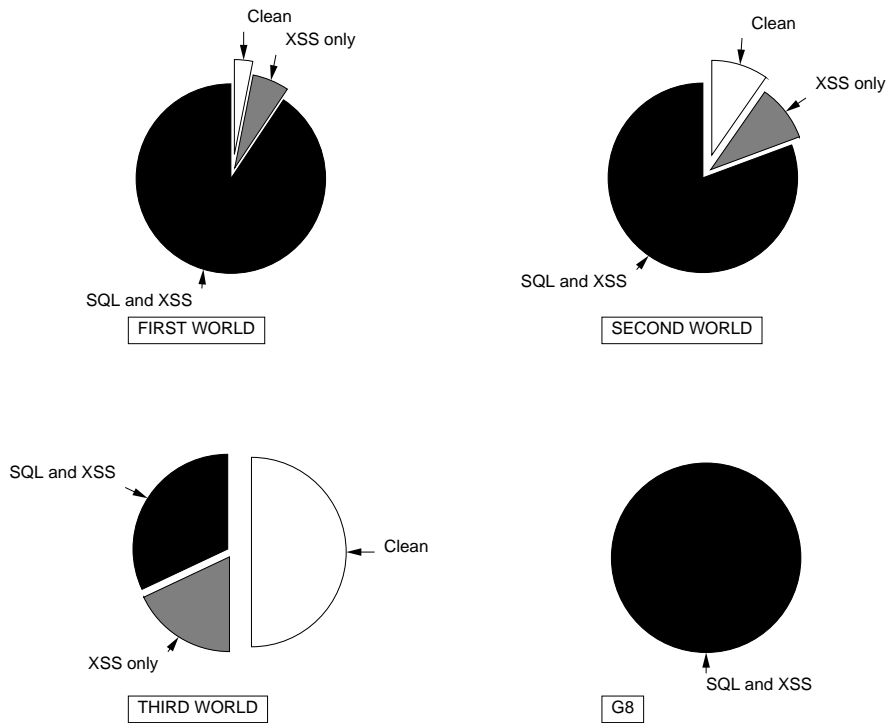


Figure 2: Pie charts showing vulnerabilities for the 1st, 2nd, and 3rd World countries, and also for the G8 countries.

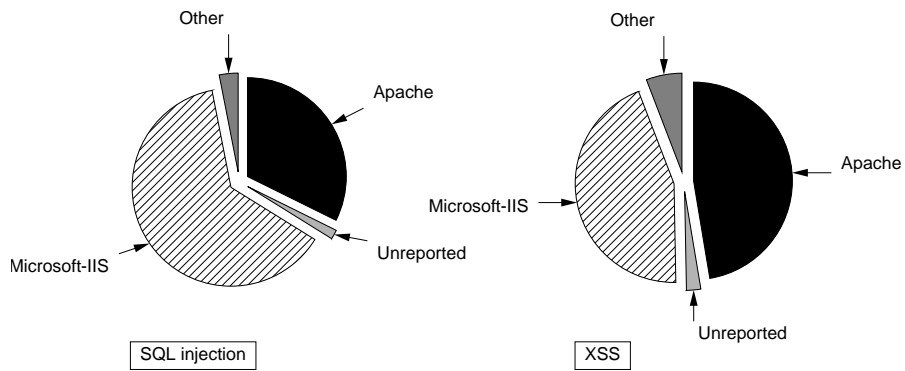


Figure 3: Pie charts for SQL injection and XSS, with regard to which Web server was serving the vulnerable portal. We used the 'Server' HTTP-header field in the HTTP reply to identify the servers, and consider the whole Apache family as Apache, and we do not distinguish between different versions of Microsoft-IIS. The 'Other' category includes Netscape-Enterprise, Oracle, Lotus-Domino, and Sun-ONE-Web-Server.

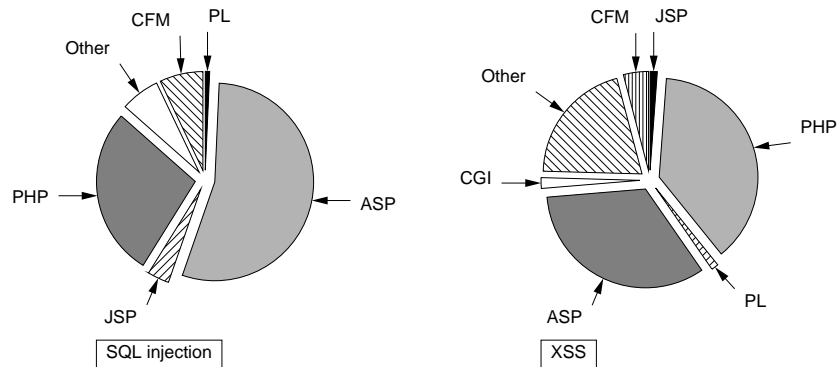


Figure 4: Pie charts showing which technologies generated the vulnerable pages, based on file-endings. We see that ASP and PHP dominates the statistics. The 'Other' category contains the pages we were unable to classify.

3.5 Defenses

Defending a site against these attacks requires input validation. Input can be validated on two different levels: either by the Web application or the Web server. The Web application can implement a function that parses all user input, handling dangerous characters/commands or rejecting the input. At the Web server level a solution such as the ModSecurity [13] module for the Apache server can be installed. This module can be used to validate all input before it is handed over to the Web application.

Successfully defending a site against SQL injection and XSS requires a constant focus on these problems. Whenever a new feature is added to an application there is a risk of introducing vulnerabilities. Even skilled and experienced programmers, who know about these attacks, make vulnerable applications. For a more detailed description of how to create more secure Web applications, [3] is highly recommended.

4 Collecting Social Security Numbers

SSNs are used around the world for purposes they were not initially intended to be used for. This has major implications. Problems arise when SSNs are used for any degree of authentication [14]. In order to investigate these issues further we decided to check how SSNs are used in Norway, and how much information we could obtain from governmental Web applications using our knowledge of the Norwegian SSN structure.

The structure of Norwegian SSNs is shown in Table 3 and described in [15]. Knowledge of the SSN structure makes it possible to generate all valid SSNs for any given day/month/year for both men and women. SSNs are not secret, but access to SSNs connected with personal information is restricted in Norway.

We have located several Web applications that allow us to filter valid SSN numbers belonging to real persons. E.g. all governmental employees are members of a pension fund. On the pension fund's Web site it is possible for members

The Norwegian SSNs consist of 11 digits: $x_1x_2x_3x_4x_5x_6i_1i_2i_3c_1c_2$	
$x_1x_2x_3x_4x_5x_6$	Birth date (ddmmyy)
$i_1i_2i_3$	Individual number. Highest available number for that day is used for each person. For persons born in 1855–1899 the possible numbers are 500–749, anyone born between 1900 and 1999 are given a individual number between 000 and 499, and finally for those who are born in 2000–2054 the numbers 500–999 are used. Girls have an even i_3 while boys have an odd i_3 value.
c_1c_2	Control digits. Used to detect every incorrect SSN with one number wrong or any mixing of two numbers $c_1 = 11 - (3x_1 + 7x_2 + 6x_3 + x_4 + 8x_5 + 9x_6 + 4i_1 + 5i_2 + 2i_3) \bmod 11$ $c_2 = 11 - (5x_1 + 4x_2 + 3x_3 + 2x_4 + 7x_5 + 6x_6 + 5i_1 + 4i_2 + 3i_3 + 2c_1) \bmod 11$ if c_1 or c_2 is 10 mod 11, then the SSN is rejected and the next possible SSN is chosen.

Table 3: Structure of Norwegian SSNs.

to apply for loans, a process that uses SSNs for identification. By supplying an SSN belonging to a member, access is granted to a page displaying the member’s full name, home address, and name and address of the member’s workplace.

To test the malicious data mining possibilities in such a portal, we wrote a simple Python script with less than 30 lines of code, which collected SSNs and the corresponding names and addresses from the Web application. We started off with the birth dates of all the members of the Norwegian cabinet and used our script to find their SSNs. The script accomplished this by generating all possible SSNs for given dates, posting them to the portal and logging the answers. Note that such a script also makes it possible to build a database containing all the SSNs and names of all the members of the pension fund (approximately 1,000,000). We have notified the pension fund about these issues on several occasions.

It is possible to combine such an attack with information gathered on other portals. For example, the portal for the tax department allows users to enter their SSNs to get a new tax card, but it gives an error message if the entered SSN does not belong to anyone. This fact makes it possible to do a binary search to determine the range of SSNs that belongs to real persons for any given day since SSNs are assigned in chronological order. Consider all persons born in Norway on 01.01.2001. The first girl to be registered that day would be given SSN number 01010199952 (boy: 01010199871), the second girl would be given 01010199790 (boy: 01010199448—birth date 01.01.01 with individual number 996 does not exist since $c_2 = 10 \bmod 11$ and so on. By executing the previously mentioned binary search we can find the last number assigned for that day. Hence, we then know the whole range of SSNs assigned for that particular day.

4.1 Using SSNs to do mischief

Many portals use SSNs to identify visitors. However, since there is no established infrastructure supporting authentication, many services only “authenticate” users based on knowledge of a valid SSN and perhaps a name. There are

several examples of this practice in Norwegian Web portals, e.g. you can order a new bank account, apply for loans, order a new tax card, and order a health card by entering a name and an SSN.

This allows several attacks; we can order/apply for a new tax card/health card for all Norwegians by using a simple script that logs on with every possible SSN number or any set of valid SSN numbers. Hence, we can affect service access for other users and create a lot of disturbance and manual work for governmental entities.

In 2003–2004 exact lists of assigned SSNs could be used to brute-force online bank accounts since several Norwegian Internet banks used SSNs for identification purposes. A description of such attacks can be found in [16].

5 Conclusions

The vast majority of dynamic e-government Web applications are vulnerable to XSS or SQL injection. XSS enables an attacker to create URLs to government sites that, if visited, will display arbitrary content controlled by the attacker. Such a malicious URL can further be used in Social Engineering attacks for the purpose of stealing passwords and cookies, or even spreading a false news story. SQL injection can potentially give an attacker the possibility to manipulate databases and in some cases execute arbitrary code on the database server.

These simple attacks have been known for several years, but still e-governments are vulnerable. More and more services are offered to the public through the portals, and more and more government databases are connected with these services, increasing the risk of malicious data mining, identity theft and compromising the integrity of the databases.

In addition to securing the Web applications, further steps are needed to secure the online services. E-government requires at least some kind of infrastructure to authenticate the users of the services, other than that of a username and password. A proper governmental PKI would make it possible to develop governmental Web applications implementing the required level of authentication and integrity.

References

- [1] The Digital Task Force, “The danish egovernment strategy 2004-2006,” last visited: October 26, 2005. [Online]. Available: http://e.gov.dk/uploads/media/strategy_pixi.pdf
- [2] European Commission, “egovernment services yield real benefits for eu citizens and businesses,” last visited: October 26, 2005. [Online]. Available: <http://europa.eu.int/rapid/pressReleasesAction.do?reference=IP/05/41&format=HTML&aged=0&language=EN&guiLanguage=en>
- [3] S. H. Huseby, *Innocent Code: a security wake-up call for Web programmers*. Wiley, 2004.
- [4] CERT, “Cert advisory ca-2000-02 malicious html tags embedded in client web requests,” 2000. [Online]. Available: <http://www.cert.org/advisories/CA-2000-02.html>

- [5] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," RFC 1738 (Proposed Standard), Dec. 1994. [Online]. Available: <http://www.ietf.org/rfc/rfc1738.txt>
- [6] C. Anley, "Advanced sql injection in sql server applications," A NGSSoftware Insight Security Research (NISR) Publication, last visited: October 26, 2005. [Online]. Available: http://www.nextgenss.com/papers/advanced_sql_injection.pdf
- [7] Google, "Directory over countries," last visited: June 12th, 2006. [Online]. Available: <http://directory.google.com/Top/Regional/Countries/>
- [8] G. Anzinger, "List over governments on the www," last visited: June 12th, 2006. [Online]. Available: <http://www.gksoft.com/govt/en/>
- [9] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in *WWW '04: Proceedings of the 13th international conference on World Wide Web*. New York, NY, USA: ACM Press, 2004, pp. 40–52.
- [10] ISO, "ISO 3166 country code lists," last visited: October 26, 2005. [Online]. Available: <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html>
- [11] Nationsonline.org, "First, second and third world," last visited: October 26, 2005. [Online]. Available: http://www.nationsonline.org/oneworld/third_world_countries.htm
- [12] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*. Addison-Wesley, 2004.
- [13] Modsecurity, "Open source web application firewall," last visited: October 26, 2005. [Online]. Available: <http://www.modsecurity.org/>
- [14] C. J. Hoofnagle and E. Mierzwinski, "U.S. PIRG answers to Chairman Shaw's questions on social security number privacy," 2004, last visited: June 12th, 2006. [Online]. Available: <http://www.epic.org/privacy/ssn/ssnanswers7.2.04.html>
- [15] E. Selmer, "Personnummerering i Norge: Litt anvendt tallteori og psykologi," *Nordisk matematisk tidskrift*, 1964.
- [16] K. J. Hole, V. Moen, and T. Tjøstheim, "Case study: Online banking security." *IEEE Security & Privacy*, vol. 4, no. 2, pp. 14–20, 2006.