

Practices in Software Security

Yngve Espelid



Thesis for the degree of Philosophia Doctor (PhD)
at the University of Bergen, Norway

2008

Contents

Abstract	5
Acknowledgements	7
Practices in Software Security	9
Motivation	9
What is Software Security?	10
Software Development Life Cycle	11
Risk Analysis	12
Summary—Paper I:	
Next Generation Internet Banking in Norway	12
Summary—Paper II:	
Open Wireless University Networks	13
Implementation Challenges	13
Summary—Paper III:	
Security Pattern for Input Validation	14
Summary—Paper IV:	
Simplifying Client-Server Application Development with Secure Reusable Components	15
Summary—Paper V:	
A Reflection-Based Framework for Content Validation	15
External Review	16
Summary—Paper VI:	
A Proof of Concept Attack against Norwegian Internet Banking Systems	16
Summary—Paper VII:	
Robbing Banks with Their Own Software—an Exploit against Norwegian Online Banks	17
Contributions	17
Impacts on Society	18
Paper I:	
Next Generation Internet Banking in Norway	21
Paper II:	
Open Wireless University Networks	43
Paper III:	
Security Pattern for Input Validation	57

Paper IV: Simplifying Client-Server Application Development with Secure Reusable Components	69
Paper V: A Reflection-Based Framework for Content Validation	79
Paper VI: A Proof of Concept Attack against Norwegian Internet Banking Systems	89
Paper VII: Robbing Banks with Their Own Software—an Exploit against Norwegian Online Banks	97

Abstract

Software continues to permeate all aspects of our life, at the same time making societies vulnerable as our appetite continues to grow for more complex and sophisticated systems. This dissertation focuses on software security, presenting seven papers related to best practices for developing secure software. The contribution of this work is discussed in relation to the software development life cycle, identifying where these best practices can be applied.

Acknowledgements

It has been three fantastic years doing research in applied software security as part of a devoted and highly talented research group at the Department of Informatics.

Thanks to Khalid Azim Mughal for supervising me during my Masters and for continuing giving positive and valuable feedback during my time as a Ph.D. student.

Thanks to Kjell Jørgen Hole for encouraging me to take a Ph.D. in this field and being my mentor throughout the period. Your commitment and concern have been invaluable sources of inspiration.

Thanks to Lars-Helge Netland for the close collaboration, enjoyable conference trips, and for arranging all the social activities contributing to a great working environment. Thanks to André N. Klingsheim for all collaboration and for getting hooked on golf along with Mr. Netland and me. I will stop under-reporting my scores so that we all can continue to play golf in peace.

Thanks to all members of the NoWires Research Group for making every workday a pleasure. Thanks to the department's student adviser, Ida Holen, for making everything run smoothly, allowing me to focus on my research.

Thanks to my family for all the support. Hopefully, you will appreciate my efforts to make your Internet-life a tiny bit safer.

Practices in Software Security

Motivation

Software is everywhere. Software can be a vital part of operations even in the most unimaginable areas. Cars are usually looked upon as all-mechanical systems, but in recent years the car industry has incorporated electronics for control and maintainability. In October 2007, General Motors presented their vision of software-based cars, where electronic systems are to replace mechanical ones, e.g. for steering and wheel motion [1]. Utilizing emergent technologies, cars are envisioned to collaborate with each other and interact with their environment to accomplish goals such as improved passenger safety. The vision culminates in passengers just telling their cars where to go and illustrates the main driving force in software development, namely functionality.

Software simplifies everyday life. Today, the scenario of societies with self-driving cars is closer to science fiction than reality. But software simplifying life on a daily basis is far from fiction. With the rise of the Internet, many services have moved online and are thereby more available to the general public. A prime example is online banking where customers do banking operations in the comfort of their own living rooms. Another example is how e-mail, instant messaging, and more recently, online communities such as Facebook [2], have changed the way people communicate. Today's communication technologies make it is easier to create and maintain large contact networks.

Society depends on software. Aviation relies heavily on software, as both aircrafts and control centers are equipped with complex software systems facilitating the explosive increase in air travel worldwide. In all sectors of society, software is playing an increasingly important role, e.g. advancing health care with improved equipment for analysis and treatment. In general, both people and industry lean on software for conducting their daily business.

The importance of secure software is evident considering the three statements emphasized above. This thesis focuses on how to develop secure software and starts with explaining the concept of *software security* and the significance of incorporating security related activities in the Software Development Life Cycle (SDLC). The scientific papers that make up this thesis relate to practices for secure software development. The papers are therefore summarized in sections where these practices are explained in more detail, while the full versions of the papers are included in the appendix. The contributions made by the papers are summarized in the conclusion of the thesis.

What is Software Security?

In a tutorial paper from 1975, Saltzer and Shroeder explore basic principles of information protection [3]. The authors focus on computer systems sharing information among multiple users; a secure system being able to prevent all unauthorized information usage. The term *security* constitutes techniques deployed to achieve secure systems, either realized in software, hardware, or physical installations. It is emphasized that the field lacks a methodology for developing secure systems and based on that, eight design principles are given to guide future system development.

The world has changed since 1975. The global network of interconnected software-enabled devices has grown formidably. In the 1990s the public started using the Internet, and in the last decade the number of distributed software systems running over the Internet has exploded. Satisfying business demands such as functionality and time-to-market, software that is still in a vulnerable state continues to be deployed and exposed to the public at large. Easily accessible systems handling valuable assets have led to favorable conditions for criminal activity. It is apparent that we still lack methodologies for developing secure software.

With the prominent need to protect information, software security has become a highly active research field. The current situation can be seen as a race between developers and scientists trying to build more secure software and hackers/crackers trying to break the systems we have today. The security level that hackers face depends on the knowledge and experience of individual team members building the system, and how the development process has embraced that knowledge. In [4], McGraw defines software security as the idea of engineering software so that it continues to function correctly under malicious attack. Through a series of articles, current best practices are discussed and how these ought to be incorporated in the SDLC.

In [3] security violations are categorized as

- unauthorized information release,
- unauthorized information modification, and
- unauthorized denial of use.

These categories respectively match the three main security goals described in nearly all security textbooks: *confidentiality*, *integrity*, and *availability*. In short, unauthorized persons must not be able to access or modify information, or be able to deny authorized persons the same actions.

The scenario with self-driving vehicles highlights the importance of achieving these security goals. If societies are to accept and build infrastructures realizing this vision, the requirement of passenger safety must be fulfilled. Consider two cars communicating over a wireless link while driving in the same direction. The speed of the car in front sets a “speed limit” for the following car. This limit is communicated over the wireless link. If someone could change this metric by intercepting and resending it, dangerous situations could occur. Hence, the *integrity* of the communication is important.

What if a malicious person tried to affect the *availability* of the information by jamming the wireless communication between the vehicles. The trailing car would act as if the car in front didn’t exist. Hopefully, the cars would also be equipped with sensor-based systems, enabling the trailing car to notice the presence of the car in front.

The supporting infrastructure could collect lots of information, e.g. on people’s movements. In the wrong hands, such data could easily be misused. Regulating the *confidentiality* of this information would be a necessity, protecting each individual’s right to privacy.

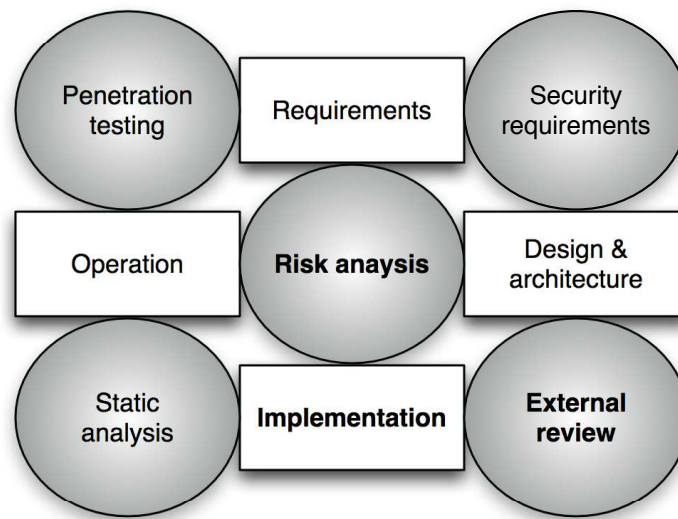


Figure 1: Development phases and software security best practices

Software Development Life Cycle

The well-known SDLC model known as the *waterfall model* was first published in 1970. The name reflects how it cascades from one phase to another; from requirements to design; from design to implementation; and so on. More flexible processes have later been published, ranging from evolutionary approaches, formal development, and development based on reuse [5]. No matter which development process a company follows, it should incorporate software security best practices [4]. Viewing security as an add-on to a system by appending security mechanisms as a final touch, has resulted in too many insecure systems.

Figure 1 illustrates how the development phases come into contact with best practices focusing on security. The figure is by no means complete with regard to development phases, security best practices, or where these interact. The idea is to highlight a process-independent view of software security. The purpose of each best practice in the figure can be summarized as follows:

security requirements, specifying data protection needs;

risk analysis, looking for risks to the business, systems, and processes;

external review, letting someone outside the development team analyze the system;

static analysis, reviewing code using tools and security experts; and

penetration testing, trying to actively break the running system.

Recommended security literature covering best practices in software security include [4, 6, 8]. As emphasized in Figure 1, risk analysis, implementation challenges, and external review are the main research areas covered by this thesis, and are further discussed in the following sections.

Risk Analysis

In [6], the authors discuss different responses from job applicants to the challenge of *how secure* the company’s software should be. A candidate who answers something similar to “secure against all attacks” or “as secure as it can possibly be” fails, while a more balanced view of making the software *just secure enough* is applauded. Doing risk analysis is a key ingredient in achieving software with just enough security built in. The vague wording of the term ‘just enough’ highlights that risk analysis is subjective, based on the skills and experience of the analyst.

The success of a risk management process can only be established in retrospect. The banking industry have shown themselves capable of managing risks associated with credit card and payment solutions. By adapting their systems to a changing environment, they have kept losses due to fraudulent activities at an acceptable level. So, what is risk in the context of software security? An accepted definition is that a risk is *a function of the level of threat, vulnerability, and the value of the information asset* [7]. Hence, for there to be a risk, there must be a threat agent capable of exploiting a vulnerability, and in doing so, cause some kind of impact. To do risk management, companies must be able to find their own vulnerabilities, determine the impact of possible exploits, and analyze their business environment in order to pinpoint threat agents.

Risk management is not a one-time exercise done somewhere in the SDLC. Systems, their environments, asset values, and development teams all change with time. Hence, risk management is a continuous process of identifying and keeping track of risks over time as a software project unfolds [8]. In this thesis, two papers relate to risk management. Paper I [9] analyzes the architecture and design of a security infrastructure established by the Norwegian banking industry, while Paper II [10] analyzes the risks associated with deploying an open wireless university network.

Paper I: Next Generation Internet Banking in Norway [9]

In the last decade, many large-scale security frameworks implementing Public-Key Infrastructures (PKIs) have been introduced worldwide. The Norwegian banking industry has established a PKI called BankID. The goal is to offer a national ID infrastructure used by government agencies and commercial companies to authenticate individuals and to provide legally binding signatures with a high degree of non-repudiation.

By evaluating public information on the architecture and design, as well signing up and using the system, a risk analysis of BankID is performed in Paper I. The evaluation is done from the customers point of view and focuses on customer authentication, non-repudiation service, and customer privacy.

It was found that the overall risk to BankID customers is significant. The list below is a summary of the most significant risks customers take when using the security infrastructure. The paper also suggests how to mitigate the identified risks.

- BankID is particularly vulnerable to Distributed Denial-of-Service (DDoS) attacks at the *application layer*. By generating a set of Social Security Numbers (SSNs) and performing a couple of wrong login attempts per SSN, an attacker can close down customer accounts.
- A DDoS attack can be followed by phishing e-mails targeting bank customers, enticing them to give up their credentials in the belief that they will regain access to their accounts. An attacker can instead empty out accounts belonging to customers taking the bait.

- Man-in-the-Middle (MitM) attacks can be combined with phishing schemes enabling attackers to empty out customer accounts. The MitM attack takes advantage of a vulnerability in the implementation of the authentication procedure, and makes it possible for attackers to steal sessions initiated by BankID customers.
- The lack of an independent third party in the non-repudiation service and the banks' security-through-secrecy policy give them an advantage over their customers during conflicts involving repudiation of digital signatures.

Paper II: Open Wireless University Networks [10]

Many universities offer employees and students with mobile terminals access to services in their information systems through wireless access points. A wireless network provides a better learning environment, e.g. enabling easier access to educational resources and facilitating teamwork at any time and from anywhere on campus. Usually, a university employs an authentication scheme to determine who can access their information system, often authenticating users at the network perimeter. Such authentication mechanisms are often seen as intrusions keeping users away from their primary tasks and forcing them to remember secrets and/or possess authentication devices.

By removing user authentication from the network infrastructure and only performing service level authentication the usability of the system increases. Such an open system can lead to increased use of information resources and consequently a better environment for learning. In addition, citizens can take advantage of effortless access to library catalogs and the Internet.

Considering a general open wireless network, it is discussed in Paper II how to mitigate risk associated with

- possible illegal downloads of e.g. music, movies, and child pornography,
- terminal-to-terminal attacks over a direct wireless link,
- attacks on local networks,
- anonymous attacks on remote networks,
- negative press coverage caused by incidents occurred on the network, and
- privacy concerns.

Finally, it is discussed how an IT department can vary the degree to which a network is open. Weighing the risks against the advantages, a university can make a decision whether to deploy an open network.

Implementation Challenges

In [11], Gries describes the “software crisis” back in 1968 having symptoms such as frequently missed deadlines, cost overruns, buggy software, and the inability to teach programming. Since then, both programming languages and software development methodologies have evolved, but a diagnosis of today’s software design and development would probably bear great resemblance to that of 1968. Gries believes that basic programming goals like simplicity and beauty have been lost in today’s teaching curriculum, as well as correctness concerns and methodology.

Following up the train of thought on simplicity, the trinity of trouble, *complexity*, *extensibility*, and *connectivity*, makes it difficult to develop secure software [12]. The abundant demands for new functionality often render modern software systems large and

complex. Adding to the intricacies are extensible software designed to incorporate foreign code, maybe not even written at the time of deployment, and the common use of the Internet as a communication channel in distributed systems. Complex software systems are difficult to analyze, making it plausible that bugs remain hidden even after testing and security analysis have taken place. Revisiting the example of software-based cars, the S-Class of Mercedes-Benz was reported in an article from 2003 to contain more than 50 controllers and over 600,000 lines of code [13].

This thesis includes three papers related to implementation challenges we face today. They all deal with input validation to some extent. The motivation is the recurring security breaches due to improper input handling. Examples include attacks exploiting vulnerabilities like buffer overflows, SQL injections, and cross-site scripting in web applications [14]. Problems related to input validation still figure high on the OWASP Top 10 list in 2007 over the most serious web application vulnerabilities [19]. The problem of handling input is not, however, limited to a web context. It is equally important in distributed software systems running over other protocols.

Input, as well as output, can be viewed as structured content. The structure is meta-information, i.e. it describes the format in a given protocol. The content is the actual information populating the structure. In essence, I/O can be defined as structured content exchanged between sub-systems of a larger system. Input validation entails both *syntax validation*, i.e. that the input adheres to the protocol, and *content validation*, e.g. that a due date for a new bill is the current date or a date in the future.

Proper sub-system communication is essential for a well-functioning system. How sub-systems exchange information depend on how they interface. Factors affecting these relationships include the communication techniques offered by the technology and how these mechanisms are made available through APIs. Over time, the communication forms tend to change, often driven by new performance and security requirements. The vast amount of opportunities combined with rapid changing technologies make it hard for developers to achieve the simplest, while secure enough, solutions.

Patterns have become a popular way of sharing structured knowledge and experience from successful projects in traditional software engineering. In [16], a collection of security patterns makes security knowledge readily accessible in a form usable by nearly all people involved in systems engineering, from planning to operations. From another angle, attack patterns focus on software vulnerabilities educating developers on the mindset and techniques of an attacker [12].

Paper III [15] describes a security pattern for input validation in web applications, while Paper IV [17] describes a secure reusable communication component that allows developers to focus on application-level programming instead of tedious networking issues. Paper V [18] builds on papers III and IV and describes a reflection-based framework for content validation.

Paper III: Security Pattern for Input Validation [15]

Paper III introduces a security pattern for input validation in web applications, and uses an online banking scenario as a motivating example. It describes how a fictive bank experiences fraudulent activity against their online banking solution. When analyzing their logs, other suspicious activities are also discovered. This example illustrates the context in which to apply the pattern and highlights the problems that must be resolved. The pattern provides an abstract object-oriented view on the structure of the solution, and offers an airport security checkpoint analogy to explain the dynamics.

The pattern then delves into a more concrete implementation of the solution, followed

by a detailed description of how the fictive bank solved their problems, e.g. specifying the validation rules defined to protect their application. A list of known technologies facilitating input validation are given, followed by expected benefits and possible liabilities from applying the pattern.

Paper IV: Simplifying Client-Server Application Development with Secure Reusable Components [17]

Paper IV presents a communication component that reduces the complexity involved in engineering secure client-server applications, thereby enabling software practitioners to develop more secure systems. When the communication technology changes, the component is updated, and all software projects utilizing the component can easily take advantage of the new technology.

Complexity is addressed through abstraction, reducing some of the tedious network programming to a matter of configuration. The goal is to properly secure transmission of data between communicating parties. This is achieved using core security services offered by a PKI, namely *authentication*, *integrity*, and *confidentiality*, enabling developers to use these services in conjunction with the applications' communication protocol.

Developers can focus on developing application logic and communication component configuration. The application logic has the responsibility for initializing, starting, and stopping the component, as well as handling the packets it produces. The configuration controls component behavior: e.g. to operate as a server or a client, and which threading model and transport mechanism to use when handling communication. To use the core security services, the location of key and trust material must be specified. Paper IV illustrates all these aspects in the context of implementing an HTTP server utilizing secure communication.

Paper V: A Reflection-Based Framework for Content Validation [18]

Paper V describes a reflection-based framework for content validation. The framework separates the inspection of data from the application logic, making it more feasible to construct and maintain a meaningful set of validation rules. The main goal of content validation is to ensure that input conforms to policy rules specified for the system domain. Such rules can define valid structure in credit card numbers, e-mail addresses, etc.

It is likely that content validation requirements change over time, resulting from e.g. further application development, feedback from the user community, or anomalies detected during inspection of server logs. It is therefore important to design a solution that can be quickly adjusted to meet changed requirements. The framework is flexible and can be integrated into almost any distributed object-oriented software system. Deployment only requires a basic understanding of XML, and expects developers to create a trust model of their own software architecture.

The solution does not dictate a specific application type or domain. It uses XML and reflection to tackle changing validation requirements. These two technologies are promising candidates for extending the longevity of software, and can achieve platform-independence [20]. Paper V uses a scenario for bill payment to show how content validation is set up, and illustrates how the solution helps to structure the validation process through defining categories of validation rules.

External Review

What is the motivation for keeping system information secret from the outside world? One obvious incentive is retaining a new innovative solution, getting a head start on the competition. But can secrecy also help to improve the security of a system? Solutions utilizing cryptography depend on the secrecy of vital information. In a typical X.509 PKI, private keys must be solely available to the entity identified in the matching certificate [21]. The disclosure of the private key belonging to the root certificate authority breaks the whole chain of trust, rendering the whole system useless. However, the cryptographic algorithms used in such systems can be safely published, since the peer review process, involving scrutiny of the algorithms by independent experts, ensures that these algorithms can withstand attacks.

There is an ongoing debate concerning open vs. closed development in the software community. A common view is that closed projects have a greater probability of having bugs in the implementation and flaws in the design due to the limited amount of eyes reviewing the system. On the other hand, in an open project, malicious individuals can easily assess a system and exploit any vulnerabilities they might find. They may even be able to introduce a backdoor by participating in the development of the system. These last considerations may lead companies to follow a *security-through-secrecy* policy, i.e. being very restrictive on who gets to participate in the design and implementation of their products. There are many forces that work against such a strategy. Over time the chances of information leakage increase due to change in staff and disgruntled employees. Also, it can be hard to estimate how much information a malicious person can obtain, e.g. by looking at public documentation and reverse engineering client software.

A more productive strategy would be to borrow the best ideas from both camps. A company using their own development team to evaluate the system they just created is a flawed strategy. It is important to realize that the sense of ownership and the tendencies towards groupthink, limit the value of self-evaluation. In terms of security, one can view the combined mindset of the whole development team as the foundation for the product's security architecture and design. An attacker may have a different mindset, allowing him to easily circumvent security mechanisms. A more viable strategy is therefore to have external experts review the system throughout the development process.

Papers VI and VII highlight the importance of using external reviews when developing secure software. Both papers spur from the discovery of the MitM vulnerability found during the risk analysis [9] (Paper I) of BankID, the new security infrastructure developed by the Norwegian banking industry. The further investigation can be compared to *red teaming*, i.e. analyzing the infrastructure from an adversary's point of view. By reverse engineering client software, studying publicly available information, and personal use of the system, proof of concept code was developed and executed to demonstrate the seriousness of the MitM attack. The approach was chosen because a request to see in-depth descriptions of the architecture and design was met with a non-disclosure signature prerequisite. Paper VI [22] gives a high-level view of the proof of concept attack, while Paper VII [23] goes into more details on the exploit and disclosure process.

Paper VI: A Proof of Concept Attack against Norwegian Internet Banking Systems [22]

In November 2007, over 800,000 people had been enrolled in the BankID security infrastructure, and by the end of 2008, it is expected to include nearly all Norwegian bank customers. Today, the infrastructure is mainly used for authentication in Internet bank-

ing, but is extending into the government sector and e-commerce in general. Hence, the infrastructure is becoming more and more attractive to criminals.

Paper VI describes a practical MitM attack against online banking applications using BankID. The infrastructure is studied through adversarial eyes, starting with identifying main entities along with their credentials and duties. Further steps focus on the authentication protocol, collecting information from public documentation, reverse engineering client software, and personal use of the system.

The attack gives an adversary access to customer bank accounts in two different online banking systems, and does not depend on malicious software being installed on the victim's computer. The paper pinpoints the need for a proper authentication mechanism in the BankID client software and end-to-end encryption of all communication.

Paper VII: Robbing Banks with Their Own Software—an Exploit against Norwegian Online Banks [23]

No publicly available independent third party evaluation of BankID confirms that the security infrastructure meets a minimum of security requirements. Such evaluations are a must when developing nationwide identity systems. The technical findings during the investigation of the MitM vulnerability, along with the experience gained from the disclosure process, reveal the need for a more thorough analysis.

Paper VII delves further into the details of the authentication protocol and elaborates on technicalities of the exploit. It is emphasized how the exploit makes use of well-known attack strategies, but also how it capitalizes on customers' trust by using the bank's own client software as an attack tool.

Paper VII includes a discussion on the disclosure process from the discovery of the MitM vulnerability in March 2007 to the countermeasures introduced in November 2007 and January 2008. The process entailed notifying system owners and national authorities of the BankID vulnerability, and providing technical descriptions of how the vulnerability could be turned into an exploit.

Contributions

This thesis contributes to best practices in software security. The importance of risk analysis has been highlighted through analyses of a security infrastructure developed by the Norwegian banking industry, and of risks involved when deploying an open wireless university network. Implementation challenges have been met by focusing on how knowledge can be transferred from security experts to developers using patterns, how code reuse can reduce complexity, and how projects can benefit from designing solutions that can be quickly adjusted to meet change in requirements. Lastly, the thesis focuses on why external reviews are vital when developing secure software.

Two components have been developed to address implementation challenges. The first is a prototype handling communications, encapsulating networking programming, shielding developers who can focus on developing business logic. The second prototype offers a flexible solution for content validation.

Proof of concept code has been developed to demonstrate the seriousness of the MitM attack on BankID. The work related to this attack has resulted in a patch inserted into the BankID authentication mechanism in January 2008.

Impacts on Society

Analyzing and discussing how to mitigate risks involved when deploying an open wireless university network help IT departments to weigh risks against advantages, making well-considered decisions on whether or not to deploy an open wireless networks.

The risk analysis of BankID and the further investigation of the MitM vulnerability were performed on behalf of Norwegian tax payers to evaluate the risks involved when enrolling in the nationwide security infrastructure. In November 2007, a feature article in a large Norwegian newspaper expresses doubts about the security of BankID [24]. The article sparked a debate during which the existence of the MitM exploit was disclosed to the public [25]. The debate caught the attention of members of the Norwegian parliament, and is the foundation for a question to the Minister of Government Administration and Reform in a Question Time session on the 16th of January 2008 [26].

During 2007, different newspapers and online news outlets wrote about the work involving BankID, thereby increasing public awareness of software security, especially in solutions which are candidates to become a new national identity system in Norway

Bibliography

- [1] Computerworld. GM presents vision of software-based cars. Retrieved January 2008 from <http://www.computerworld.com.au/index.php/id;1737481711>
- [2] Facebook. Retrieved January 2008 from <http://www.facebook.com/>
- [3] J.H. Saltzer and M.D. Shroeder, “The protection of information in computer systems,” in *Proc. of the IEEE*, 63(9):1278-1308, September 1975.
- [4] G. McGraw, “Software Security,” *IEEE Security & Privacy*, vol. 2, no. 2, 2004, pp. 80–83.
- [5] I. Sommerville, *Software Engineering*. Addison-Wesley, sixth edition 2001.
- [6] M.G. Graff and K.R. van Wyk, *Secure Coding—Principles & Practices*. O’Reilly, 2003.
- [7] A. Jones and D. Ashenden, *Risk Management for Computer Security*. Elsevier, 2005.
- [8] G. McGraw, *Software Security—Building Security In*. Addison-Wesley, 2006.
- [9] K.J. Hole, T. Tjøstheim, V. Moen, L-H. Netland, Y. Espelid, and A.N. Klingsheim, “Next Generation Internet Banking in Norway,” submitted to *IEEE Security & Privacy*.
- [10] K.J. Hole, L-H. Netland, Y. Espelid, A.N. Klingsheim, H. Helleseth, and J.B. Henriksen, “Open Wireless University Networks,” to be published in *IEEE Security & Privacy*, 2008.
- [11] D. Gries, “What Have We Not Learned about Teaching Programming,” *IEEE Computer*, vol. 39, no. 10, 2006, pp. 81–82.
- [12] G. Hoglund and G. McGraw, *Exploiting Software—How to Break Code*. Addison-Wesley, 2004.
- [13] K. Grimm, “Software Technology in an Automotive Company - Major Challenges,” in *Proc. International Conference on Software Engineering (ICSE)*, 2003.
- [14] S.H. Huseby, *Innocent Code—A Security Wake-Up Call for Web Programmers*. John Wiley & Sons, 2004.
- [15] L-H. Netland, Y. Espelid, and K.A. Mughal, “Security Pattern for Input Validation,” in *Proc. Nordic Pattern Languages of Programs Conference (VikingPLoP)*, Helsingør, Denmark, September-October 2006.

- [16] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns—Integrating Security and Systems Engineering*. John Wiley & Sons, first edition 2006.
- [17] Y. Espelid, L-H. Netland, K.A. Mughal, and K.J. Hole, “Simplifying Client-Server Application Development with Secure Reusable Components,” in *Proc. International Symposium on Secure Software Engineering (ISSSE)*, Washington D.C., USA, March 2006.
- [18] L-H. Netland, Y. Espelid, and K.A. Mughal, “A Reflection-Based Framework for Content Validation,” in *Proc. International Conference on Availability, Reliability and Security (ARES)*, Vienna, Austria, April 2007.
- [19] OWASP Top 10, 2007. Retrieved January 2008 from http://www.owasp.org/index.php/Top_10_2007
- [20] J. Bishop and N. Horspool, “Cross-Platform Development: Software that Lasts,” *IEEE Computer*, October 2006, pp. 26–35.
- [21] C. Adams and S. Lloyd, *Understanding PKI—Concepts, Standards, and Deployment Considerations*. Addison-Wesley, second edition 2003.
- [22] Y. Espelid, L-H. Netland, A.N. Klingsheim, and K.J. Hole, “A Proof of Concept Attack against Norwegian Internet Banking Systems,” in *Proc. International Conference on Financial Cryptography and Data Security (FC)*, Cozumel, Mexico, January 2008.
- [23] Y. Espelid, L-H. Netland, A.N. Klingsheim, and K.J. Hole, “Robbing Banks with Their Own Software—an Exploit against Norwegian Online Banks,” submitted to the *International Information Security Conference (SEC 2008)*, Milan, Italy, September 2008.
- [24] K. Gjøsteen and K.J. Hole, “Nettbanking ikke trygg,” *Aftenposten* (17. Nov, 2007). Retrieved January 2008 from <http://www.aftenposten.no/meninger/kronikker/article2106538.ece> (In Norwegian)
- [25] K. Gjøsteen and K.J. Hole, “Nei, ennå ikke trygg,” *Aftenposten* (29. Nov, 2007). Retrieved January 2008 from <http://www.aftenposten.no/meninger/debatt/article2126133.ece> (In Norwegian)
- [26] Stortinget. Spørretime onsdag 16. januar 2008. <http://epos.stortinget.no/SpmTime.aspx?MoteId=7469&Dagsid=26493> (In Norwegian)

Paper I:
Next Generation Internet Banking in
Norway

Next Generation Internet Banking in Norway

Kjell J. Hole* Thomas Tjøstheim Vebjørn Moen
Lars-Helge Netland Yngve Espelid André N. Klingsheim

NoWires Research Group
Department of Informatics
University of Bergen

Version 0.9, last updated April 24, 2007

Abstract

The Norwegian banking industry has introduced a new security infrastructure for web applications, including Internet banking. The infrastructure, called BankID, has the potential to increase the security of today's web applications and facilitate new business opportunities. The authors consider BankID from the customers' point of view, analyze the risk the customers take when using BankID, and discuss how to mitigate the risk.

1 Introduction

Many countries, including Norway, Sweden, Denmark, Finland, Estonia, Austria, Belgium, and Canada, have introduced large-scale security frameworks implementing Public-Key Infrastructures (PKIs). In general, a PKI is a collection of hardware, software, processes, and people providing applications with security services based on public-key cryptography. The Norwegian banking industry has introduced a PKI called BankID. While BankID mostly authenticates Internet banking customers at the time of writing, the Norwegian banking industry wants BankID to become a national ID infrastructure used by government agencies and commercial companies to authenticate individuals and to provide legally binding digital signatures with a high degree of non-repudiation.

Two earlier papers [1], [2] analyzed the Norwegian Internet banking and Automatic Teller Machine (ATM) systems. This paper applies elements of risk management [3] to BankID. Since the Norwegian banking community

*Contact address: Professor K. J. Hole, Department of Informatics, University of Bergen, PB. 7800, N-5020 Bergen, Norway. E-mail: Kjell.Hole@ii.uib.no, Mobile: +47 920 38 164, Web: www.nowires.org.

declined to share technical information about BankID, we were forced to evaluate BankID exclusively from the customers' point of view. The evaluation was completed in March of 2007 and was only based on publicly available descriptions of the BankID architecture and design, as well as personal use of the system.

In the remainder of the paper, we first determine how BankID differs from a typical X.509 PKI, before carrying out a risk analysis of the end-user authentication, non-repudiation service, and customer privacy. The risk to the customers is found to be significant. We therefore suggest steps to mitigate the risk.

2 PKI primer

This section introduces a typical X.509 PKI [4]–[7].

2.1 Keys, signatures, and certificates

In general, each end-user/customer in an X.509 PKI generates his own pair of asymmetric cryptographic keys—one *private key* and one *public key*. The public key is available to all end-users in the PKI, while the private key is only known to its owner. To secure the private key, it is stored in an encrypted file on the end-user's computer, or better, on a tamper-resistant smart card. As an example, all end-users can utilize Alice's public key to encrypt a message to her, but only she can decrypt the message because nobody else has access to her private key.

Bob can use his private key to *digitally sign* a message, or document. During the cryptographic signing procedure a value, denoted the *digital signature*, is calculated over the message. Alice verifies Bob's digital signature by applying a cryptographic procedure which takes Bob's public key, the received message, and the signature as input. If somebody tampered with the message after it was signed, then the verification will fail, else Bob must have signed the message since only he had access to the private key.

An X.509 *certificate* binds a public key to an identifier, e.g. a personal name, an assigned user number, or a web address. The identifier points to the end-user or web site with the corresponding private key.

Commercial PKIs, including BankID, utilize different pairs of keys for digital signatures and encryption/decryption. For simplicity, we do not always differentiate between these pairs.

2.2 PKI architecture

An end-user requests a certificate from a *Registration Authority* (RA) by providing the information required to issue a certificate. The RA verifies the information and sends a certification request, containing the end-user's

public key, to the *Certification Authority* (CA). The CA generates the certificate and signs it with its own private key. A web site owner initiates a similar procedure to obtain a web site certificate.

A CA revokes a certificate when the public key should no longer be used. Certificate revocation is frequently caused by the termination of a customer relation. More importantly, the CA must revoke a certificate immediately if the corresponding private key is compromised. The CA keeps track of revoked certificates. Often, a CA maintains a digitally signed *Certificate Revocation List* (CRL) containing unique references to the revoked certificates, the dates they were revoked, as well as the reasons for revocation.

A PKI may have multiple CAs. For simplicity, we consider a strict, two-level hierarchy of CAs containing a single root CA on the zeroth level with a special self-signed certificate, i.e., the signature is generated by the root CA's own private key. The root CA generates and signs certificates to the CAs on level one in the hierarchy. These level-one CAs again issue certificates to the end-users in the PKI.

2.3 Transitive trust model

An entity X *trusts* another entity Y when X assumes it knows exactly how Y behaves. As an example, an end-user trusts a CA when she assumes that the binding between the name and the public key in an issued certificate is correct. If X trusts Y and Y trusts Z , then X also trusts Z , e.g., an end-user trusting a root CA also trusts a level-one CA. If she doesn't trust the root CA she will not use the PKI services.

The root CA is the starting point of all trust in a PKI. In a two-level CA hierarchy, certificate path processing extends trust from the root CA to a level-one CA. Consider an instance where Bob and Alice are issued certificates from different level-one CAs. To verify Alice's certificate, Bob first builds a path of certificates back to the root CA. The path consists of Alice's certificate, the certificate of the level-one CA that issued her certificate, and the root CA certificate. Using the root CA's public key, Bob then verifies the signature of the level-one CA certificate. Next, he extracts the public key from this CA certificate and uses it to verify the signature of Alice's certificate. Because Bob trusts the root CA, he now assumes that the content of Alice's certificate is correct.

2.4 Authentication

Authentication can be defined as the process of establishing an understood level of confidence that an identifier refers to an end-user or a web site. The authentication is said to be *strong* if the level of confidence is high.

Authentication in a PKI is based on certificates, the corresponding private keys, and a source of revocation information, e.g. a CRL. A simplified

authentication protocol illustrates the authentication process. When Bob wants to authenticate Alice, he first asks for her certificate and then uses the CRL to verify that the certificate has not been revoked. He also confirms Alice's ownership of the public key by building and verifying a certificate path to a trusted root CA. Bob now asks Alice to sign a random number, denoted the challenge, with her private key. If Bob verifies the signed challenge using Alice's public key, then he assumes he is communicating with Alice. The same protocol is carried out when Alice authenticates Bob.

The strength of the authentication relies on a well tested authentication protocol, such as the Secure Sockets Layer (SSL) protocol, on how securely the private keys are stored, and on the computational difficulty of calculating the private key corresponding to a public key. To maintain the authentication strength over time, the CAs must update their CRLs often and the lists must always be available to both parties during the authentication process.

2.5 Legal view of non-repudiation

Non-repudiation offers a person protection against a false claim by another person that a communication never took place. The two most commonly discussed types of non-repudiation are *non-repudiation of origin* in which a person cannot falsely deny having originated a message, or document, and *non-repudiation of delivery* in which a person cannot falsely deny having received a message.

From a legal point of view, non-repudiation consists of the ability to convince a third party that a specific message originated with, or was delivered to a certain person. Credible evidence is needed to persuade a judge, jury, or arbitrator. Both the quality and the presentation of the evidence determine the level of non-repudiation.

A non-repudiation service utilizing digital signatures can be built on top of a standard PKI. A high level of non-repudiation can be obtained if the basic PKI services are combined with the correct combination of legal and technical non-repudiation protocols. It is also essential that at least one trusted third party collects, validates, time stamps, signs, and stores relevant information [5, Ch. 9], [7, Ch. 4]. The third party must be able to withstand pressure from the communicating parties and present the non-repudiation evidence in an unbiased manner during a conflict.

It is important to note how the burden of proof is on the party wanting to rely on a digital signature. Hence, non-repudiation does not take away a person's legal right to refute a signature. A high level of non-repudiation only implies it will be possible to show, with high probability, that a person digitally signed a particular document even if he later denies it.

3 BankID explained

This section outlines the BankID architecture [8], [9] using the nomenclature established in the PKI primer.

3.1 Certificate types

There are three different types of 'user' certificates. The first type is a personal certificate for regular end-users, the second type is an employee certificate for end-users representing a company or an organization, and the third type is a certificate for web sites. All these certificates follow Version 3 of the X.509 Recommendation [4]. In addition, CAs, RAs, and other entities in the BankID infrastructure have their own certificates.

3.2 Architectural overview

The BankID architecture is divided into two main parts as depicted in Figure 1. The first part, referred to as the *central infrastructure*, is operated by the Norwegian Banks' Payment and Clearing Centre (also known as BBS). The central infrastructure contains CAs, a certificate *Validation Authority* (VA), central storage facilities for cryptographic keys and certificates, and functionality for digital signing of documents. The *distributed infrastructure* comprises the BankID server, which is part of a web application, the RA at an end-user's bank, and the BankID client running on the end-user's computer.

3.3 Central infrastructure

The central infrastructure includes a two-level hierarchy with one root CA and multiple level-one CAs. The root CA is owned by the Norwegian Financial Services and Saving Banks Associations. While the level-one CAs are part of the central infrastructure, each level-one CA is owned by a separate bank (or group of banks). A bank uses its CA to issue certificates to its own customers. The root-CA certificate is valid for 26 years, while the level-one CA certificates are valid for 12 years [9].

The VA utilizes CRLs from the level-one CAs to determine if certificates have been revoked. This validation service is available to both the BankID server and client in Figure 1.

3.4 BankID server

A web application utilizing BankID, e.g. an online store or an Internet banking site, runs a BankID server. This server software is available in both the C and Java programming languages, and can be incorporated into the web application. The BankID server stores its certificate and private key in

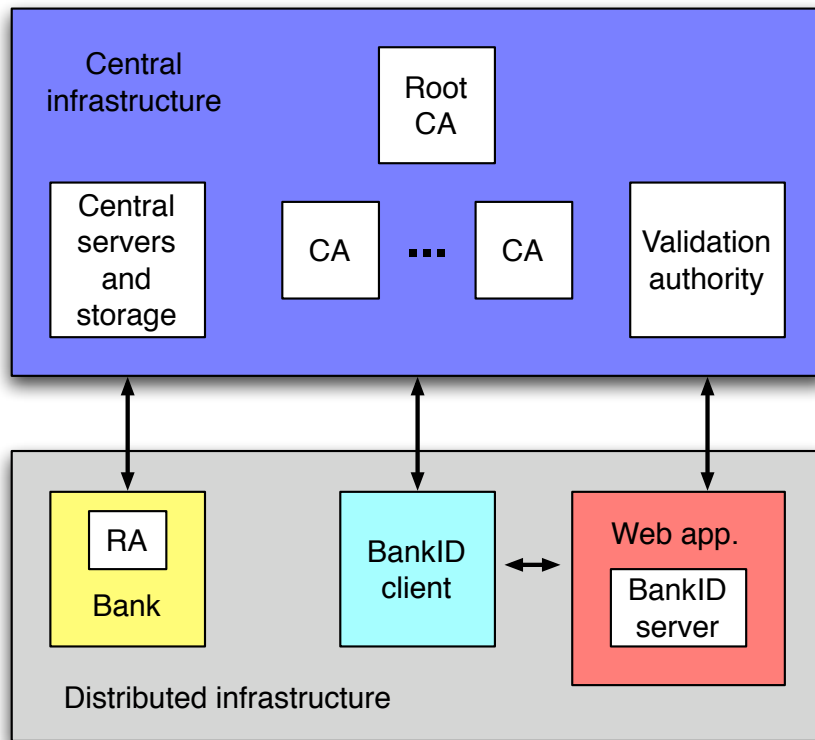


Figure 1: BankID architecture with lines indicating information flow.

a *Hardware Security Module* (HSM) or an encrypted PKCS #12 file [10]. HSMs also provide web applications with dedicated hardware for processor-intensive cryptographic operations.

3.5 Bank RA

Each bank operates its own RA software, which is likely to be integrated into the bank's customer service software. A new BankID customer requests a certificate from the RA using his Internet browser or by showing up in person at the local branch office. The RA sends a certification request to the central BankID infrastructure using the SSL protocol.

3.6 BankID client

The certification request from the bank RA starts the initialization of a new customer record. The central infrastructure first generates (at least) one public-private key pair. The private key is stored in a secure central database and the public key, as well as the request from the RA, are sent to the CA belonging to the customer's bank. The new certificate generated by the CA is stored on the central infrastructure. The customer downloads a Java applet to her Internet browser each time she wants to use the BankID

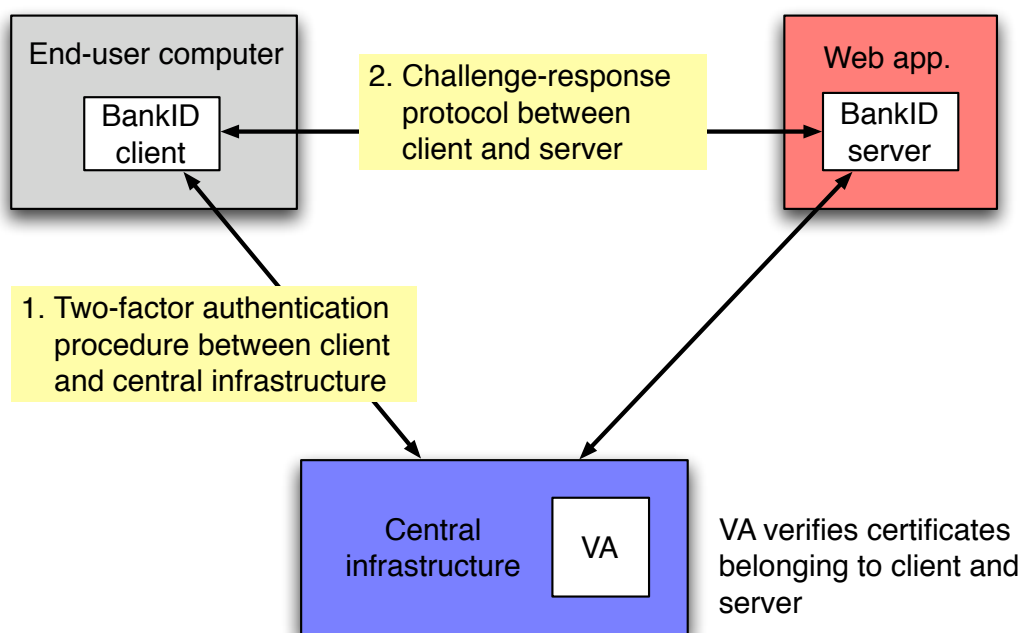


Figure 2: Overview of the BankID authentication procedure.

client. No software or information is stored permanently on her computer.

3.7 BankID authentication procedure

The *mutual* authentication between an end-user and a web application is divided into two parts as depicted in Figure 2. The first part, a two-factor authentication procedure (to be explained), authenticates the end-user to the central infrastructure and ensures that the user controls the access to her own centrally stored cryptographic keys. The second part is a challenge-response protocol between the BankID client and server. The protocol is similar to the one described earlier. All communications between the entities in Figure 2 are executed inside SSL tunnels.

The first part of the authentication operates as follows. Since an end-user can have multiple BankID certificates issued by CAs belonging to different banks, she first enters her Norwegian Social Security Number (SSN) into the BankID client. The central infrastructure responds with a list of all her BankID affiliations enabling her to choose the one she wants. Next, the client prompts the user for a one-time Personal Identification Number (PIN) which is verified by the central infrastructure. The one-time PIN is taken from a list of PINs supplied by the end-user's bank, or generated by a hardware token, often called a PIN calculator. A fixed PIN is sometimes needed to activate the PIN calculator. Finally, the client prompts the user for a fixed password used by the central infrastructure to get access to the end-user's private key.

The described procedures are essentially traditional two-factor authentication mechanisms based on something you have (the PIN calculator or list of PINs) and something you know (the fixed password and perhaps a fixed PIN needed to activate a calculator) [6].

The second part consisting of the challenge-response protocol is completed once the central infrastructure has access to the end-user's cryptographic keys. During the protocol execution, the central infrastructure carries out the cryptographic functions for the BankID client. The BankID server uses the VA to verify the certificate from the BankID client and vice versa.

The reader should note that BankID differs from a typical X.509 PKI because the central storage of asymmetric keys and the central execution of cryptographic operations make it necessary for a BankID client to transmit a one-time PIN and a fixed password over the Internet to the central infrastructure.

3.8 Non-repudiation

BankID aims to provide both non-repudiation of origin and delivery. Unfortunately, nearly all information about the legal and technical non-repudiation protocols and storage of non-repudiation information are kept secret. It is known, however, that signed documents contain the signed data, the signatures of the parties, and the results of the VA requests at the time of signing [8]. BankID again differs from a typical X.509 PKI since no trusted third party is used to establish non-repudiation information for dispute resolution [8], [9].

4 Risk of authentication service

It is convenient to define *risk* as the possibility of suffering harm or loss. There are several types of risks [3]. On one hand we have pure risk associated with natural accidents such as fires, floods, hurricanes, and earth quakes. Speculative risk on the other hand is related to man-made failures, e.g., terrorism and misuse. We'll only consider speculative risk, denoted 'risk' for simplicity.

Let a vulnerability be a flaw in BankID and let a threat be an adversary with the capabilities and intentions to exploit a vulnerability. The risk taken by BankID customers is a function of the exploitable vulnerabilities and the danger from the threats [3]. This section discusses the risk associated with the BankID authentication procedure and suggests how to mitigate the authentication risk.

4.1 Exploiting the BankID authentication procedure

The BankID client gets access to the cryptographic functionality in the central infrastructure after the end-user has provided the correct SSN, one-time PIN, and fixed password. Experiments have shown that if the customer enters correct SSN and a wrong PIN three times, then the access to the central infrastructure closes down and the customer must contact his bank to reopen the account. (Access is also denied if the end-user first types in correct SSN and PIN and then enters a wrong password enough times.) The described process can be automated. In a simple proof of concept, an executable script starts Internet Explorer 7, downloads the BankID client, and simulates a user logging in with the correct SSN and wrong PIN three times.

Because SSNs have a well-defined structure it is possible to generate a large set of SSNs containing SSNs belonging to BankID customers. (The set may also contain SSNs not belonging to customers. See [1] for details.) A small program can then run through the set of SSNs and close down the customers' accounts as described above. The attack can be spread over many computers to construct a Distributed Denial-of-Service (DDoS) attack at the application layer, which is very difficult to stop if the number of computers is large ($\geq 10,000$) [1], [11]. Unlike DDoS attacks at lower network layers, it is not necessary to transmit a large amount of dummy traffic for a long time to close down the accounts, it is only necessary to try to log into each account a small number of times. Because this efficient DDoS attack on the authentication procedure has the potential to deny all users access to BankID, it can be said that the authentication procedure represents a *single point of failure*.

During the fourth quarter of 2006, about 600,000 of the 2.3 million Internet banking customer in Norway had moved to BankID. If most of the remaining customers also move to BankID during 2007, as envisioned by the Norwegian banking industry, then BankID may well get more than two million end-users in the near future. Because they'll all rely on the same infrastructure, a successful DDoS attack will have severe economic consequences. Roughly two million end-users will not be able to access their accounts, transfer funds, or pay bills, and some of the online stores using BankID will not be able to sell goods. If BankID also becomes a national ID used by government agencies, then the consequences of a DDoS attack will be even more severe.

It has been argued that the likelihood of a DDoS attack against an online banking system in Norway is small because there is no group of people with strong enough motivation to carry out an attack. On the other hand, BankID will become a national "monoculture" [12] for online banking, which makes it easy to attack two million customers at the same time. As long as the BankID authentication procedure is not changed, it will not be difficult

to repeat a DDoS attack at the application layer. While traditional crackers may be reluctant to attack BankID because they fear the inevitable investigation by Norwegian government agencies concerned with national security, other groups wanting to attack the Norwegian financial system to create chaos may find BankID a tempting target.

Examples of potential threats are terrorist groups in strong opposition to the Western society in general, and Norway in particular. Cyber terrorists [13] are both able and willing to disrupt critical information structures to cause harm in order to advance their own political or religious agendas. Other terrorist groups without the needed skill set can hire or coerce crackers to carry out attacks. The danger of terrorist attacks in Europe, the increasing usage of BankID, and the increasing number of DDoS attacks against businesses in Norway reported by the Norwegian National Security Authority, lead us to conclude that the likelihood of a DDoS attack against BankID will grow during the coming years.

Observation 1 *Because the authentication procedure in BankID utilizes SSNs and denies an end-user access after a few wrong login trials, it is particularly vulnerable to DDoS attacks—just like the authentication procedures in the other Norwegian Internet banking systems. The potential DDoS attacks represent a growing risk to end-users and web site owners.*

4.2 Strength of end-user authentication

Let us consider the strength of the end-user authentication in BankID consisting of a traditional two-factor authentication procedure followed by (part of) a request-response authentication protocol as depicted in Figure 2. Assume that an adversary has obtained an end-user’s PIN calculator or list of PINs. Furthermore, the adversary knows the end-user’s fixed password, and—if needed—the fixed PIN used to activate the calculator. The adversary can then download the BankID client and give it access to the cryptographic functionality at the central infrastructure. The client can now complete the request-response authentication protocol with the BankID server. Hence, if the two-factor authentication is compromised, then the authentication of the end-user is compromised. Because BankID-member banks utilize different two-factor authentication mechanisms, the exact strength of the end-user authentication varies between the banks.

Observation 2 *The end-user authentication in BankID is no stronger than the two-factor authentication used in many older Internet banking systems.*

Surprisingly, the fixed password provided by the end-user during the first part of the BankID authentication procedure doesn’t always strengthen the end-user authentication. One particular BankID-member bank, which provides each end-user with a PIN calculator requiring an activating PIN,

lets any end-user ask for a new password by simply providing his SSN and a one-time PIN. The new password is displayed in the user's browser. Hence, if an adversary gets hold of the calculator and the activating PIN, he can also obtain a new valid password without knowing the old password!

4.3 Phishing/man-in-the-middle attacks

Each time an end-user downloads the BankID client, he also downloads HTML code containing parameters to the Java applet. While the applet itself is signed, the HTML code is not signed and changes to the parameters will not be detected by the user's Internet browser. Two parameters specifying URLs can be altered to make the BankID client communicate with the BankID server and central infrastructure through a proxy (server) controlled by a cracker. The proxy can be realized by software running on some computer.

To initiate this Man-in-the-Middle (MitM) attack, a phishing attack may first be used to trick a BankID customer into downloading modified HTML code together with the unaltered BankID client. When the customer starts a new session, the BankID client then connects to the proxy, which again connects to the BankID server and central infrastructure. We have developed proof-of-concept code to show that the proxy can steal the session after the user has authenticated himself to the central infrastructure and the BankID server.

MitM attacks are a well established form of deceit. In 2006, several Norwegian Internet banks, not based on BankID, were victims of MitM attacks. The likelihood of MitM attacks on BankID will increase as BankID gets more users and the system becomes a national monoculture.

Observation 3 *Combined phishing/MitM attacks can be used to steal sessions initiated by BankID customers because it is possible to change the addresses to which the BankID client connects.*

4.4 DDoS/phishing attacks

A DDoS attack closing down BankID customers' accounts can be followed by a phishing e-mail attack to let the same customers "know" how they again can use their accounts. It is clear from Observation 1 that it is particularly easy to execute a DDoS attack to close down accounts. An e-mail can then notify each customer about the account closure, which the customer can easily verify, and then entice the customer into entering a one-time PIN and a password at a fake banking site to open the account. The attacker must either call the bank to try to reopen the account, or wait for the bank to do it, and then use the stolen one-time PIN and fixed password to access the account before the legitimate owner.

No such combined DDoS/phishing attack has yet been reported in the press as far as we know. However, both DDoS attacks and phishing attacks are increasing in frequency, and it may only be a question of time before combined attacks occur.

Observation 4 *BankID is potentially vulnerable to combined DDoS/phishing attacks where customers are tricked into entering one-time PINs and passwords on fake Internet banking sites.*

4.5 Exploiting the BankID server

A web application using the BankID infrastructure incorporates the BankID server software. The private key and the certificate belonging to the web application are stored in an encrypted PKCS #12 file or an HSM. At least one version of the BankID server does not support the use of an HSM. In this case, the web site owner provides a fixed password to give the BankID server access to the decrypted file. If an outside cracker or rogue insider gets hold of the file and the password, then the web site owner's private key is exposed. An attacker with the private key can pass himself off as the web site owner. The seriousness of this threat is determined by how hard it is to get hold of the PKCS #12 file and how difficult it is to determine the owner's fixed password.

The encrypted PKCS #12 file has a known structure, which makes it possible to locate on a computer. When a cracker has access to this file, he can run a brute-force or dictionary attack to try to recover the accompanying password [14]. On the other hand, "social engineering" or "shoulder surfing" may be all that is needed. The attacker can also try to install a hidden camera or a hardware keylogger to obtain the password.

In the case where a cracker has no physical access to the computer, he may still be able to employ malicious software, or *malware*, to obtain the PKCS #12 file. The cracker can then run a dictionary attack to try to determine the password and obtain the private key. Alternatively, the malicious software can try to both copy the file and sniff the password using a software keylogger.

A cracker can implement a DoS attack by developing self-propagating malware which spreads to many computers and deletes the PKCS #12 files, thus, preventing web applications from utilizing BankID. Since it is only possible to detect malware by scanning for known patterns, the risk associated with the described DoS attack cannot be completely eliminated.

Observation 5 *A BankID server utilizing an encrypted PKCS #12 file to store the private key is potentially vulnerable to well-known password attacks to decrypt the file, and DoS attacks where malware deletes the file.*

4.6 Mitigating authentication risk

From Observation 1, the policy of using SSNs to identify customers and denying them access after a few wrong login trials must be changed because it enables efficient DDoS attacks on the application layer, potentially affecting more than two million customers in the near future. According to Observation 3, transactions should be authenticated for BankID to become more robust against combined phishing/MitM attacks, and from Observation 4, passwords and PINs should not be transmitted from a BankID client to the central infrastructure since this solution is vulnerable to combined DDoS/phishing attacks. Passwords and PINs should only be used locally by the end-user to give the client access to the end-user's PKI credentials. A new end-user authentication solely based on the end-user's public-private key pair will increase the strength of the authentication beyond what is possible with traditional two-factor authentication. From Observation 5, all web applications using BankID should employ HSMs to store private keys.

5 Risk of non-repudiation service

In the following we first discuss three vulnerabilities in the BankID architecture with the potential to limit the degree of achievable non-repudiation. We then consider how the strength of the end-user authentication influences the degree of non-repudiation. Finally, we discuss how to mitigate the non-repudiation risk.

5.1 No trusted third party

BankID does not employ a trusted third party to achieve non-repudiation despite the fact that this is required by most non-repudiation protocols described in the literature [7]. On the contrary, the BankID-member banks have strong financial relationships with both end-users and merchants owning web sites. In particular, when the web sites are Internet banking sites, the banks own the sites as well as the complete BankID infrastructure, giving the banks a large amount of control over financial operations requiring non-repudiation.

The following scenario illustrates the problem with the non-existent third party. Assume that a BankID customer and his bank have both digitally signed a document. At some later point in time there is a conflict between the bank and the customer during which the bank claims it didn't sign the document. It is then up to the customer to show that the bank did in fact sign. Since the bank controls the Internet banking application and BankID is controlled by the Norwegian banking community, the bank has access to a wealth of technical and judicial information, whereas the customer has only a copy of the digitally signed document on his computer. Furthermore,

the bank has readily access to BankID experts, while the customer has only access to security experts without any inside knowledge of BankID. Consequently, since no third party has collected non-repudiation information to assist the customer during the conflict, the customer and his lawyers will find it very difficult to convince a judge that the bank really signed the document when it denies having done so.

Observation 6 *The non-repudiation service in BankID gives a bank an advantage over its customers during conflicts involving repudiation of digital signatures because the customers cannot rely on help from a trusted third party.*

5.2 Insecure local key storage on BankID server

A high degree of non-repudiation requires that it is very difficult for an outside cracker (or rogue insider) to obtain a web site's private key. Unfortunately, in some cases the private key is stored in an encrypted PKCS #12 file. A cracker may be able to obtain the fixed password used to decrypt the file since the password is vulnerable to dictionary and social engineering attacks. Once the cracker has the private key in the decrypted file he can sign documents without the knowledge of the key's rightful owner.

Observation 7 *Only web sites utilizing HSMs to store and use private keys can support a high level of non-repudiation.*

5.3 Central key storage

An end-user's private key is stored on the central BankID infrastructure controlled by The Norwegian Banks' Payment and Clearing Centre. According to [8], this key is only used inside an HSM. Since private keys must *only* be available to end-users [4, pp. 52, 93, 156] to achieve a high degree of non-repudiation, the central key storage raises several questions. How are the private keys generated on behalf of the customers and placed in the HSM without anyone possibly learning the value of the keys? How can customers provide the fixed password to activate the authentication and signing functionality without anyone else obtaining the password? Is a network connection from a customer made directly to the HSM to avoid MitM attacks from rogue insiders?

During a dispute a bank must provide a judge with convincing answers to these questions. In particular, a bank must explain why a rogue insider isn't able to exploit the HSM's application programming interface, modify existing code, install new malicious code, and/or modify server configurations to get access to keys.

The banks have already been able to convince the Norwegian Post and Telecommunications Authority how only the end-users can grant the central

infrastructure access to private keys inside an HSM. However, the authority has refused to share its reasons for accepting central key storage. This is unfortunate since the lack of information makes it very difficult for an end-user (or his lawyer) to challenge a bank's claims about the non-repudiation during a dispute.

Not surprisingly, the Norwegian banking community has also refused to share any information with us about the system they use for non-repudiation. As far as we know, the non-repudiation protocols have not been analyzed by independent security experts or tested in Norwegian courts.

Observation 8 *The security-through-secrecy policy of the BankID-member banks gives them an advantage over their customers during conflicts because the customers and their lawyers have no access to technical information about the non-repudiation service.*

5.4 End-user authentication limits degree of non-repudiation

If the end-user authentication in a PKI is too weak then it is possible for a skilled cracker to steal a user's digital identity. The cracker can then digitally sign a document using the victim's identity. It can be difficult for the unfortunate user whose digital identity was misused, to show that he didn't sign the document. If the PKI offers a non-repudiation service, this will only increase the problem for the unfortunate user because the other signee has access to "credible evidence" showing that the user did sign when in fact it was the cracker who misused the user's identity. Hence, strong authentication of users is needed to achieve a high degree of non-repudiation.

According to Observation 2, the strength of the end-user authentication in BankID is limited by the strength of the two-factor authentication utilized by the BankID-member banks. Furthermore, Observations 3 and 4 point out that the two-factor authentication is vulnerable to well-known attacks. As a result, it is possible to steal users' identities.

Observation 9 *The end-user authentication in BankID limits the degree of non-repudiation. The strength of the authentication should be increased before customers digitally sign contracts concerning large-valued assets.*

5.5 Mitigating non-repudiation risk

From Observations 6 and 8, the non-repudiation service gives a bank an advantage over its customers because the customers cannot obtain technical information about the service or rely on help from a third party during a conflict. The Norwegian banks should release information about the technical and legal non-repudiation protocols. In particular, the banks need to publish their dispute resolution procedures. Only then will it be possible for the users to get an understanding of the true risk associated with the

non-repudiation service. It follows from Observation 7 that web site owners wanting to use the service must invest in HSMs to store cryptographic keys. From Observation 9, the strength of the authentication should be increased to improve the level of non-repudiation.

6 Privacy risk

The meaning of the term *privacy* depends on the context in which it is used. We define privacy as the right of an individual to decide when and how sensitive personal information should be revealed. To get an understanding of the privacy risk associated with BankID, we'll consider how the system may be used to build customer profiles.

Let us first consider the situation where an end-user wants to authenticate a web site. During the authentication process the BankID client depends on the central infrastructure to verify the web site's X.509 certificate. Because the end-user has to authenticate to the central infrastructure and the certificate contains the web address of the site, both the end-user and the web site are uniquely identified by the central infrastructure.

The CAs in the central infrastructure have access to the personal information provided by all individuals wanting to become BankID end-users. Once they start using BankID, it follows from the above observation that the central infrastructure can record e.g. where end-users shop and which government agencies they have dealings with.

Because the end-users must enter their SSNs during the authentication process, it is possible to link information from BankID with information in other commercial and governmental systems such as credit reporting agencies and taxation agencies. Hence, assuming BankID becomes the prevalent tool for online authentication of individuals in Norway, it will be possible to build increasingly *detailed profiles* over time revealing business and personal relationships of more than 2 million customers.

The US National Research Council has published a report [15] stating that individuals should know what information about them is stored in a national computer system, how this personal information is made available to third parties, how the information is updated, and most importantly, how they can prevent disclosure of the information. Neither of these requirements are fulfilled by BankID.

Observation 10 *The Norwegian banking community controls an ID system with the potential to build detailed profiles of roughly half of the Norwegian population as long as the BankID authentication utilizes X.509 certificates and SSNs. The BankID customers don't know how their personal information is utilized.*

We remark that the BankID client leaks information. It can be downloaded by anyone with a computer. If you enter the SSN of a BankID customer into the client, then the client will list the banks for which the customer uses BankID for authentication.

6.1 Mitigating privacy risk

The BankID system should be reviewed by independent privacy experts before it is allowed to become a de facto national ID system. Identified weaknesses in the privacy protection should be carefully examined and mitigated. In the long run a new authentication procedure, not using X.509 certificates and SSNs, should be introduced to minimize the system's negative effect on the end-users' privacy.

7 Conclusions

In April of 2007—after the risk analysis was completed—we were informed by a BankID representative that changes had been made to the system to mitigate the risk associated with MitM attacks (Observation 3). These changes may also increase the level of non-repudiation (Observation 9).

At the time of writing, late April 2007, the risk to BankID customers is still significant because (i) BankID is particularly vulnerable to DDoS attacks at the application layer, (ii) combined DDoS/phishing attacks may empty out customers' accounts, (iii), the lack of an independent third party in the non-repudiation service and the banks' security-through-secrecy policy gives them an advantage over their customers during conflicts involving repudiation of digital signatures, and (iv) the customers don't know how BankID utilizes their personal information. We recommend that steps are taken to mitigate (i)–(iv).

7.1 Who should own the remaining risk?

Even after a risk mitigation process is carried out there still is a residual risk associated with BankID. Hence, it is interesting to observe that if an outside cracker or rogue insider is able to empty out an account in a BankID-member bank, then the bank's responsibility is limited to one-hundred thousand Norwegian Kroner (about sixteen thousand US dollars) [9, p. 15].

If a bank is grossly negligent, then the limit doesn't apply. During a dispute it is up to the customer or his lawyers to establish that the bank has been grossly negligent. Experience shows that this is close to impossible since the Norwegian banks have long refused to share any technical information about their systems (see [2] for a further discussion of this point).

Since only the banks can strengthen the security of BankID, they should take the remaining risk and, thus, be liable for any loss caused by crackers

and rogue insiders. It is then up to the banks to determine the best balance between investing in better security and simply covering the loss caused by fraud.

7.2 General recommendations

We make four general recommendations to limit the risk customers take when using a national PKI such as BankID. First, traditional secrets, e.g. PINs and passwords, should not be transmitted from a client to the central PKI infrastructure. Hence, two-factor authentication should only be used locally to give the client access to the end-user's PKI credentials.

Second, if a PKI is found to be vulnerable to well-known attacks such as phishing and MitM attacks, then immediate steps should be taken to mitigate the risk to the end-users. This is of particular importance when the end-users don't have a convenient alternative to the services offered by the PKI.

Third, the non-repudiation service in a national PKI should not be sanctioned by any government before independent lawyers and security experts have evaluated the legal and technical protocols and determined the true level of non-repudiation. The results of such an evaluation should be made public.

Fourth, no citizen in any country should be forced to use a national ID system before an analysis of the system's privacy implications is made public and any discovered weaknesses in the privacy protection are corrected. Robust safeguards against profile building should be in place before any ID system is accepted by a nation's government.

References

- [1] K. J. Hole, V. Moen, and T. Tjøstheim, "Case Study: Online Banking Security," *IEEE Security & Privacy*, vol. 4, no. 2, 2006, pp. 14–20.
- [2] K. J. Hole, V. Moen, and A. N. Klingsheim, "Lessons from the Norwegian ATM system," submitted to *IEEE Security & Privacy*.
- [3] A. Jones and D. Ashenden, *Risk Management for Computer Security*, Elsevier, 2005.
- [4] C. Adams and S. Lloyd, *Understanding PKI*, 2nd Edition, Addison-Wesley, 2003.
- [5] W. Ford and M. S. Baum, *Secure Electronic Commerce*, 2nd Edition, Prentice Hall, 2001.
- [6] S. T. Kent and L. I. Millett, Editors, *Who Goes There?*, National Academies Press, 2003.

- [7] J. Zhou, *Non-repudiation in Electronic Commerce*, Artech House, 2001.
- [8] The Norwegian Banks' Payment and Clearing Centre (BBS), "BankID FOI White Paper," Release 2.0.0, 2006 (in Norwegian).
- [9] Bankenes Standardiseringskontor, "Norsk BankID sertifikatpolicy for banklagrede kvalifiserte sertifikater til personkunder," Version 1.1, 2005 (in Norwegian).
- [10] RSA Laboratories, "PKCS 12 v1.0: Personal Information Exchange Syntax," 1999.
- [11] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service*, Prentice Hall, 2005.
- [12] D. Geer, R. Bace, P. Gutmann, P. Metzger, C. P. Pfleeger, J. S. Quarterman, and B. Schneier, "Cyberinsecurity: the cost of monopoly," Sep. 2003;
www.ccianet.org/filings/cybersecurity/cyberinsecurity.pdf.
- [13] S. C. McQuade, *Understanding and Managing Cybercrime*. Pearson, 2006.
- [14] R. E. Smith, *Authentication*. Addison-Wesley, 2002.
- [15] S. T. Kent and L. I. Millet, Editors, *IDs—Not That Easy: Questions About Nationwide Identity Systems*, National Academies Press, 2002.

Paper II:
Open Wireless University Networks

Open Wireless University Networks

Kjell J. Hole, Lars-Helge Netland, Yngve Espelid, André N. Klingsheim, Halvar Helleseeth, Jan B. Henriksen

Department of Informatics

University of Bergen

PB. 7800

N-5020 Bergen

NORWAY

Email: {Kjell.Hole,larshn,yngvee,andre,k,hallvar,Jan.Berger.Henriksen}@ii.uib.no

July 1, 2007

Version 2.1

Abstract

The authors first argue why universities should employ wireless networks without authentication to the network infrastructure. They then analyze and mitigate the risk associated with such open networks.

Introduction

Information systems at universities are collections of computers, storage equipment, and communication networks used to transport, process, and store information. Wireless communication networks are of particular interest to universities because students using mobile terminals (e.g. laptops and PDAs) with wireless communication capabilities can download lecture slides, watch educational programs, as well as video and audio presentations, and take online practice quizzes at any time and from anywhere on campus. During university courses, the number of paper handouts can be significantly reduced and paperless assignments and submissions are simplified. Wireless networks furthermore strengthen teamwork among students because they can more easily email each other preliminary results, use chat channels to discuss problems with other team members, and use information resources more actively during problem solving sessions.¹ Faculty similarly benefits from wireless access to online services. It may even be possible for universities to retire expensive computer labs when students and faculty can access all the information they need via their own mobile terminals.

The usability of a university's information system is mainly determined by how simple it is for users to achieve their goals when they utilize the system. Usually, a university employs individual authentication, often shortened to authentication, to determine who can access their information system. Authentication mechanisms tend to reduce the usability of the system because they're seen as intrusions keeping the users away from their primary tasks, and because they require users to remember a secret and/or possess a certain authentication device.²

There is also a trade-off between a user's right to privacy and individual authentication. Users authenticating to a wired network infrastructure have long made it possible for an IT department at a university to collect information about what users do and when they come and go. The introduction of a campus-wide wireless network utilizing authentication makes it simpler to track users' movements and activities on campus.

A university's information system is said to be *open* if it grants users access to the network infrastructure without any form of authentication. In this case, users can freely surf the Internet, access library catalogs and other services providing non-sensitive information. Sensitive services such as email access still require authentication. An open information system can provide both wired and wireless access to unauthenticated services. The degree to which an information system is open depends to a large extent on the number of different devices able to access the system, the number of services not requiring authentication, and the availability of the network.

This paper first introduces a simple model of a university information system where mobile terminals all over campus communicate over wireless links with services on a wired infrastructure. It is then argued that the usability of the system increases when individual authentication is removed from the network infrastructure. Considering the resulting open system, it is determined how to mitigate risk associated with possible illegal downloads, different types of attacks, negative press coverage, and lack of privacy. Finally, it is discussed how an IT department can vary the degree to which a network is open.

During the risk analysis the authors assume that all users access the services on the wired infrastructure via their mobile terminals. Hence, we often refer to the modeled information system as a wireless network even though it contains a large wired infrastructure. While it's common to group attackers into rogue insiders and outside crackers, we don't make this distinction since it makes less sense on an open network.

Authentication in an information system

This section first defines the term 'individual authentication' and explains how authentication differs from 'authorization' and 'device identification'. It then models two versions of an information system on a university

campus, where the first version employs individual authentication at the network boundary, and the second version is an open system where users only need to authenticate when they want to access services with sensitive information.

Authentication defined

Individual authentication is the process of establishing an understood level of confidence that an identifier, e.g. a name, refers to a particular individual. The authentication occurs in two phases: (1) an identification phase, during which an identifier is selected, and (2) an authentication phase, during which the required level of confidence is established.³ The authentication is said to be strong if the resulting level of confidence is high. To keep the discussion general, we don't specify any particular techniques for individual authentication. Readers interested in learning more about these techniques should consult the literature.⁴

It will be important later on to understand that device identification, i.e. the process of determining to what identifier a particular device corresponds, doesn't necessarily lead to strong individual authentication. A user's mobile terminal can be identified by its unique Media Access Control (MAC) address. The MAC address can again be associated with a specific individual. Some IT departments can more easily make this connection because they register the MAC addresses of all new terminals given to students and faculty. However, the resulting individual authentication is weak since it's easy to fake, or spoof, a MAC address and since the owner of the terminal can claim that some other individual used his terminal during a particular time period.

Authentication and authorization are distinct, but related concepts. Authentication establishes what an individual "is", while authorization determines what an individual "is allowed" to do. An authorization policy determines how authorization decisions are made. We simply assume that an authorization process is carried out and that the authorization policy requires individual authentication of all potential users.

Network boundary authentication

Figure 1 depicts an example of a university information system where mobile terminals communicate with wireless access points to gain access to servers on a wired backbone. The users are authenticated at the network perimeter, prior to accessing any of the services offered. The solid black line indicates the authentication boundary. The resources inside this boundary are only made available to users that have authenticated themselves correctly.

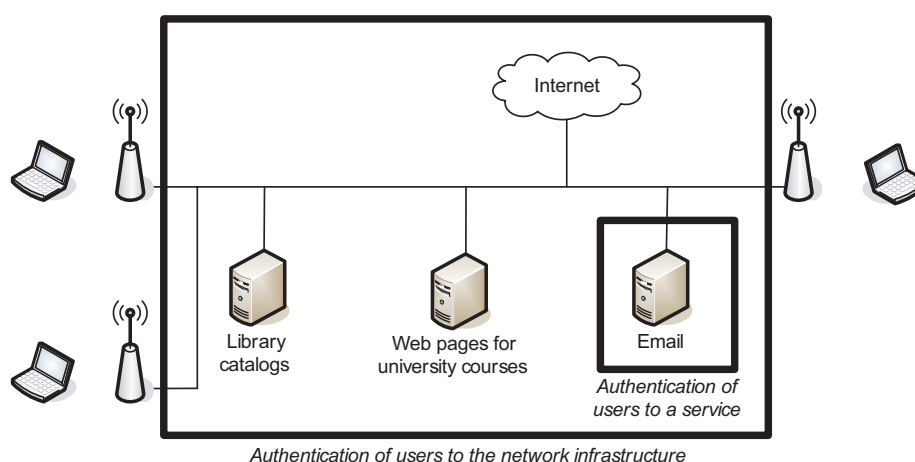


Figure 1: University information system with authentication at the network boundary

The model in Figure 1 depicts the information system from the mobile users' point of view. The thin black lines indicate possible information flows between entities, but they are not meant to indicate how the information system should be implemented. Observe that access to services is simpler from the Internet than from the wireless network since the network boundary authentication only applies to wireless users. Also note that the sensitive email service requires a service level authentication of all users.

Service level authentication

Figure 2 is based on the same information system as in Figure 1. However, the authentication is removed from the network perimeter to create an open network, where users get unrestricted access to library catalogs, web pages with course information, and the Internet. These services, as well as other non-sensitive services, are available to anyone with equipment that can communicate with the wireless access points. Users must still authenticate to sensitive services. An example is the email service in Figure 2.

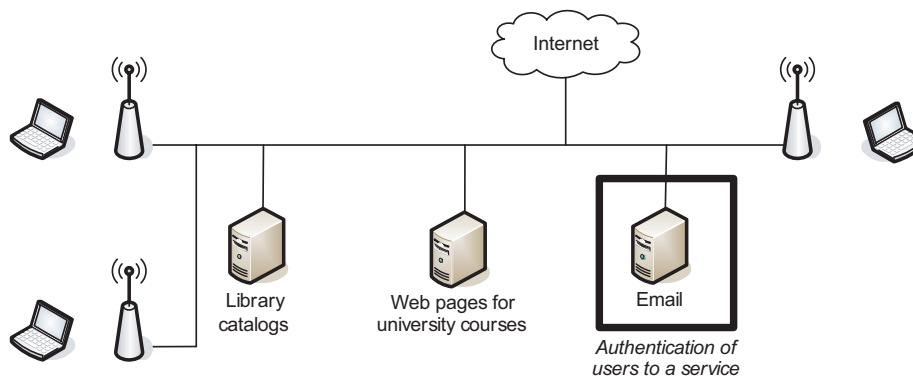


Figure 2: University information system with service level authentication

Because users now authenticate to individual services it's possible to adjust the strength of the authentication to fit the sensitivity of a particular service. A service containing very sensitive information requires strong authentication e.g. based on a hardware token and/or a PKI (Public Key Infrastructure), while a service with less sensitive information can do with password-based authentication.

Defense in depth

Even though the individual authentication has been removed from the network perimeter, the open information system in Figure 2 still supports a defense-in-depth strategy. The perimeter can contain security mechanisms such as firewalls and intrusion detection systems. The inside defense consists of security mechanisms on the client and server machines where the information resides.⁵ It may also include monitoring techniques⁶ to verify that all entities, including the users, behave according to established security policies.

Increased usability

This section considers the usability of a wireless campus network for students, faculty, and formal visitors (the insiders), as well as for invited short-time guests and the general public (the outsiders).

Increased usability for insiders

When a campus network doesn't authenticate students, faculty, and formal visitors at the network perimeter, the usability increases because they don't have to remember passwords or carry authentication tokens to access non-sensitive services. An open network can therefore lead to increased use of information resources and, thus, a better learning environment and increased knowledge among insiders.

To achieve a high degree of usability in a campus network with network boundary authentication, it is necessary to provide a well-functioning authentication mechanism for all insiders' wireless devices, e.g. laptops, PDAs, and mobile

phones. This task can be daunting because popular wireless devices run multiple versions of different OS platforms (Windows, Mac OS X, Linux, Symbian OS, Palm OS, etc.), all with their own unique challenges. An open campus network reduces the problem because insiders can access most services without any form of authentication. It may, however, still be a challenge to give insiders wireless access to sensitive services requiring authentication.

Increased usability for outsiders

Universities are important democratic institutions with a responsibility to make essential information easily available to the general public. Open wireless networks can provide citizens with effortless access to university library catalogs and other important information. The added complexity introduced by an authentication scheme should be avoided since people in general have rather limited understanding of security mechanisms and find it hard to authenticate themselves to the network. Strong authentication of individuals also requires some form of initial generation of information needed to carry out the authentication process, further lowering the usability of the network and adding to the expense of operating the network.

Employees at a university we've studied must request temporary accounts from the IT department to give short-term guests access to the wireless network. Unfortunately, visitors and participants in more loosely organized activities get no wireless access because they don't know an employee they can contact to establish the needed accounts in advance. Most first-time visitors don't even know they need an account to get wireless access. A much more user-friendly alternative is to have an open wireless network without any need for short-term accounts.

The digital divide refers to the gap between people that have regular access to digital technologies and those without. It's widely recognized that the digital divide must be reduced, especially between people in developing and industrialized nations. Universities in developing countries can offer outsiders with little computer training easy Internet access via low-cost laptops and open wireless networks. Educational institutions have the knowledge to remove authentication mechanisms, thereby increasing usability, and still protect sensitive resources.

Risk analysis and mitigation

Let a *vulnerability* be a flaw in the design or a bug in the implementation of an information system, and let a *threat* be an adversary with the capabilities and intentions to exploit a vulnerability. The *risk* associated with the system is a function of the exploitable vulnerabilities, the seriousness of the threats, and the values of the information assets.⁷ We'll analyze the risk associated with the open information system modeled in Figure 2. The following risk analysis only considers wireless access from the mobile terminals and determines whether the removal of the authentication from the network perimeter increases the risk. It's also discussed how to mitigate the risk.

Illegal downloads

Students don't utilize a wireless network only for their studies. Illegal downloading of music and movies goes on in student dorms. Some university employees also engage in illegal downloading. When the individual authentication is removed from the network perimeter, the temptation to use the network for illegal downloads may increase. A university deploying an open wireless network must therefore present users with rules governing network usage. In particular, the university must reserve the right to prosecute any user who causes economic loss or damage to the university's reputation through illegal downloads over the open wireless network.

The promise of legal action against perpetrators will limit the amount of misuse, but it will not eliminate it. However, this is not very different from the situation we already have with networks on most campuses today. The IT department responsible for an open wireless network can monitor and log the traffic on the network to stop unwanted download activity. It's also possible to "filter out" many illegal downloads by only allowing traffic on certain network ports (more on this later).

An IT department worrying that pedophiles will download child pornography over the university's wireless network can introduce a special child pornography filter. The Norwegian police and Norwegian ISPs (Internet Service Providers) together maintain a child pornography filter, which warns the ISPs' customers about accessing sites

containing child pornography. The ISPs update the filter on a regular basis using lists of domain names provided by the police. In May of 2007 the filter contained 4,235 domain names. The police in other countries like Sweden and Denmark also help maintain filters against child pornography.

Terminal-to-terminal attacks

When students and faculty use mobile terminals, it's possible for a cracker to attack any terminal over a direct wireless link from his own terminal, or via a wireless access point without going through the wired infrastructure. On an open wireless network, it's also feasible for the cracker to attack from the wired infrastructure. He could for example install an unauthorized access point, often called a rogue access point, running malicious software. However, mobile terminals today are already attacked from the many wired networks they're connected to as the users move around. A terminal not running a personal firewall and an up-to-date antivirus program has, most likely, been compromised many times even if it has never been connected to a wireless network. Hence, most terminals should already have the necessary protective software needed to access an open wireless network.

We remark that rogue access points constitute a serious security problem in wireless campus networks using authentication at the network boundary because the access points can give crackers unauthenticated access to information assets. While this problem is reduced in open networks, it's still necessary for the IT staff at a university to search for and remove rogue access points because they can be exploited by crackers.

Attacks on local networks

When the authentication is removed from the network perimeter in Figure 1, the unauthenticated wireless network access makes it simpler to attack services on the wired infrastructure using masquerade attacks, or spoofing attacks, where one entity illegitimately poses as another entity to gain access to restricted information. Introducing rules and threatening with legal prosecution do not significantly reduce the risk associated with masquerade attacks. Instead, security techniques must be used to mitigate the risk. In the following, we consider attacks on both sensitive and non-sensitive services.

When users don't have to authenticate to the network infrastructure to use a non-sensitive service, a cracker may be able to introduce false services providing e.g. fake lecture notes and bogus research papers. Risk mitigation can be obtained by installing anti-virus software and firewalls on the servers, and by running auditing programs to record the activities of all users. Regular reviews of the audit logs help detect illegal activity and aid in identifying attackers. The IT department should be prepared to quickly reinstall and secure a web server when an attacker has managed to breach its defenses and perhaps modified the content of some web pages. Universities already have services available on the Internet and should therefore have experience and competence on securing systems, as well as dealing with security breaches.

Strong authentication is needed to ensure that only legitimate users get access to a sensitive service. Since it is easy for anyone to 'sniff' passwords on unencrypted wireless links, password-based authentication also requires the use of end-to-end encryption between the mobile terminals and the server. The risk related to masquerade attacks on sensitive services is mitigated by requiring users to access the services via encrypted VPN (Virtual Private Network), SSH (Secure Shell), or SSL (Secure Sockets Layer). In addition, the steps suggested to mitigate the risk associated with the non-sensitive services also alleviate the risk linked to sensitive services. Very sensitive services, e.g. services processing sensitive medical information, should not be on a network connected to the Internet.

A campus network we've evaluated employed a VPN solution, which didn't encrypt the network traffic. As a result, it was easy to 'sniff' usernames and passwords on the wireless links. Once a cracker had an employee's username and password he could download a central password file to a local machine. The file contained many thousand password hashes. Using a password cracker, it was possible to obtain several hundred usernames and passwords. One of the passwords vulnerable to cracking belonged to one of the network engineers at the IT department, enabling a cracker to escalate privileges. This example illustrates the need for a robust design employing both strong authentication and strong encryption to protect sensitive information.

Anonymous attacks on remote networks

Removing the authentication from the network perimeter of a university network can make it possible for a cracker to carry out anonymous attacks from the open network on information assets anywhere on the Internet. These anonymous attacks include music and software piracy, identity theft, denial-of-service attacks, spam and phishing, and attacks on remote machines.

When analyzing the risk of remote criminal attacks it is important to realize that an open wireless network at a university doesn't represent a major new possibility for a cracker to gain anonymous Internet access. A previous paper⁸ documented the existence of a large number of open Wi-Fi (Wireless Fidelity) networks in the city of Bergen, Norway. In this particular case, a Wi-Fi network is said to be 'open' if the WEP (Wired Equivalent Privacy) and WPA (Wi-Fi Protected Access) security protocols aren't used. While the percentage of open networks in Bergen seems to have gone down since the investigation in 2004, a survey in 2006 indicates that roughly 40% of the networks are still open. Furthermore, the total number of wireless networks has continued to increase since 2004, giving crackers a large number of open networks to choose from. Similar observations have been made many other places around the world.

In addition to the many open Wi-Fi networks already existing, there are other ways of obtaining anonymous Internet access not requiring an open university network. Perhaps the most famous example is Tor, an anonymous Internet communication system (tor.eff.org). Tor consists of a network of computers, or proxies, used to reroute users' Internet traffic. Multiple layers of encryption inside the Tor network protect the traffic from eavesdropping. Tor was primarily created to anonymize web browsing and publishing, instant messaging, and other applications that use the TCP protocol. The system allows individuals and organizations to share information over the Internet without compromising their anonymity. While Tor gives individuals anonymous access to e.g. chat rooms and web forums for rape and abuse survivors, and lets journalists communicate safely with whistle-blowers and dissidents, Tor also enables crackers to carry out anonymous attacks.

It must be pointed out that a particularly good way for a cracker to conceal his identity is to access Tor via an open wireless network using a mobile terminal with a spoofed MAC address. In fact, this technique allows a cracker to conceal his identity to the extent where:

- The owner of the wireless network cannot determine what remote system a cracker is accessing.
- The wireless network owner cannot determine the content of communication between the cracker and the remote system.
- The owner of the remote system cannot determine the originating network of the cracker.
- If the cracker reveals his identity to someone on the Internet, then that someone can still not determine the cracker's home network
- It is hard for the wireless network owner to determine which mobile terminal the cracker is using on a campus with many active terminals.

The anonymity isn't completely fail-safe because the application layer may leak information. As an example, if the same cookie is used during two web-browsing sessions, where one session is via Tor and the other isn't, then the web server can match the two sessions and determine a valid Internet address for the cracker. Advanced browser plug-ins and scripts can also access private information on the cracker's computer and send it to the web server. However, a cracker aware of these and other similar issues can disable troublesome software to protect his anonymity.

The risk associated with anonymous Internet access from an open university network can be reduced, but not eliminated, by the introduction of network monitoring. The remaining risk is, however, limited because a cracker is

more likely to choose an open Wi-Fi network owned by a private party or small business rather than attempt to exploit an open university network monitored by a large IT department.

The main threat isn't a cracker using spoofed MAC addresses and Tor to mount an anonymous attack via the open wireless network. What a cracker really wants is to use the identity of a legitimate user. Alas, the main vulnerability is weak (password-based) authentication used to control access to sensitive services. In one instance we saw how it was possible to steal a large number of online identities. Once a cracker takes on the identity of e.g. an employee at the IT department it may not be difficult for the cracker to hide all his activities.

Press coverage

A university may worry that an open wireless network will lead to negative press coverage because an incident occurred on the network; thus, the university chooses to employ mandatory authentication of all users to make incidents less likely. While it may be true that an open network can lead to an increase in illegal downloads and different forms of local and remote attacks, we've earlier discussed how security techniques can mitigate the risk. It is also possible for a university to cooperate with the press to reduce the negative impact of an incident. The IT department should explain what happened and outline steps it'll take to better secure the system in the future. A university that actively values and builds trust⁹ in this way is far less vulnerable to loss of reputation caused by a network incident than a university trying to hide bad news.

We recognize that full disclosure of security breaches comes with a short-term financial penalty for commercial vendors. The attacked companies lose on average 2.1 percent of their market value within two days of the announcement.¹⁰ However, disclosing security problems to the general public doesn't translate to poor security. To claim that a system is infallible reveals a limited understanding of security. A far superior strategy from a security standpoint is to realize that intruders may very well find a way into business systems, and therefore divert resources into procedures for dealing swiftly with attacks to come. Currently, the marketplace doesn't value full disclosure as a better security policy. Universities are key players in bringing about a change, and should therefore disclose security breaches and value open processes when it comes to security.

Privacy concerns

The meaning of the term *privacy* depends on the context in which it's used. For our purposes, it's useful to define privacy as the right of an individual to decide when and how sensitive personal information should be revealed. An information system at a university contains large amounts of personal information such as medical information, social security numbers, annual salaries, student grades, and information about disciplinary actions. Traditionally, most students and employees haven't worried about privacy issues despite having little or no knowledge about how their personal information is collected, processed, and stored. Recent press reports describing large information thefts have, however, raised questions about the level of privacy afforded to students and employees at universities. Clearly, an open network should not make it easier for crackers to steal personal information.

Users authenticating to a wired network infrastructure have long made it possible for an IT department at a university to collect information about what users read, the services they use, and when they come and go. The introduction of a campus-wide wireless network employing individual authentication makes it even easier to track users' movements on campus. This increased possibility of information gathering raises new privacy concerns, especially since information from the wireless network can be combined with information collected from the wired infrastructure. Because the users' identities are known, it's possible to combine data from many network sessions and build accurate user profiles including the users' preferred whereabouts.

The risk that a university IT department tracks users' locations and collects information to build user profiles can be alleviated by not authenticating individuals on the wireless network infrastructure. The IT department must also choose a network monitoring technique which do not reduce the privacy afforded to students, employees, and guests to an unacceptable low level. An IT department has a natural tendency to prioritize network monitoring over individual privacy because it must stop misuse. To curb this tendency, the department should be open about the

techniques it uses for network monitoring; especially since openness builds trust, which again reduces users' privacy concerns.

There is also a risk that a cracker invades another mobile user's privacy by sniffing the wireless network traffic or by accessing information stored on the user's device. As mentioned before, all mobile devices must incorporate security mechanisms to protect local data and the transmission of sensitive information must be protected by end-to-end encryption. When a user authenticates to the network infrastructure, he or she is often forced to use encryption. In the case of an open university network, this decision is left to the user. Hence, to mitigate the risk, it is necessary to educate users about the fact that they are responsible for their own security and educate them on how to take the necessary security precautions.

Legal issues

The EU parliament has adopted Directive 2006/24/EC to track EU citizens' Internet communications. With regard to Internet access, Internet email, and Internet telephony, the EU member states must retain data to: (i) trace and identify both the source and destination of a communication, (ii) identify the date, time, and duration of a communication, (iii) type of communication, (iv) identify the communication device, and (v) identify the location of mobile terminals. The directive specifies the data to be retained in more detail. Collected data must be stored for periods of not less than six months and not more than two years from the date of the communication. No data revealing the content of the communication may be retained pursuant to the directive.

There is uncertainty about the directive's full impact at the time of writing. Privacy advocates claim the directive makes it illegal for any entity, including a university, to give users access to the Internet without satisfying the requirements (i)–(v). While each member state can postpone application of the directive to the retention of communications data relating to Internet access, Internet telephony, and Internet email until 15 March 2009, universities in the EU may be unable to operate campus networks without mandatory authentication of all users after this date.

In the US, the Federal Communications Commission wants universities to obey the Communications Assistance for Law Enforcement Act (CALEA) originally written to make telephone companies open up their digital lines to the law enforcement agencies. At the time of writing, it is not completely clear whether CALEA applies to universities. However, if this is found to be the case, then universities in the US must introduce technology in campus networks to facilitate wiretaps during investigations. Universities may also find they need to introduce mandatory authentication of all network users to further facilitate the wiretapping and to avoid any future legal actions from the US government.

The risk associated with the future legal status of open wireless networks in the US and EU cannot be eliminated at this point in time. Universities planning to introduce open wireless networks should consult with lawyers to assess the legal risk.

Negative responsibility

Negative responsibility is the responsibility for what one doesn't do but could've done. In the event that an information asset gets "hacked", the head of a university IT department may worry he'll get blamed for not requiring all users to authenticate to the network infrastructure. It is therefore necessary to back up a proposal for a new open wireless network with a solid analysis of the security properties of the complete system. This is a large task if previous security analyses aren't well documented, and it may be more convenient for some to maintain the "default belief" that threats from crackers and pedophiles make it too risky to introduce an open network. However, the analysis in this paper shows that a university is ill served by buying into the "default belief" without carrying out their own security analysis.

Network openness

In the following, we'll discuss how the usage of a captive portal, the introduction of port filtering, and the level of network accessibility influence the degree to which a wireless network is open. An IT department at a university must carefully consider these aspects during the design and implementation of an open wireless network.

Captive portals

Web browsers on mobile terminals can display the rules governing the use of an open network. It is advantageous for a university to have all terminal users confirm that they accept the rules before they get full network access. A "catch and release" captive portal may be utilized to achieve this goal. The selected portal must be able to run on different platforms and give users uninterrupted network access over long periods. A portal requiring pop-up windows in the browser should be avoided because many users find these windows annoying and many terminals do not support them.

During the next few years more users will access wireless networks via a smart phone (combination of a PDA and a mobile phone) with Wi-Fi communication capabilities. These advanced phones have software such as email clients, media players, and web browsers. An IT department introducing a new open network should therefore make sure the selected portal solution is accessible from smart-phone browsers.

While a captive portal ensures that all users have the possibility to read the network policy, it also reduces the number of different devices being able to access the network because not all devices run browsers. An example of a device without a browser is the Vocera Communication Badge, a Wi-Fi device without a screen, which enables instant two-way voice communication over a wireless network. Hence, the reduction in openness caused by captive portals may be unacceptable in some open university networks.

Port filtering

It's not a major problem that members of the general population living close to a university campus utilize an open university network to a limited extent. However, it becomes a serious problem if these individuals start to download mass quantities of data over the open network and, thus, deny students and faculty the network performance they need.

The use of file sharing applications over a wireless network can undoubtedly cause network capacity problems. Closing the network ports most often used by file sharing applications can reduce the extent of the problem. If the wireless network has few open ports, then it'll be less attractive for people to try to download large data sets. (It's not possible to stop all downloads because file sharing can also be carried out over e.g. port 80.) The open network at the Department of Informatics, University of Bergen only allows SSH (port 22), HTTP (port 80), and HTTPS (port 443) traffic. A much higher degree of network openness is of course achieved by not doing any port filtering at all.

Accessibility

The service area of an open network is the total indoor and outdoor area from which a mobile terminal can communicate with at least one of the network's access points. The service area should cover all buildings where university employees and students work, including faculty and administration offices, classrooms, dorms, and libraries. If the service area also covers large areas outside these buildings, the open network is likely to interfere with other wireless networks run by private individuals and businesses. Hence, the IT department must work together with owners of neighboring wireless networks to avoid interference problems. The resulting size of the service area, in particular the portion of the area available to the general public, influences the degree of openness. If the size is too small, then the degree of openness is inadequate because it is not possible for enough people to access the network.

Conclusion

It is possible to mitigate the risk associated with misuse of a university's open wireless network. Misuse by regular users and attacks from crackers aren't likely to increase significantly if the IT department monitors and filters the traffic on the wireless network. This conclusion is supported by the experience we have with the open wireless network at the Department of Informatics, University of Bergen. This network has run for several years without major incidents. Telling the truth, and thus building trust can reduce the impact of negative press reports. In other words, an open wireless network need not represent an unacceptable risk to a university. However, since the future legal status of open wireless networks is unclear in the EU and US, it may become necessary to introduce mandatory authentication of all users sometime in the future.

To make a final decision on whether to deploy an open wireless network at a university, the associated risk must be weighted against the advantages. An open wireless network can improve the education of students and make important information more easily available to both the faculty and the general public. If the network is designed and implemented correctly, the risk associated with crackers is acceptable according to our analysis. The introduction of an open network may even help to better focus the security initiatives where they can do most good.

Still, the decision whether to deploy an open network remains somewhat of an ethical dilemma because an open network can give a cracker (using Tor and a mobile terminal with a spoofed MAC address) anonymous access to the Internet. There is no absolutely correct answer to whether or not to employ an open university network. The legitimate privacy requirements of guests, students, and university employees discussed earlier make the authors favor an open network over a network with authentication at the network boundary. Monitoring of the network traffic is an acceptable alternative if the IT department informs the users about the activity.

Since we only have practical experience with wireless networks in a university setting, we've focused on open university networks in this paper. Nevertheless, we believe much of our analysis is highly relevant to open networks in general. More work is needed to determine the best way to implement open wireless networks.

Acknowledgment

The authors would like to thank the reviewers for their thorough reviews which significantly improved the paper.

References

- ¹ R. B. Kvavik and J. B. Caruso, "ECAR Study of Students and Information Technology, 2005: Convenience, Connection, Control, and Learning," Research Study from the EDUCAUSE Center for Applied Research, vol. 6, 2005; www.educause.edu/ers0506.
- ² L. F. Cranor and S. Garfinkel, Editors, *Security and Usability*, O'Reilly, 2005.
- ³ S. T. Kent and L. I. Millett, Editors, *Who Goes There?* National Academies Press, 2003.
- ⁴ R. E. Smith, *Authentication*, Addison-Wesley, 2001.
- ⁵ P. Kuper, "A Warning to Industry—Fix It or Lose It," *IEEE Security & Privacy*, vol. 4, no. 2, 2006, pp. 56–60.
- ⁶ R. Bejtlich, *The Tao of Network Security Monitoring*, Addison-Wesley, 2005.
- ⁷ A. Jones and D. Ashenden, *Risk Management for Computer Security*, Elsevier, 2005.
- ⁸ K. J. Hole, E. Dyrnes, and P. Thorsheim, "Securing Wi-Fi Networks," *IEEE Computer*, vol. 38, no. 7, 2005, pp. 28–34.
- ⁹ S. Bibb and J. Kourdi, *Trust Matters*, Palgrave Macmillan, 2004.

- ¹⁰ H. Cavusoglu, B. Mishra, and S. Raghunathan, "The Effect of Internet Security Breach Announcements on Market Value: Capital Market Reactions for Breached Firms and Internet Security Developers," *International Journal of Electronic Commerce*, vol. 9, no. 1, 2004, pp. 69–104.

Paper III:
Security Pattern for Input Validation

Security Pattern for Input Validation

Lars-Helge Netland, Yngve Espelid, Khalid Azim Mughal
Department of Informatics, University of Bergen
{larshn, yngvee, khalid}@ii.uib.no

Abstract

This paper discusses a security pattern for input validation in web applications. A template description facilitates understanding of the important concepts, and allows developers with a security background to quickly adapt the pattern in their own context.

1 Introduction

E-commerce has become a major factor in the marketplace. In Norway, the percentage of total turnover from E-commerce grew from 2.2% in 2002 to 3.9% in 2005 [1]. This rapid growth has fueled a need for new software for selling goods and services over the Internet. In creating such systems, security aspects have been set aside in favor of time-to-market concerns and new application functionality. As a consequence, massive quantities of sensitive information lie inadequately protected on the Web.

Enter the villain. The Internet is a candy store for computer literate criminals. Snacks are readily available in form of credit card numbers, classified corporate data, and sensitive medical information. A long-time favorite target among hackers is the web-based shopping cart. Many developers use hidden fields to store and camouflage data from users, and fail to realize that the information can be easily manipulated. Through such tampering attacks, attackers can dictate their own prices in the web store, and hence purchase goods at extreme discounts. This vulnerability was reported in public no later than February 2000 [2], but software developers continue to release applications that are wide open to such attacks [3].

Software security is not a new research area. Saltzer and Schroeder's pioneering work "The protection of information in computer systems" [4], describes basic principles that remain true even today. More recently, much effort has gone into describing how attackers break software. In terms of building secure software, best practice guidelines have dominated the security literature. Patterns have become a popular way of sharing structured knowledge and experience from successful projects in traditional software engineering. A similar approach looks promising for distributing software security knowledge. The first paper in this direction was written by Yoder and Barcalow [5] in 1997.

Input handling is the process of checking data supplied by others against a set of predefined rules. The Open Web Application Security Project (OWASP) [6] has defined 3 categories of input handling:

Integrity checks, ensure that data has not been tampered with.

Validation, a set of rules that make sure that data is of a certain type, has correct syntax, is within specified length boundaries, or contains only permitted characters.

Business rules, checks that enforce back-end business logic, such as disallowing due dates already passed in an online banking application when paying bills.

The Input Validator pattern described in this paper addresses *validation* and *business rules*. It should be noted that specialized attacks such as SQL injection and cross-site scripting can be dealt with more effectively through prepared statements and HTML encoding, respectively [7]. It is possible to design a filter to look for metacharacter attacks, but identifying valid input (positive validation) is considered better than looking for data that can do you harm. Choosing the latter strategy is more costly in terms of maintenance, where you have to be constantly up-to-date with new threats, while the former option can provide protection against some of the attacks to come.

Unvalidated input is number one on OWASP's top ten list over web application vulnerabilities [8]. The rest of the paper presents a security pattern for input validation. We follow the template used in [9, p. 9], which is the de facto standard for writing security patterns.

2 Input Validator

A number of attacks on web applications target the application protocol through which vendors conduct business. Firewall technology and a PROTECTION REVERSE PROXY [9, p. 457] can enforce access control rules, but usually do not stop exploits embedded in message content. The back-end server(s) needs an INPUT VALIDATOR to validate data sent from the client.

Example

An Internet banking provider has recently been plagued by a series of hacker attacks. It started a few weeks back when about a dozen of the bank's clients filed reports demanding to know why their accounts had been cleared out. In the investigation that followed another problem was discovered: a fair quantity of bills paid by customers contained negative amounts. Fortunately, the bank invested heavily in a highly praised logging system a few months back, so sorting out the details should not be a problem. Section 2 describes how the bank improved their system's security.

Context

The input validator pattern applies to services offered through a client-server model communicating over HTTP. The goal is to validate input from the client. Input validation does not make network security solutions obsolete. It is a supplementary defense mechanism that helps to secure the environment in which the system runs. Any so-called client-side validation checks are worthless from a security standpoint. They serve the following three purposes: a) Enhance user experience through instant feedback on non-conforming input, b) conserve network bandwidth by reducing requests to the server, and c) save processing resources on the server. The term client-side validation is misleading, as none of the benefits mentioned above are related to security.

Problem

Many software systems fail to inspect and filter untrustworthy input. Through cleverly crafted input hidden in message content, attackers can alter the program logic and possibly exploit implementation weaknesses for economic gain. The pertinent question to ask is “How can one defend client-server applications against attacks hidden in the message content?”

Forces

The input validation solution must resolve the following forces:

- Extensive examination of input means more computational overhead. Developers should strive to make input validation as non-intrusive as possible. This is especially so in data-intensive systems, where delays that are acceptable in smaller applications, get compounded and render the system useless.
- The constant tug of war between software developers and malicious hackers has repeatedly shown how a dedicated attacker community discovers new ways of exploiting software. Systems should be designed with this ongoing battle in mind and incorporate flexible solutions that can be easily extended, if necessary, to counter future threats.
- Usability is a key factor for security constructs to prevail. Systems that are complex and difficult to operate cause maintainers and users to do mistakes that jeopardize security.

Solution

Perform syntactical and semantical validation of all input that is to be processed on the server. These 2 categories of validation are termed “validation” and “business rules” by OWASP. This process entails constructing appropriate validation rules for each source of input.

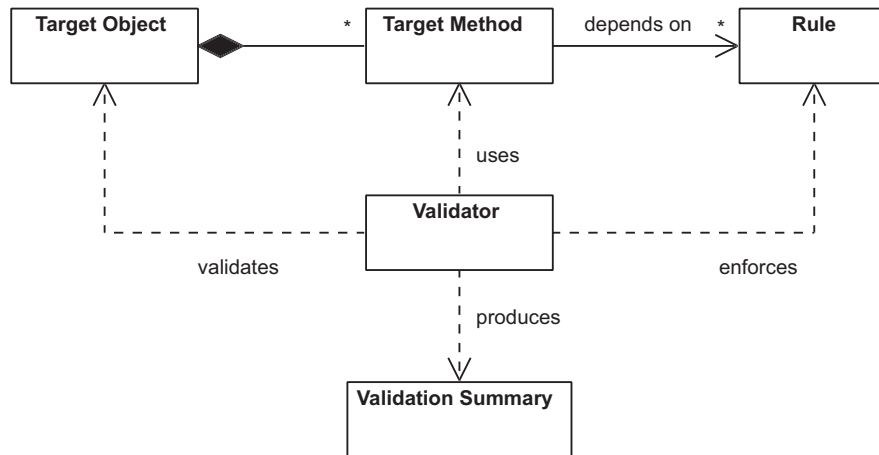


Figure 1: Content validation entities

Structure

Fig. 1 shows the class diagram for this pattern. The primary entities are

Target Object, which is the target for the validation process, comprising properties subject to inspection.

Target Method, which exposes an object property.

Rule, which defines constraints for the exposed object properties.

Validator, which controls the validation process, validating the exposed properties of an object against sets of rules and reporting any rule violations.

Validation Summary, which provides a summary of all rule violations.

Dynamics

The workings of an INPUT VALIDATOR can be explained in terms of an airport security checkpoint. Fig. 2 shows a typical setup, where travelers must pass a metal detector frame, and carry-on items are inspected in an X-ray machine. In this context, the primary entities can be interpreted as follows: The **Validator** is the airport security personnel and equipment; **Target Objects** are represented by airline passengers; **Target Methods** correspond to individual inspection targets, such as briefcases, cameras, or laptops; **Rules** express which items should be disallowed, typical examples include knives, guns, and can openers; **Validation Summaries** would be incident reports produced by the airport security personnel in the event of rule violations.

Consider the sequence of steps executed when a passenger shows up at the security checkpoint. First, the commuter must show a passport and valid travel documents to prove that he or her has purchased a service from one of the airlines

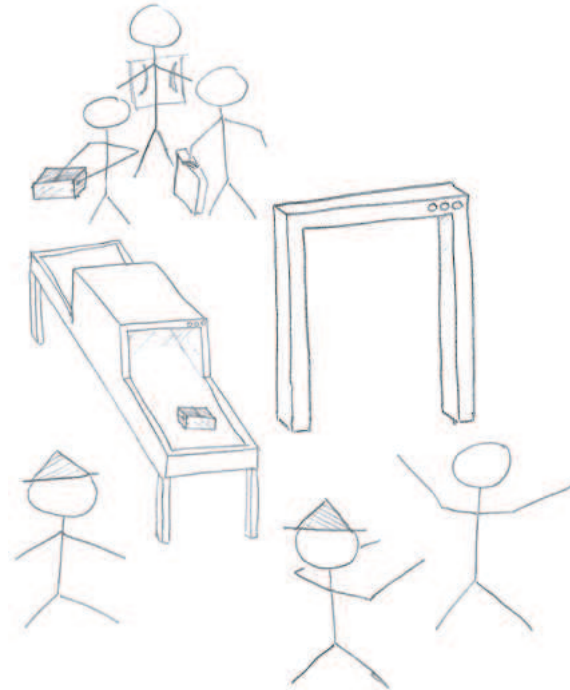


Figure 2: Passenger inspection at an airport security checkpoint

inside the security boundary. This is a precondition for input validation, and can be thought of as a firewall mechanism. Next, the security officials examine the traveler for illegal items, in accordance with a predefined list of disallowed objects. The passenger walks through a metal detector, while clothes, luggage, and other accessories are inspected with X-rays. If the airport security personnel finds it necessary, additional measures such as bomb-sniffing dogs and full body searches may be used. Travelers that do not carry prohibited items can continue their journey, while perpetrators are guided away from the security checkpoint and face further inquiries. Personnel at the checkpoint report offenses to other institutions, such as the police. Likewise, the INPUT VALIDATOR generates a `Validation Summary`, which in turn is handled by the application.

The process of explicitly specifying illegal items is more commonly referred to as *blacklisting*. This is a useful technique when you have a complete overview of the potential enemies of your system.

Implementation

Fig. 3 shows an example implementation of the INPUT VALIDATOR using firewalls [9, p. 403] and an INTEGRATION REVERSE PROXY [9, p. 465]. To implement this pattern, the following tasks should be performed

1. Identify all potentially vulnerable subsystems. Developers must single out components where message content will be processed. In Fig. 3, the stock application and the account manager can be exploited.

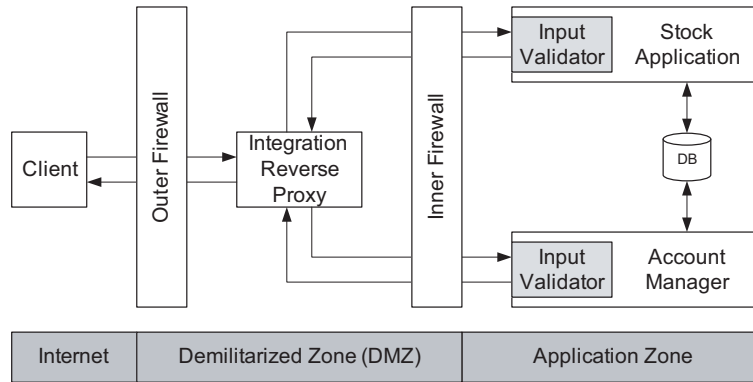


Figure 3: Example content validator architecture

2. Determine the entire set of client input sources and define what is to be considered valid input. We recommend using a whitelist approach [7, p. 71] to specify the format of valid input, also known as *positive validation*. All data not conforming to the format should be discarded.
3. Configure target objects and methods for the validators using the set of client input sources. Construct rules from the valid input definitions from the previous step, and map these rules to the corresponding target methods.
4. Deploy the validators. As shown in Fig. 3, validators should be integrated into every potentially vulnerable web application.

The input validator pattern should be combined with an accounting mechanism which can capture and store information about rule violations, and enable a reviewer to identify the involved participants and reconstruct the events that led to the violations. SECURITY ACCOUNTING REQUIREMENTS [9, p. 360] can help in selecting an appropriate accounting mechanism.

With a focus on functionality and user experience, many web application developers try to scrub content on behalf of the client. If violations are reported during validation of a newly received request, programming logic tries to alter the input with the goal of making it valid. Often, this scrubbing logic is complex and may incur significant processing overhead. A better alternative is to use the logs to evaluate the validation rules, and decide if the rules should be altered to cover a larger set of input.

Example Resolved

The Internet bank mentioned in Section 2 was able to determine the cause of the anomalies by inspecting the server logs. The emptying of client accounts was caused by a brute-force attack launched against the bank's authentication mechanism. During the analysis it also became clear that the system lacked validation of business rules.

To rectify the situation the bank integrated an input validator into their web application. The following steps were taken:

1. The authentication mechanism was improved by increasing the password from 4 digits to 6 characters, with the minimum requirements of one lower case letter, one upper case letter, and one digit. All customers were forced to change their passwords using a web form presented at their next log in. The following regular expression rule was created to enforce that new passwords adhered to the new policy:

```
^(?=.*?[a-z])(?=.*?[A-Z])(?=.*?\d)\S{6}$
```

2. A number of business rules were developed to enforce business logic when paying bills. For instance, a rule only allowing numbers greater than zero was specified for the amount field. To prevent customers from entering due dates in the past, the following rules were added:

- (a) A regular expression rule defining `dd.mm.yyyy` as the valid date format:

```
^\d{2}\.\d{2}\.\d{4}$
```

- (b) A rule for checking that the due date is the current date or in the future

Known Uses

A number of worked solutions for input validation exist

Commons Validator [10] is an open-source validation component which originated from the Apache Struts framework. Target objects are JavaBeans. Rules are named *validator actions* and are individual static boolean methods in Java objects. The component enables developers to configure target methods and rules in configuration files.

Stinger [11] is an HTTP Request Validation Engine used in a J2EE environment. The HTTP request is the target object. The engine supports regular expression rules and rules for missing or extra parts in HTTP requests. Developers can specify the rules using the Security Validation Description Language, which is tailored to map the target methods in HTTP requests.

.NET Validation Server Controls [12] are part of Web Forms in the .NET framework, and enable developers to perform input validation on client input in web applications. The target objects are the input server controls, and the framework provides predefined rules such as required field, ranges, and regular expressions. The validation controls are specified as part of the presentation logic.

Consequences

The following benefits can be expected from applying this pattern:

- Input validation can prevent attacks embedded in message content.

- Whitelisting wards off new unknown attacks that fall outside the format dictated by the validators.
- The mechanism enables accounting of attacks by capturing rule violations. The information can later be used to reconstruct attempted security breaks.
- The input validator provides a centralized and manageable mechanism for input validation.
- The same validation rule can be applied to multiple target methods. Input sources that share the same characteristics should reuse the same validation logic.

The following potential liabilities can arise from applying this pattern:

- A client request will carry additional computing overhead, causing delayed server response. The addition of validation logic also causes the server to reach its processing limits sooner.
- The added components introduce new points of failure. Potential bugs can make the application unavailable.
- The new layer of security increases system complexity, which in turn can entail a higher maintenance cost.

See Also

CLIENT INPUT FILTERS [13] discusses the input validation problem in a web context. It points out the inadequacy of client-side checks, and argues that the same checks must be carried out on the server-side as well. The pattern focuses on best practices and issues in implementing web applications.

INTERCEPTING VALIDATOR [14] is a J2EE pattern for validating client input. It is not tied to business logic, and attempts to filter out known attacks early in the request-handling process. In contrast with the INPUT VALIDATOR, the authors recommend using the INTERCEPTING VALIDATOR to also protect against injection attacks.

Useful patterns that can be beneficial in conjunction with the Input Validator pattern include accounting, firewall, and patterns for secure Internet applications [9].

Acknowledgements

Our sincere thanks to Andreas Rüping, who did a great job in shepherding our paper, providing invaluable feedback. Also, we would like to extend our deepest gratitude to the fellow members of our writers workshop group at VikingPLoP 2006: Kristian Elof Sørensen, Aino Vonge Corry, Birgit Zimmerman, Andreas Rüping, Michael Weiss, James O. Coplien, Jayashree Kar, and Rebecca Rikner.

References

- [1] Eurostat. Retrieved May 2006 from http://epp.eurostat.cec.eu.int/portal/page?_pageid=1996,39140985&_dad=portal&_schema=PORTAL&product=_STRIND&root=theme0/strind/innore/ir080&zone=detail.
- [2] ISS E-Security Alert: Form Tampering Vulnerabilities. Retrieved June 2006 from <http://www.cert-rs.tche.br/listas/infoseg/msg00033.html>.
- [3] Open Source Vulnerability Database. Retrieved June 2006 from http://www.osvdb.org/displayvuln.php?osvdb_id=18449.
- [4] J.H. Saltzer and M.D. Schroeder, “The protection of information in computer systems,” in *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.
- [5] J. Yoder and J. Barcalow, “Architectural Patterns for Enabling Application Security,” in *Proceedings of Pattern Languages of Programs (PLoP)*, 1997.
- [6] Data Validation - OWASP. Retrieved June 2006 from http://www.owasp.org/index.php/Data_Validation.
- [7] S.H. Huseby, *Innocent Code—A Security Wake-Up Call for Web Programmers*. John Wiley & Sons, first edition 2004.
- [8] OWASP Top Ten. Retrieved May 2006 from <http://www.owasp.org/documentation/topten.html>.
- [9] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns—Integrating Security and Systems Engineering*. John Wiley & Sons, first edition 2006.
- [10] Commons Validator. Retrieved May 2006 from <http://jakarta.apache.org/commons/validator/>.
- [11] Aspect Security, Stinger HTTP Request Validation Engine. Retrieved April 2006 from <http://www.aspectsecurity.com/stinger/>.
- [12] Validation Server Controls. Retrieved May 2006 from <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenre/html/cpconASPNETSyntaxForValidationControls.asp>.
- [13] Client Input Filters. Retrieved May 2006 from <http://www.scrypt.net/~celer/securitypatterns/>.
- [14] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns—Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall, first edition 2005.

**Paper IV:
Simplifying Client-Server Application
Development with Secure Reusable
Components**

Simplifying Client-Server Application Development with Secure Reusable Components

Yngve Espelid, Lars-Helge Netland, Khalid Mughal, Kjell Jørgen Hole

Department of Informatics, University of Bergen

{yngvee,larshn,khalid,kjellh}@ii.uib.no

Abstract

Internet-related crime is increasing at a rapid rate. Attackers exploit weaknesses in the Web's underlying communication protocols, which were originally designed for a non-hostile environment. Meanwhile, society is facing a large deficit of security trained people that can remedy the situation. The development of secure software is simply too complex for most of today's IT professionals. In this paper, we present a communication component that reduces the complexity involved in engineering secure client-server applications, thereby enabling software practitioners to develop more secure systems. Specifically, we propose a solution based on a Public Key Infrastructure, that encapsulates communication, allowing programmers to focus on high-level application development issues.

1. Introduction

The production of secure software often involves ad-hoc development methods. Unlike the established field of software engineering, there do not exist well-established processes that can be applied when developing a secure software system [1]. Currently, developers rely on best practices and the experience of individual team members more than on structured procedures when it comes to secure programming. In addition, the development of secure software necessitates a mindset that encompasses the eminent threat of powerful adversaries. Web-application programmers have discovered the power of the dark side the hard way. Recently, a number of vulnerabilities have surfaced, all related to the same problem: improper *input validation*. The problem and solutions are discussed in detail in *Innocent Code—A Security Wake-Up Call for Web Programmers* [2].

The design and implementation of security-critical applications lie in the intersection between cryptography, computer security, and software engineering, requiring development teams in this segment to have strong skills within all areas. The coupling of these three fields intro-

duces complexity due to conflicting interests. An example is usability versus authentication mechanisms. Also, in developing client-server applications, the choice of thread models, network programming, and session management add to the complexity.

Presently, there is a shortage of people with security background and training [3]. As a consequence, a large number of unqualified technicians are responsible for developing secure software systems. The situation in today's software industry bears much resemblance to that of the British banking industry in the 80's and early 90's, where implementations by untrained personnel were the cause for the majority of security breaches [4].

Due to the complexities involved in designing secure systems, application developers and end users cannot be expected to fully understand a security-critical program's inner workings. They will need to *trust* software provided by third parties. Designers of secure components face many challenges when it comes to building trustworthy systems. Trust is influenced by such diverse properties as a person's inclination to trust, ease of use, and availability of informational content [5]. For secure programming to succeed, security professionals must find ways to gain trust from consumers.

Client-server applications are important in offering services throughout the Internet. Citizens of Canada can log into government web servers to renew passports, file tax returns, and change home addresses [6]. As more and more authorities move services online, the same basic security building blocks will be required. Software reuse can prevent people from having to re-invent the same security functionality. Moreover, a joint effort could lead to better quality software if it can attract the attention of security experts.

In this paper, we present a communication component addressing security needs in client-server application development. The problem of input validation is tackled through definition of regular expressions. Complexity is addressed through abstraction, reducing some of the tedious network programming to a matter of configuration. The shortage of security trained professionals in the indus-

try can be overcome through software reuse. Also, it is important that users of security software can trust the system. Key ingredients in earning user trust are openness and simplicity.

The rest of the paper is organized as follows: Section 2 outlines the current state of secure software engineering and describes characteristics for good quality code, Section 3 gives an overview of the architecture and discusses the component in detail, Section 4 shows how application developers can use the component to implement an HTTP server, Section 5 emphasizes the importance of trust, Section 6 presents related work, and Section 7 provides a summary of our approach.

2. Security engineering

The production of secure software is a relatively new research field. Initial texts defining the discipline include *Building Secure Software* [7] and *Security Engineering* [8]. Current trends emphasize the importance of considering and dealing with security throughout the Software Development Life Cycle (SDLC). Retrofitting security into an existing application has proved to be more costly and less successful than including it in all development stages. A mixture of best practices, security knowledge, and security tools can be applied at various steps in the SDLC to improve software security. In the article series “Building Security In” [9], security expert G. McGraw gives a detailed presentation of security touchpoints throughout the software engineering process.

A central issue for vendors and users of secure software is the definition of security. Network security people often associate security with systems such as firewalls and intrusion detection systems. Developers of web applications tend to think of the Secure Sockets Layer (SSL) protocol as a silver bullet for web security. However, security is not a feature. It is an emergent system property. Security is the sum of a set of non-functional goals, which include procedures for prevention, traceability, auditing, monitoring, privacy, confidentiality, anonymity, authentication, and integrity [7]. Experts are needed to perform an extensive security evaluation of software, but such professionals are a rare breed. An initiative to collect and catalog the knowledge of experienced practitioners has been launched [10], but it is too early to say whether or not it will be successful. Lacking established guidelines in security engineering, there is an urgent need for key players in the field to distinguish between good and bad software. This distinction can be very hard to make prior to deployment, as most bugs and flaws are discovered during usage. A few important characteristics that can help decide the quality of code include

Documentation An undocumented system has no value in

a security setting. Documentation is a necessity in understanding what a system does and does not do. Without such an understanding people cannot trust the software.

Development process The SDLC used in developing the software must incorporate security activities such as creating abuse cases, establishing security requirements, risk analysis, external review, risk-based security tests, static analysis, and penetration testing [9].

Cryptography The design of new cryptographic algorithms should only be done by experts in cryptography. In choosing key sizes, recommendations from local authorities or “Selecting Cryptographic Key Sizes” [11] can be helpful.

History As the software is put into production, a recording of security related events such as system crashes, bug fixes, and security breaches give indications of the system’s quality. Especially interesting is the software’s ability to cope with these events, as it is an indicator of what to expect in the future.

3. The communication component

The communication component is intended to be utilized in the development of client-server applications. The goal is to properly secure transmissions of data between communicating peers. This is achieved through core security services offered by a Public Key Infrastructure (PKI) [12]. The *authentication service* identifies the participating entities and offers assurance of data origin, the *integrity service* ensures the receiver that the data has not been altered in transit and the *confidentiality service* assures the sender that only the intended receiver is able to read the transmitted data. The purpose of the communication component is to enable developers of secure software systems to use these services in conjunction with the applications’ communication protocols.

3.1. Security infrastructure

Our communication component relies on a well-functioning PKI. In the following discussion it is assumed that such an infrastructure is in place, and that it is successfully managing certificates and keys for users of the component. Both server and client side authentication will be employed. More information on PKIs can be found in *Understanding PKI* [12]. In building and managing a PKI, the OpenSSL project [13] and The Legion of the Bouncy Castle [14] can be very helpful.

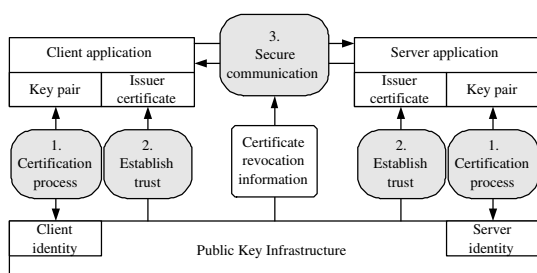


Figure 1. PKI relations

Figure 1 illustrates the necessary interactions between the communicating peers and the security infrastructure, needed to ensure secure communication. The certification process (1) binds a key pair to the entity requesting certification. The binding is represented by a signed certificate holding the entity's name and public key. These credentials can be confirmed by the entity's possession of the corresponding private key.

Before communicating, the validity of the peers' certificates must be verified. The verification is based on the common trust in the certificate issuer. In order to verify certificate signatures, the entities must obtain the issuer's certificate (2). In addition, the expiration and revocation information must be checked prior to establishing secure communication (3).

3.2. Utilizing the component

Figure 2 shows a typical context in which the communication component is deployed. The component reflects the fundamental concept of secure network communication in client-server applications. In a software engineering context, such reusable components could increase system reliability, lower development costs and reduce process risk [15]. Sensitive information is often communicated over the Internet and information assurance is an essential goal in such situations. The Internet is a hostile environment and there is a risk that sensitive information traversing the network is eavesdropped, recorded, modified or replaced. In a software development scenario, developers may choose to use the communication component to facilitate secure communication. Developers using the communication component are relieved of programming network communication, and can focus on the application logic and configuration of the component.

3.2.1. Input validation. Consider a scenario where the goal is to develop a server application, as depicted in Figure 2. The external interface comprises the port to the ser-

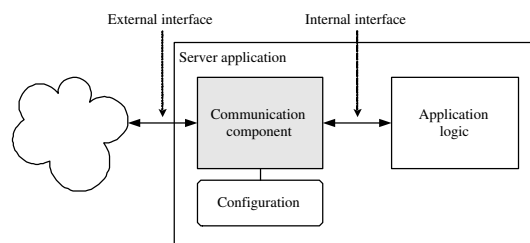


Figure 2. Server application

vice and the application's communication protocol. When servicing the port, the application introduces an entry point for external input and the issue of trust in client input arises. The majority of literature devoted to the challenge of developing secure software includes discussions and guidelines on this topic [2, 16, 17]. Software with too much trust in client input gives rise to a large amount of today's software vulnerabilities. The common principle is to consider all input from clients guilty of malicious intent until proven otherwise. The communication component forces developers to define regular expressions used to validate client input, a practice known as *white listing*. Section 4 describes in depth the use of regular expressions for this purpose.

3.2.2. Control model. The communication component fits in a centralized control model where the application logic has overall responsibility for initializing, starting and stopping the component. Figure 3 illustrates this centralized control. The application logic passes a configuration file to a factory responsible for creating a *communicating peer*. The communicating peer is then requested to start executing the communication logic, for instance servicing a port. The component is an independent executable entity which is executed in parallel to the application logic. This control model makes it easier to develop applications following the principle of graceful degradation [18]. The application logic can simply stop the communication component and try to recover if an error occurs.

3.3. Managing complexity

The concepts presented so far do not require a specific programming language. In the following section, Java will be used to implement the communication component. In a security context, Java is an attractive choice because of features such as type safety, array bounds checking, and memory management. The widely deployed SSL protocol is a natural choice in protecting network data.

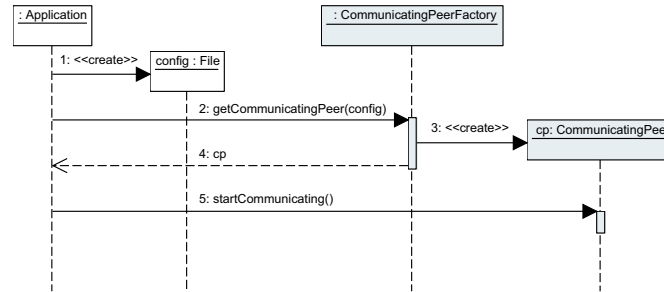


Figure 3. Centralized control

3.3.1. Advanced Java API. In version 5.0 of the Java 2 Standard Edition (J2SE) Platform, developers are given the possibility to combine SSL and enhanced I/O functionality. Prior to the latest release of Java, SSL and the New I/O (NIO) API could not be used in conjunction. The SSL API required a stream-based socket transport mechanism. In version 5.0, a new abstraction is introduced, giving programmers the freedom to choose any suitable transport mechanism. In addition, the API does not dictate a specific threading model, allowing developers to choose freely among different approaches. On the downside, the new flexibility comes with a price of added complexity. Setting up an SSL connection using the old stream-based approach was easy. First, an `SSLContext` had to be initialized with trusted certificates, the end user's own certificate, and the corresponding private key. The next and last step was simply to create an SSL client or server from the `SSLContext`. In total, setting up the connection called for about 10 lines of code on both sides of the communication. In addition, very little knowledge of the SSL protocol was needed to set up the secure channel. The only part developers had to take care of was providing the underlying PKI material, meaning the certificates to be used as part of the protocol.

Setting up SSL connections in J2SE version 5.0 necessitates a more thorough understanding of I/O, threading models and the SSL protocol. In terms of I/O, programmers must choose a specific transport model. Key ingredients in getting the I/O right include `Selectors`, `Channels` and various `Buffers`. Threading issues can be addressed using the `java.util.concurrent` package. Handling SSL requires an understanding of the protocol. In particular, developers must know the SSL Handshake Protocol. The actual processing of SSL data is taken care of by an `SSLEngine`. This engine is essentially a state machine, keeping track of the current stage in the SSL protocol. The programmer's responsibility is to send SSL data to and from the engine. The bottom-line is that in order to set up an SSL connection in the new paradigm, a lot more effort

is required. It is not surprising that Sun describes the new library as an advanced API [19]. On top of the intricacies of setting up the communication, the code for the new setup entails a greater chance of making programming errors.

3.3.2. Hiding complexity in configuration. The complexity of implementing secure network communication is reduced to a trivial configuration task when utilizing the communication component. The most obvious configurations are whether the communicating peer should operate in server or client mode, and which port the peer should service or connect to. In addition, developers can configure the threading model handling communication, either choosing a single thread, a fixed number of threads or an extensible pool of threads. Also, the component's transport mechanism can be specified, leaving it up to developers to choose from the old stream-based model or the NIO model. To use the core security services, the location of key and trust material must be specified. The most challenging task when configuring the communication component is the construction and subsequent use of regular expressions applied in validating external input.

When implementing a server application, the task of handling a substantial number of client connections is non-trivial. The communication component provides a much simpler mental model in form of a packet queue of client actions, as illustrated in Figure 4. The component's communication engine continuously handles client interactions and produces packets representing the following events:

1. Acceptance of new client connections.
2. Completion of handshakes in the SSL Handshake Protocol.
3. Conformance of client requests to the communication protocol.

The packet hierarchy representing the above events is shown in Figure 5. The `Communicator` entity represents

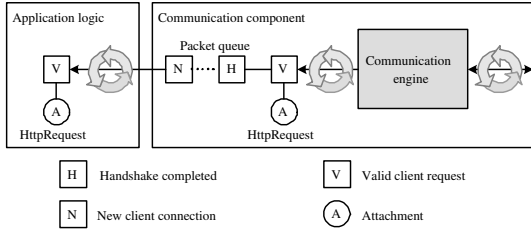


Figure 4. Packet queuing of client actions

the state of a client connection. Each `Packet` holds a reference to the `Communicator` it originated from, thereby enabling the application logic to use the `Communicator` to send application data to the client. The `Packet` entity is a generalization of the three packet categories mentioned earlier. An `Object` attached to the `DataPacket` represents application data. The next section provides details on how the communication component is deployed in a concrete application.

4. Developing an HTTP server

Figure 6 illustrates that developers utilizing the communication component can focus on high-level development issues when implementing the application logic. Network communication programming is reduced to component configuration. This section illustrates these aspects in the context of implementing an HTTP server. The goal is to facilitate secure HTTP communication.

4.1. Component configuration

The task of configuring the communication component entails selecting appropriate options for the component's parameters. These options were mentioned in Section 3.3.2. We can formulate the following criteria for implementing the HTTP server:

- The application should run in server mode, servicing port 443.
- NIO should be used to facilitate scalable network communication. The HTTP server is expected to handle a large number of concurrent connections.
- An extensible pool of threads should be used to handle the communication logic. The hardware architecture on which the HTTP server is to be deployed has multiple processors, and a multi-threaded approach is expected to boost performance significantly.

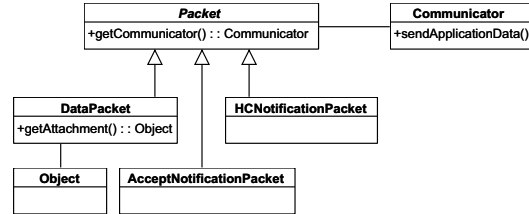


Figure 5. Packet structure

- The Transport Layer Security (TLS) version of SSL and client authentication should be employed.
- Key and trust materials exist in Java Key Stores (JKSs), using SunX509 certificate encoding.

A crucial step in the configuration is to construct regular expressions representing the application's communication protocol, i.e. HTTP. In this simple scenario only a single regular expression is required

$$(?m)(GET)(.\r\n)^+(\r\n)$$

The regular expression above matches HTTP requests using the GET method. The method line can be followed by zero or more header lines and a request is ended with an empty line. Only the structure of a request is validated by the regular expression. To facilitate validation of the content, appropriate application classes for wrapping matched character sequences must be implemented. The class names of such wrapper classes are mapped to the corresponding regular expressions. In this scenario, the `example.server.HttpRequest` class is mapped to the regular expression given above. Client input failing to match the regular expression is discarded. This white listing of client input prevents processing of bogus data.

4.2. Application logic

Configuration information discussed in Section 4.1 is provided in a configuration file, and developers need to implement application logic based on the centralized control model illustrated in Figure 3, to initialize and start a `CommunicatingPeer`. The application logic continuously services the packet queue provided by the `CommunicatingPeer`.

As illustrated in Figure 4, when the communication engine matches a character sequence with the given regular expression, it creates a `DataPacket` with an `HttpRequest` attached and adds it to the packet queue.

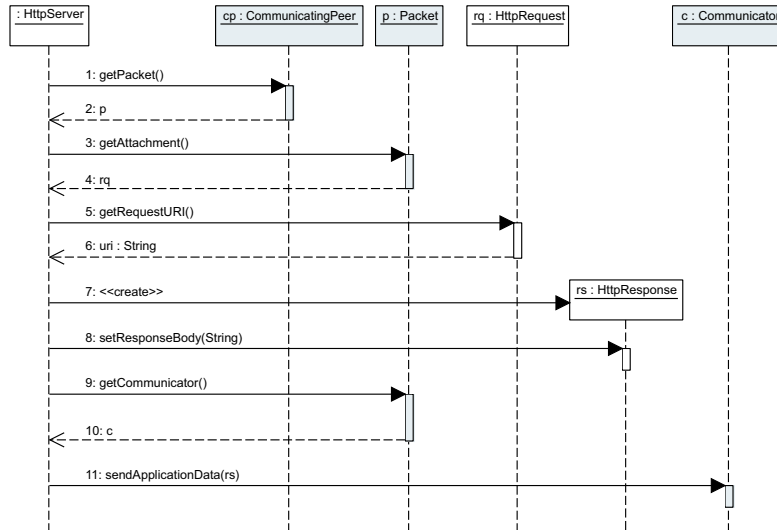


Figure 6. Handling an HTTP request

Figure 6 shows the sequence of actions in the server's application logic when handling such a request, where the numbers in the left-hand column below refer to the steps in the figure:

- 1–2 A packet, `p`, is retrieved from the packet queue.
- 3–4 The packet `p` is identified as a `DataPacket` and the packet's attachment is requested.
- 5–6 The attachment is identified as an `HttpRequest` (`rq`) and the content can be queried. In this sequence, the Unified Resource Identifier (URI) is retrieved to identify the resource targeted by the request.
- 7 Developers have implemented an `HttpResponse` class to facilitate the task of constructing responses to client requests. An object of this class is instantiated.
- 8 In case of a static resource targeted by the request, the content of the resource is set as the body of the `HttpResponse`.
- 9–10 To send this response back to the client, the application logic fetches the `Communicator` (`c`) representing the client connection.
- 11 The `Communicator` (`c`) facilitates the sending of the response to the client.

The application logic also retrieves packets representing new client connections or completion of the SSL Handshake Protocol. If no specific actions are planned for these events, these packets can safely be ignored. Otherwise, some application logic handling these packets must be implemented. For instance, when a client completes the SSL Handshake Protocol and the notification packet is retrieved, the application logic can initiate a client session. The name in the client's certificate can in such scenarios be used to distinguish client sessions.

5. Trust

The success of any piece of security software is closely coupled with its trustworthiness. If potential customers cannot trust your product, they will look for other alternatives. Trust is an active area of research. An overview of the current status quo can be found in *Security and Usability* [5]. Patrick defines software agent success in terms of trust and perceived risk [20]. Factors such as experience with a given product, a user's inclination to trust, system predictability, and interface design determine how much a person is willing to trust the software. On the opposing end, an increasing amount of risk will refrain users from using a software system. Risk is, among other factors, influenced by how well-documented a system is, how much personal information users have to divulge, and the degree of application autonomy.

Our approach must build trust towards two groups; de-

velopers and end users. The former is a direct relationship, where programmers must be satisfied with the level of security facilitated by the communication component. The latter is of indirect nature, an example of propagation of trust, where developers must extend their trust to potential customers. In this regard, our task is to make the transfer of trust possible. The authors believe that two factors are important in achieving trust:

Openness Comprehensive information about the project will be made available to mitigate feelings of risk. We strongly believe that documented source code alone is not sufficient documentation of a secure system. A detailed overview of the software development process and deliverables from activities throughout the SDLC should be made available to the involved parties. With better access to information, consumers should feel more comfortable in making trust decisions.

Simplicity Interface design and ease of use are addressed through a simple mental model that intuitively explains the workings of the component. Our packet queue model serves as an example of such simplification.

6. Related work

OpenSSL [13] is a very popular open-source library used to implement secure network communication. The approach presented in this paper could be used in developing a similar component based on the OpenSSL API. We do not know of any such initiative. From a security standpoint, the OpenSSL library is unattractive due to its incomplete documentation. In contrast, Sun's Java API is fully documented. The OpenSSL community is currently working on improving the project's documentation.

Unvalidated client input, also known as *tainted input*, has received much attention recently. Haldar et al. [21] describe an approach to tag and track tainted data throughout its lifetime in Java. The proposed framework reports cases where input is passed directly to security-sensitive methods without validation. It requires some instrumentation of core string classes in Java. Similar to our approach, developers are forced to think about sanitation of input. However, a tainted string will become untainted by a successful call, i.e. returning true, to any of its string checking or matching methods. This is regardless of the developers intended use. In contrast, the communication component presented in this paper forces application developers to write regular expressions and wrapper classes to validate the structure and content of network data. While the communication component is intended to be used in new development and re-factoring, the tainted-input approach is independent of source code, enabling use in already deployed software.

Probst et al. argue the need for reusable high-level security constructs [22]. They describe a framework called Generic Authorization Mechanisms for Multi-Tier Applications (GAMMA) that offers authentication, access control, and auditing mechanisms. GAMMA separates security functionality from business logic through dedicated security components, residing between the application and backend layers.

7. Summary

There is a shortage of professionals trained to build secure applications. The situation necessitates a transfer of knowledge from security professionals to software engineers. In particular, incorporating explicit security activities in the SDLC and an understanding of security best practices are important factors in building secure software. Our contribution is a client-server communication component that reduces the complexities of programming secure networking code to a configuration task. As an example, we have shown how an HTTP server can be implemented using the communication component. We intend to deploy the communication component in other client-server applications to further test the validity of our approach.

We thank the anonymous reviewers for their comments, which greatly improved this paper.

References

- [1] J. Whittaker, "Why Secure Applications are Difficult to Write," *IEEE Security & Privacy*, vol. 1, no. 2, 2003, pp. 81–83.
- [2] S.H. Huseby, *Innocent Code—A Security Wake-Up Call for Web Programmers*. John Wiley & Sons, first edition 2004.
- [3] President's Information Technology Advisory Committee, "Cyber Security: A Crisis of Prioritization," February 2005.
- [4] R. Anderson, "Why Cryptosystems Fail," ACM 1st Conference on Computer and Communication Security 1993.
- [5] Editors L. Cranor and S. Garfinkel, *Security and Usability—Designing Secure Systems that People Can Use*. O'Reilly, first edition 2005.
- [6] Government of Canada Official Web Site. Retrieved January 2006 from http://canada.gc.ca/main_e.html
- [7] J. Viega and G. McGraw, *Building Secure Software—How to Avoid Security Problems the Right Way*. Addison-Wesley, first edition 2002.
- [8] R. Anderson, *Security Engineering—A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., first edition 2001.
- [9] G. McGraw, "Software Security," *IEEE Security & Privacy*, vol. 2, no. 2, 2004, pp. 80–83.
- [10] S. Barnum and G. McGraw, "Knowledge for Software Security," *IEEE Security & Privacy*, vol. 3, no. 2, 2005, pp. 74–78. Portal website:

- <https://buildingsecurityin.us-cert.gov/portal/>
- [11] A.K. Lenstra and E.R. Verheul, "Selecting Cryptographic Key Sizes," *Practice and Theory in Public Key Cryptography*, PKCS 2000.
 - [12] C. Adams and S. Lloyd, *Understanding PKI—Concepts, Standards, and Deployment Considerations*. Addison-Wesley, second edition 2003.
 - [13] OpenSSL. Retrieved September 2005 from <http://www.openssl.org>.
 - [14] The Legion of the Bouncy Castle. Retrieved September 2005 from <http://www.bouncycastle.org>.
 - [15] I. Sommerville, *Software Engineering*. Addison-Wesley, sixth edition 2001.
 - [16] M. Howard and D. LeBlanc, *Writing Secure Code—Practical Strategies and Techniques for Secure Application Coding in a Networked World*. Microsoft Press, second edition 2003.
 - [17] G. Hoglund and G. McGraw, *Exploiting Software—How to Break Code*. Addison-Wesley, first edition 2004.
 - [18] M.G. Graff and K.R. van Wyk, *Secure Coding—Principles & Practices*. O'Reilly, first edition 2003.
 - [19] Java Secure Socket Extension Reference Guide. Retrieved September 2005 from <http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html>
 - [20] A.S. Patrick, "Building Trustworthy Software Agents," *IEEE Internet Computing*, November-December 2002.
 - [21] V. Haldar, D. Chandra, and M. Franz, "Dynamic Taint Propagation for Java," 21st Annual Computer Security Applications Conference 2005.
 - [22] S. Probst, W. Essmayr, and E. Weippl, "Reusable Components for Developing Security-Aware Applications," 18th Annual Computer Security Applications Conference 2002.

Paper V:
A Reflection-Based Framework for
Content Validation

A Reflection-Based Framework for Content Validation

Lars-Helge Netland
Department of Informatics
University of Bergen
Email: larshn@ii.uib.no

Yngve Espelid
Department of Informatics
University of Bergen
Email: yngvee@ii.uib.no

Khalid A. Mughal
Department of Informatics
University of Bergen
Email: khalid@ii.uib.no

Abstract—Attacks embedded in application-level data have become one of the most successful ways to circumvent software security. Skilled hackers capitalize on misplaced trust by concealing their malicious code within a seemingly innocuous stream of application data. In systems that do not perform the most elementary data checks, even unintentional user mistakes may cause a program to behave unexpectedly or crash.

Any distributed software system with potentially untrustworthy sources of input should design and implement a mechanism to inspect application-level data. Such a solution should defend against mischievous attacks, as well as be robust enough to handle user slip-ups. Important steps in creating a successful validation regime include specifying what input to accept, and translating that policy into working code. Once in production, the validation routine must be adaptable in order to accommodate continuously changing requirements.

This paper describes a reflection-based framework for content validation. It separates the inspection of data from the application logic, making it more feasible to construct and maintain a meaningful set of validation rules. The framework is flexible and can be integrated into almost any distributed object-oriented software system. Deployment only requires a basic understanding of XML and expects developers to create a trust model of their own software architecture.

Keywords: content validation, distributed systems security, maintainability, reflection, robustness.

I. INTRODUCTION

A common mistake among developers is misplaced trust in sub-systems. Problems occur when the exchanged bytes deviate from the expected format, which could be orchestrated by an attacker or inadvertently caused by a user. A viable I/O validation strategy combines software security knowledge with risk management analysis, and must be continuously revised to face new threats. A thorough understanding of the environment where the software runs is important in establishing a trust model and identifying forces that can have an impact on the overall security of the system.

Unvalidated input ranks as number one on the Open Web Application Security Project's top ten list [1] over web application vulnerabilities. The problem of handling content is not limited to a Web context, but is equally important in distributed client-server systems running over protocols other than HTTP. The solution described herein is limited to object-oriented software systems, but does not dictate a specific application type or domain. Content validation should be done

prior to processing of data from untrusted sources, and applied throughout the system as needed.

Content validation requirements are likely to change over time, resulting from e.g. application development, feedback from the user community, or anomalies detected during inspection of server logs. It is therefore important to design a solution that can be quickly adjusted to meet changed requirements. The content validator proposed in this paper uses XML and reflection to tackle the dynamic nature of the problem. These two technologies in combination are promising candidates for extending the longevity of software, and can achieve platform-independency [2]. In addition, the suggested solution helps structure the validation process through defining categories of validation rules. These categories encourage developers to think of different types of validation rules.

The work described in this paper builds on [3] and [4]. The former describes a secure communication component that allows developers to focus on application-level programming instead of tedious networking issues, while the latter presents a security pattern for input validation. The framework discussed herein is now available as a SourceForge project called Heimdall [5] under an MIT license.

The rest of this paper is organized as follows: Section II defines content validation and gives an overview of the problem domain; Section III describes other important defense mechanisms that should be considered; Section IV introduces an online banking scenario to show how the framework can be used; Section V explains the inner workings of the content validation framework; Section VI elaborates on the implementation of the online banking example; Section VII discusses related work; Section VIII addresses future work; and Section IX concludes the paper.

II. MOTIVATION

In a broad sense, I/O can be defined as the *signals* that sub-systems of an information processing system use to communicate with each other. Information received by a functional unit is called *input*, while information sent by the unit is called *output*. In our context, examples of such units include web servers, Internet browsers, databases, and operating systems. The above-mentioned signals are synonymous with data, which can be more appropriately defined as *structured content*. The structure is metainformation, i.e. it describes the format of

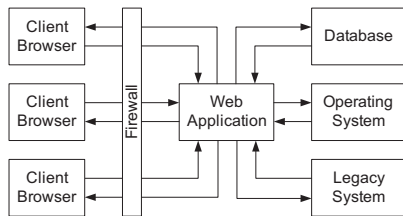


Figure 1. Architecture

data in a given protocol. The content is the actual information populating the structure. For the purpose of this paper I/O can be defined as structured content exchanged between sub-systems of a larger system.

I/O validation is the process of checking the structure and the contents of data against validation rules. The former means verifying that the information conforms to the protocol format. The latter requires a more careful inspection of the bytes at hand. The difference between structure and content validation can be explained in terms of XML parsing. Checking that an XML message adheres to the Document Type Definition [9] addresses structure validation, while content validation requires analysis of the element and attribute values.

It is also worth noting that classifying information as input or output is a matter of point of view. From the viewpoint of a web server, an HTTP request is input and the corresponding response is output. For an Internet browser, it is reversed. Our communication model does not fit in the producer-consumer paradigm, as many sub-systems take on both roles. As a consequence, the distinction between input and output becomes unimportant. The crucial factor is to establish where the content is to be processed.

Most systems are comprised of several different sub-systems. A typical scenario is shown in Fig. 1. Customers request services from a web server through Internet browsers. A web application receives the initial traffic, and communicates with the back-end sub-systems when serving the clients. A firewall protects against network-level attacks.

A sequence of steps must be completed to create a well-functioning distributed system:

- 1) Identify sub-systems and dataflow. Pinpoint the potential targets for an adversary. Any component in the system that processes input is a possible target.
- 2) Implement the system using defense mechanisms ensuring secure sub-system interaction.
- 3) Specify content validation rules. Define what is to be considered valid content for the system as a whole and

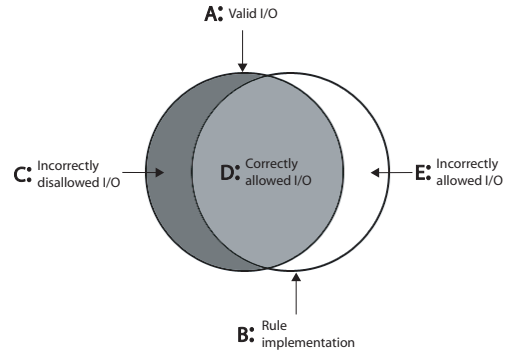


Figure 2. Implementation mismatch

for each information processing sub-system.

- 4) Implement validation rules. Translate the rule specification into working code and deploy as needed throughout the system.

Step 1 is carried out during design of the system; step 2 is discussed further in Section III; step 3 is exemplified in Section IV by formulating validation rules for bill payment in an online banking scenario; and step 4 is illustrated in Section VI by implementing the rules for bill payment using the framework.

Implementing a perfect I/O validation routine is a very difficult task, mainly due to the rapidly changing environment in which the software operates. As new threats and customer requirements emerge, the solution must be continuously updated to accommodate the changes. Fig. 2 illustrates the relationship between a perfect I/O validation mechanism and an actual implementation. Region A symbolizes the optimal solution, while region B represents a deployed validation routine. The intersection between the two circles, i.e. region D in Fig. 2, embodies I/O that is correctly accepted by the validation logic. Region C represents I/O that is rejected by the current implementation, but should be considered valid according to system policy. This category of I/O does not pose a direct threat in terms of security, but could become a major inconvenience for legitimate users, as input they rightfully consider valid is rejected. Fixing the problem usually involves a user reporting the input that caused a problem to a system operator, who in turn updates the set of rules to accept the incorrectly rejected input. Region E in Fig. 2 represents invalid input that passes undetected through the validation mechanism. Failure to recognize and reject such I/O can have serious security implications. Regions C and E are termed false positives and false negatives, respectively, in the Intrusion Detection Systems community.

Developers should strive to achieve as much overlap as possible between region A and B in Fig. 2. An optimal I/O validation strategy involves maximizing the intersecting region D and minimizing regions C and E. Managing the region E is the key to improving security.

Minimizing region E is not a trivial task. A number of approaches can enhance the quality of I/O validation. Examples include hiring external security expertise for a given application domain, who can help formulate validation rules; investing time and effort into a suitable logging and monitoring system that can reveal new validation needs; and making extensive use of feedback from the user community. Our contribution is a framework that makes it easier to create and maintain a validation regime. At the heart lies the ability to adapt the framework to changing validation requirements. As will be demonstrated later, XML and reflection provide this flexibility.

III. DEFENSE MECHANISMS

Proper sub-system communication is essential for a well-functioning system. How sub-systems exchange information depend on how they interface. Factors affecting these relationships include the communication techniques offered by the technology and how these mechanisms are made available through APIs. Over time, the communication forms tend to change, often driven by new performance and security requirements. Under these circumstances, developers choose implementation strategies for sub-system communication.

The most dominant server-side sub-system relationship is that of a web application interacting with a back-end database. The first querying technique was string-based, and involved populating SQL commands with client input values. The query string was subsequently submitted and executed in the database. This technique has led to one of the most common venues for client input attacks, namely SQL injection [6]. In short, the attacker alters the querying logic constructed in the web application, by submitting cleverly crafted input, often involving SQL metacharacters. The first defense mechanism against such attacks was to *escape* metacharacters in client input, forcing a literal interpretation in the parser. This is a common approach often used to protect other similar sub-system interactions.

A better solution is to use prepared statements that separate application and querying logic, rendering the context switching attacks impossible. The use of prepared statements removes the need for escaping routines prior to database communication. This evolution highlights the diversity of implementation choices software developers face, and how much proper sub-system communication depends on their knowledge base.

As a contrast to the defense mechanisms just discussed, the main goal for content validation is *not* handling metacharacters or ensuring proper internal communication in a system. The purpose of content validation is to ensure that input conforms to policy rules specified for the system domain. Such rules can define valid structure in credit card numbers, e-mail addresses, etc. Thus, if a rule specifies a valid input set excluding sub-system metacharacters, content validation can have the side-effect of defending against injection attacks as well.

In large software development projects, content validation rules are often implemented by different developers throughout the system code. As a consequence, the larger the system gets,

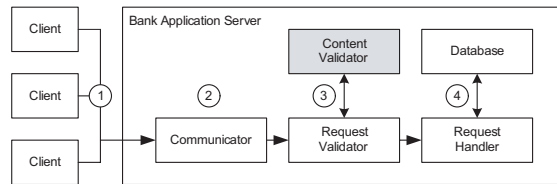


Figure 3. Online bank architecture

the more difficult it becomes to maintain these rules when requirements change. The process also entails re-compilation and re-deployment of system components.

The following sections demonstrates our generic solution for content validation. In Section IV, we present the architecture of an online banking application and formulate policy rules. The theoretical underpinnings of the framework are presented in Section V, while Section VI shows how to translate the rules specified in Section IV into working code.

IV. RUNNING EXAMPLE: ONLINE BANKING

Online banking provides a typical scenario in which the content validation framework is useful. Fig. 3 presents the data flow when a client sends a request to the online banking server. The circled numbers indicate the order in which the information propagates, with the lowest number representing the initial point of contact. A scenario for handling a client request can be described by the following steps:

- 1: A bank client sends a request to the bank server.
- 2: A `Communicator` parses the data and creates objects representing the request, based on a communication protocol. These objects are passed to a `Request Validator`.
- 3: The `Request Validator` uses a `Content Validator` to determine which objects in the request are valid. The result of the validation process is a summary of rule violations. Invalid objects are not processed, instead such requests are handled in accordance with policy. E.g., a critical error could cause the bank server to log the incident and terminate the client connection. Valid requests are sent to the `Request Handler`.
- 4: The `Request Handler` determines what to do with the request. Interaction with back-end resources, such as a database, may be necessary to fulfill the request.

The task of developing a secure `Communicator` is discussed in [3].

Describing the validation requirements for an entire Internet banking system falls outside the scope of this paper. We have chosen a scenario for bill payment to show how content validation is set up. The techniques presented are applicable in similar contexts. The example is implemented in Java, and the

Amount	€ 25 99 c
Payment date	25.04.2006
From account	-----select-----
To account	3546 7645 78 9928315026
Pay bill	

Figure 4. Bill payment web form

enterprise owner is a made-up Spanish bank that needs content validation of domestic payments. The bank uses a web form for bill payments, as shown in Fig. 4. Bank customers are expected to fill in the amount in euros and cents, a payment date, from which account the amount should be withdrawn, and to which account the money should be transferred. Based on application domain expertise, the bank development team can now formulate validation rules for the different fields in Fig. 4:

Amount Let x denote the euro entry, and y the cent value. Then the following relations must be satisfied:

$$0 \leq x < 1000000$$

$$0 \leq y \leq 99$$

$$x + y > 0,$$

meaning that the amount must be greater than zero and can be up to one million euro.

Payment date A valid date has the format $dd.mm.yyyy$, where all entries are integers. It should not be possible to backdate bills, i.e. the date must be equal to or after the current date. Also, the date must be a valid Gregorian calendar date.

From account/To account Must be 20 digits, in compliance with Spanish domestic account numbers.

V. A GENERIC VALIDATION MECHANISM

This section provides a top-down walk-through of our content validation approach. First, we present an overview of the main concepts in our work. Next, a class diagram accounts for the core entities and their relationships. After describing the important elements in our solution, a sequence diagram is presented to show how they all interact. In addition, different categories of content validation rules are presented.

A. Overview

The `Validator` is the driver of the validation process and is the main entity in our validation framework. The validation logic for a `Validator` is configurable in XML. Multiple `Validators` can be created and used throughout the system as needed. Fig. 5 shows how the application can fetch a named `Validator` (1–2) and run the necessary communication in

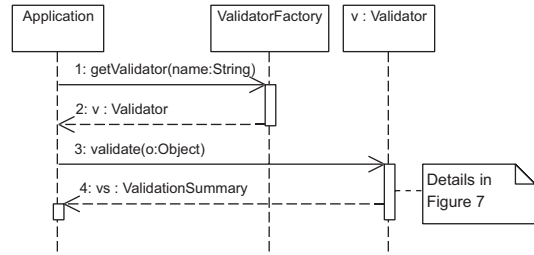


Figure 5. Using a validator

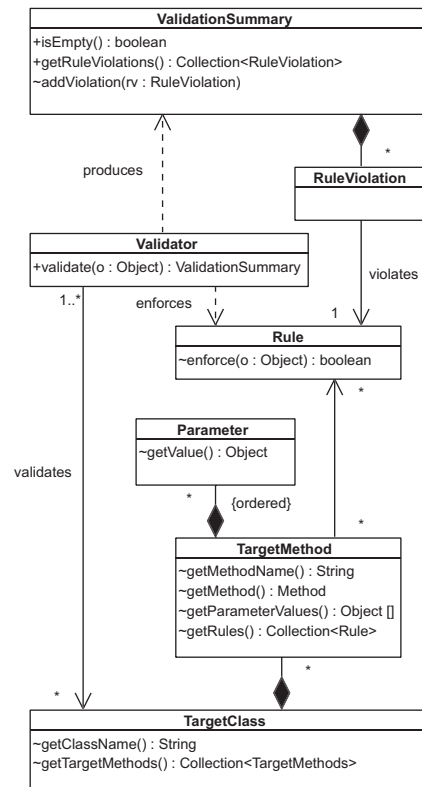


Figure 6. Validation entities

order to validate a given object. When the application has obtained an object o , it is submitted for validation to the `Validator` (3). The results from the validation process are summarized in a `ValidationSummary` and returned to the application (4). The application can now take action depending on the information provided in the `ValidationSummary`.

B. Main entities

Fig. 6 identifies the main entities involved in the validation process. The `Validator` holds a collection of target classes representing objects that must be inspected. Each `TargetClass` holds information on their `TargetMethods` and corresponding method `Parameters`, enabling the `Validator` to retrieve these object properties. In addition, sets of rules define valid return values `TargetMethods` can produce. The `Validator` enforces these rules and produces a `ValidationSummary` containing rule violations. `Rules` and `RuleViolations` represent content validation rules and potential violations of these rules. Return values can be compounded, which necessitates recursive validation.

C. Validation process

Since the content validation framework is generic and not specific to any application domain, it is unaware of target classes and methods. Class reflection [10] enables the validation framework to determine the class of an object at runtime and invoke its target methods. The configuration of the `Validator` therefore entails specifying class and method names, and defining the corresponding sets of rules. The use of reflection leads to a flexible and dynamic solution for content validation.

Fig. 7 shows the sequence of steps after the target class has been established. The numbers in the left-hand column refer to the steps in the figure.

D. Rules

Rules constitute the implementation of validation logic. A few common rule categories are already available in the validation framework. For each category, developers can specify initialization parameters, such as range limits, to employ instances of rules in the validation process. The following list describes the different categories:

Range Strings, integers etc. should be within certain ranges given by minimum and maximum values.

Boolean Boolean values should be either `true` or `false`.

Required Values should not be `null`.

Pattern Character sequences should adhere to regular expressions.

Schema Character sequences should conform to XML schemas.

In scenarios comprising complex validation, the rule categories given above may be insufficient. This limitation is overcome by allowing developers to implement custom validation logic and incorporate it into the framework. By specifying class and method names identifying these custom validation rules, developers are able to use their own validation logic as part of the validation process.

VI. ONLINE BANKING IMPLEMENTATION

In this section we describe the technical details of the solution implemented for the Spanish bank introduced in Section IV. It should be noted that the validation framework is very flexible, and the following implementation only serves as an example.

First, the validation rules specified in Section IV are translated into XML elements that can be executed by a `Validator`. The rule for payment amount is expressed as follows:

```
<rule name="paymentAmount">
  <range type="java.lang.Double">
    <min>0.01</min>
    <max>999999.99</max>
  </range>
</rule>
```

The Content Validator comes with *built-in* validation logic for ranges. The type must be specified, and the `min` and `max` elements define the valid interval. In this case from 1 cent, inclusive, up to 1 million euros, exclusive.

In most cases, context-specific validation rules must be implemented. Developers can create their own *custom rules* by creating Java classes with methods to enforce these rules. The following XML element for payment date validation shows how to specify custom rules:

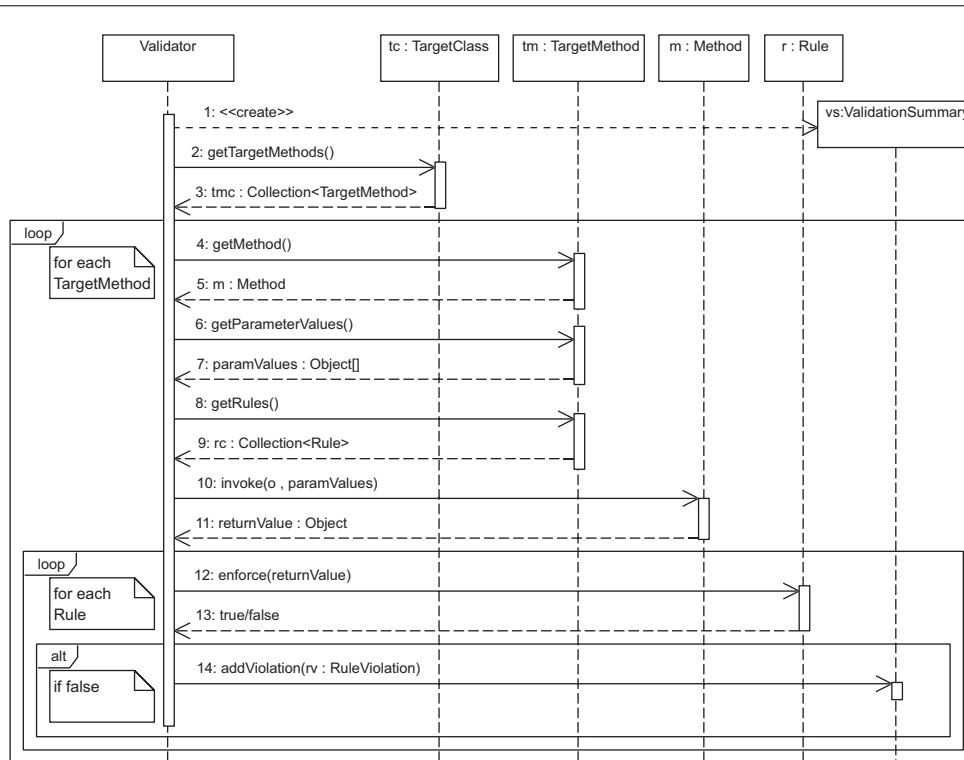
```
<rule name="paymentDate">
  <custom>
    <class>PaymentDateRules</class>
    <method>todayOrInTheFuture</method>
  </custom>
</rule>
```

The `class` element specifies a Java class named `PaymentDateRules`, and the `method` element identifies its static boolean method called `todayOrInTheFuture`. This method validates the date returned by the `Payment` object.

Analogues to range validation, the content validator comes with built-in functionality to specify *regular expression* rules. The following XML segment ensures that an account has exactly 20 digits:

```
<rule name="spanishAccountNumber">
  <regex>^\d{20}$</regex>
</rule>
```

Next, the validation rules need to be associated with target methods. In our simple scenario, a single class holds the payment data. Upon receiving a client request, the `Communicator` shown in Fig. 3 creates a `Payment` object based on the web form input. Fig. 8 shows the getter methods in the `Payment` object that enable the `Validator` to retrieve the payment data. The construction of business objects in the `Communicator` forces a partial syntactical validation of the client data. For instance, a non-numeric amount of euros or cents results in a `NumberFormatException` when



1: The Validator creates a ValidationSummary to hold rule violations.

2-3: The Validator queries the TargetClass for the collection of TargetMethods that should be evaluated on this object.

The rest of the steps are repeated for all target methods in the collection

4-5: The Validator retrieves the actual object m which provides access to the class method.

6-7: The parameter values for this method is retrieved by the Validator.

8-9: The set of rules defining a valid return value is retrieved.

10-11: The Validator invokes the method on the object o.

The rest of the steps are repeated for all rules in the set

12-13: The return value from step 10-11 is submitted for rule validation.

The last step is performed if the validation of the rule returns false.

14: A rule violation is added to the ValidationSummary

Figure 7. Validation process

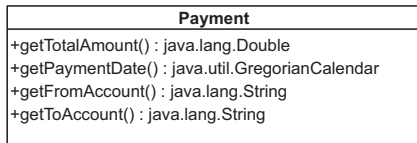


Figure 8. Object for holding payment data

instantiating the `java.lang.Double` object representing the total amount.

Below, the target methods in the `Payment` class are associated with the validation rules specified earlier:

```
<class name="Payment">
  <method name="getTotalAmount">
    <rulebinding>
      <rule>paymentAmount</rule>
    </rulebinding>
  </method>

  <method name="getPaymentDate">
    <rulebinding>
      <rule>paymentDate</rule>
      <message type="USER">
        Payment date must be the current
        date or a date in the future
      </message>
    </rulebinding>
  </method>

  <method name="getFromAccount">
    <rulebinding>
      <rule>spanishAccountNumber</rule>
    </rulebinding>
  </method>

  <method name="getToAccount">
    <rulebinding>
      <rule>spanishAccountNumber</rule>
    </rulebinding>
  </method>
</class>
```

The `getTotalAmount` method is linked to the `paymentAmount` rule. In addition, different types of messages can be specified in the configuration and used for various purposes such as logging and user response. Above, the string 'Payment date must be the current date or a date in the future' can be returned to the client when the system encounters a backdated bill. Also, validation rules can be reused, as illustrated by the `spanishAccountNumber` rule associated with both the `getFromAccount` and `getToAccount` methods.

Multiple validators can be used in a system. Each one of them specifies which classes it validates, as shown below:

```
<validator name="bankValidator">
  <class>Payment</class>
  ...
</validator>
```

VII. RELATED WORK

Commons Validator [11] is an open-source project originating from the Apache Struts framework that addresses input validation. Their approach centers around a configurable validation engine and a dynamic set of reusable validation methods. The project uses the Java reflection API to create a flexible solution that allows developers to configure and run their own validation logic. The Commons Validator initiative has close ties to JavaBeans, and is primarily concerned with validating fields in Web forms.

Similar solutions include Stinger [12] and Server Validation Controls [13]. The former is an HTTP request validation engine, while the latter provides validation capabilities for Web forms in the .Net framework. Our approach does not require a specific domain or technology, and can be applied in all distributed systems that rely on sub-system communication. E.g. the running example from Sections IV and VI could have been implemented as proprietary bank client software relying on other standards than HTTP and HTML.

Marking and tracking potential harmful data, better known as *tainting*, has been suggested as a solution to the input validation problem. The technique is available in a few programming languages, including Perl [14] and Ruby [15]. In addition, Ngyen-Tuong *et al.* [16] and Haldar *et al.* [17] have developed extensions that enable tainting in PHP and Java, respectively. The taint approach can be fruitful for already deployed applications that were designed without the input validation problem in mind. In such scenarios, tainting can be applied at run-time to identify and track potentially malicious I/O. Any subsequent security sensitive operations involving tainted data are disallowed. This category of input must be *untainted*, i.e. validated, prior to further processing. Developers are trusted to perform meaningful validation. In essence, tainting is an awareness technique that forces people to think about input validation. The approach is a quick fix to cover up for a design flaw, namely a failure to identify untrusted I/O sources. Our solution addresses untrustworthy I/O in the first of the four steps given in Section II, rendering tainting useless, when building new software systems.

VIII. FUTURE WORK

Reflection incurs performance overhead. To overcome the performance penalty, the framework could replace reflection with code generation, which provides the `Validator` with direct access to target objects. A comparative study between the two techniques would reveal how costly reflection is. We plan to further investigate the relationship between reflection and code generation, and benchmark the results against other similar solutions, such as Commons Validator and Stinger.

IX. CONCLUSION

Many current distributed software systems lack or implement poor I/O validation. Attackers manipulate the structure or contents of application-level data in order to sidestep defensive measures.

In this paper, we have proposed a content validation strategy to address attacks embedded in message content. The approach includes activities throughout the Software Development Life Cycle. In the design phase it is important to establish trust relationships. All input from sources that are not completely trustworthy should be validated. Implementing the validation logic boils down to configuration, giving developers the opportunity to focus on constructing a viable set of validation rules. Our validation package offers some basic general-purpose rules that can be applied. Developers have the option to add their own rules.

REFERENCES

- [1] OWASP Top Ten Project. Retrieved November 2006 from http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- [2] J. Bishop and N. Horspool, "Cross-Platform Development: Software that Lasts," *IEEE Computer*, October 2006, pp. 26–35.
- [3] Y. Espelid, L-H. Netland, K.A. Mughal, and K.J. Hole, "Simplifying Client-Server Application Development with Secure Reusable Components," Proc. International Symposium on Secure Software Engineering (ISSSE), Washington D.C. USA, March 2006.
- [4] L-H. Netland, Y. Espelid, and K.A. Mughal, "Security Pattern for Input Validation," Proc. Viking Pattern Languages of Program (VikingPLOP), Helsingør, Denmark, Sept.-Oct. 2006.
- [5] Heimdall, SourceForge project. Retrieved November 2006 from <http://www.sourceforge.net/projects/heimdall>
- [6] S.H. Huseby, "Innocent Code—A Security Wake-Up Call for Web Programmers. John Wiley & Sons, 2004.
- [7] M. Howard, D. LeBlanc, and J. Viega, *19 Deadly Sins of Software Security—Programming Flaws and How to Fix Them*. McGraw-Hill/Osborne, 2005.
- [8] G. McGraw, "Software Security," *IEEE Security & Privacy*, vol. 3, no. 2, 2004, pp. 80–83.
- [9] Extensible Markup Language (XML) 1.0 (Third Edition). Retrieved November 2006 <http://www.w3.org/XML/>
- [10] Reflection. Retrieved November 2006 from <http://java.sun.com/j2se/1.5.0/docs/guide/reflection/index.html>.
- [11] Commons Validator. Retrieved November 2006 from <http://jakarta.apache.org/commons/validator/>.
- [12] SourceForge.net: Stinger HTTP Request Validation Engine. Retrieved November 2006 from <http://sourceforge.net/projects/stinger/>.
- [13] Validation Server Controls. Retrieved November 2006 from <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenre/html/cpconASPNETSyntaxForValidationControls.asp>.
- [14] Introduction to Perl's Taint Mode. Retrieved November 2006 from <http://www.webreference.com/programming/perl/taint/>.
- [15] Programming Ruby: The Pragmatic Programmer's Guide. Retrieved November 2006 from <http://www.rubycentral.com/book/taint.html>.
- [16] A. Nguyen-Tuong, S. Guarnieri, D. Green, J. Shirley, and D. Evans, "Automatically hardening web applications using precise tainting," Proc. The International Federation for Information Processing Security Conference (IFIP sec), May 2005.
- [17] V. Haldar, D. Chandra, and M. Franz, "Dynamic Taint Propagation for Java," Proc. Annual Computer Security Applications Conference (ACSAC), Dec. 2005.

**Paper VI:
A Proof of Concept Attack against
Norwegian Internet Banking Systems**

A Proof of Concept Attack against Norwegian Internet Banking Systems

Yngve Espelid, Lars–Helge Netland, André N. Klingsheim, and Kjell J. Hole
{yngvee, larshn, klings, kjellh}@ii.uib.no

NoWires Research Group
Department of Informatics
University of Bergen, Norway
Short paper version, Dec. 10th, 2007

Abstract. The banking industry in Norway has developed a new security infrastructure for conducting commerce on the Internet. The initiative, called BankID, aims to become a national ID infrastructure supporting services such as authentication and digital signatures for the entire Norwegian population. This paper describes a practical man-in-the-middle attack against online banking applications using BankID. The attack gives an adversary access to customer bank accounts in two different online banking systems. Proof of concept code has been developed and executed to demonstrate the seriousness of the problem.

1 Introduction

The Norwegian banking community has created a new infrastructure for secure e-commerce, called BankID.¹ As of October 2007, BankID has more than 700,000 end-users. This number is expected to exceed 1.5 million come 2008. At the time of writing, the infrastructure is mainly used for authentication of Internet banking customers, but BankID is extending into other markets, such as the government sector and e-commerce in general. It has also been used in conjunction with e-voting in some companies. BankID won a European prize, namely the *ema Award for Excellence in Secure Electronic Business* in 2006. The system is modeled after a Public-Key Infrastructure (PKI), where the banks themselves own and operate the central infrastructure. Within a few years, the Norwegian banking industry wants BankID to become a nationwide identity system.

No publicly available independent third party evaluation of the system confirms that BankID meets a minimum of security and privacy requirements. This is worrisome for several reasons: Firstly, a report by the US National Research Council [1] states that public review is essential when developing a nationwide identity system. The social costs of a poorly thought-out system are simply too high to justify. Secondly, the banking industry both owns the BankID infrastructure and provides financial services on top of the framework. It is not clear how potential conflicts of interest, involving the bank as a service provider and

¹ Not to be confused with the Swedish BankID initiative.

PKI operator, will be resolved. Uncontested, the combination of no trusted third party and a security-through-secrecy policy could undermine the legal protection of Norwegian bank customers. This issue was explored in depth in a previous report that performed a risk analysis of the BankID infrastructure [2]. Our work was done in parallel with the mentioned evaluation, and examines the therein suggested Man-in-the-Middle (MitM) attack in detail.

This paper is organized as follows: Section 2 looks at BankID from the attacker's point of view and describes a MitM attack against Norwegian online banks that use the security infrastructure; Section 3 provides improvement suggestions for BankID; Section 4 comments on related work; while Sect. 5 concludes the paper.

2 BankID through Adversarial Eyes

A rough sketch of BankID can be drawn after inspecting a white paper released by the BankID project [3]. The system is built around three entities: a central *infrastructure*, *customers*, and online *merchants*. Private keys and the corresponding public-key certificates issued to customers are stored and used by the central infrastructure. This design differs from a typical X.509 PKI, which requires private keys to be solely available to the entity identified in the matching certificate [4]. As a consequence, all customer authentication and digital signature services with PKI credentials are executed by the infrastructure. Merchants control their own cryptographic keys and rely on server software distributed by the BankID project.

A Java applet is central in the authentication procedure. The applet is readily available from the central infrastructure, and is provided to end-users by all affiliated merchants. This makes it a natural target for uncovering technical details about BankID.

2.1 Reverse Engineering

A common technique to understand undocumented software is *reverse engineering* [5]. The information gathered from public written sources was insufficient to understand the inner-workings of BankID protocols involving customers. Hence, we reverse engineered the applet to study the protocols in more detail. The process included studying input and output data, the communication flow, and representing the application as human-readable source code.

By inspecting merchant web pages we discovered that the applet is controlled through HTML parameters. Two parameters specify addresses to the infrastructure server running a two-factor authentication procedure and the merchant server carrying out a challenge-response protocol. Consequently, all merchants can use the same applet by configuring these initialization parameters.

2.2 The Attack

By changing the two parameters to the applet, it willingly communicates—over either HTTP or HTTPS—with the MitM proxy depicted in Fig. 1. We choose HTTP to avoid having to install a certificate on the MitM proxy. The decision to store the customers’ cryptographic keys at the infrastructure results in a complex authentication protocol:

- The customer presents her birth number, one-time password (OTP), and fixed password to the central infrastructure. This action unlocks PKI functionality.
- The customer engages in a challenge-response protocol with the merchant. The infrastructure handles all PKI operations on behalf of the user.

The proxy learns the communication between the applet and the merchant, which is sufficient to obtain an authorized session to the merchant. However, the information flow between the customer and the infrastructure is encrypted, preventing the proxy from obtaining the customer credentials. The attack is carried out through the following steps:

1. Trick the user into visiting a webpage on the proxy, initializing the applet with malicious parameters.
2. Start the HTTPS session between the MitM proxy and the merchant to obtain session IDs.
3. Relay the traffic until the authentication completes.
4. Seize the HTTPS session to the merchant after the authentication is completed.

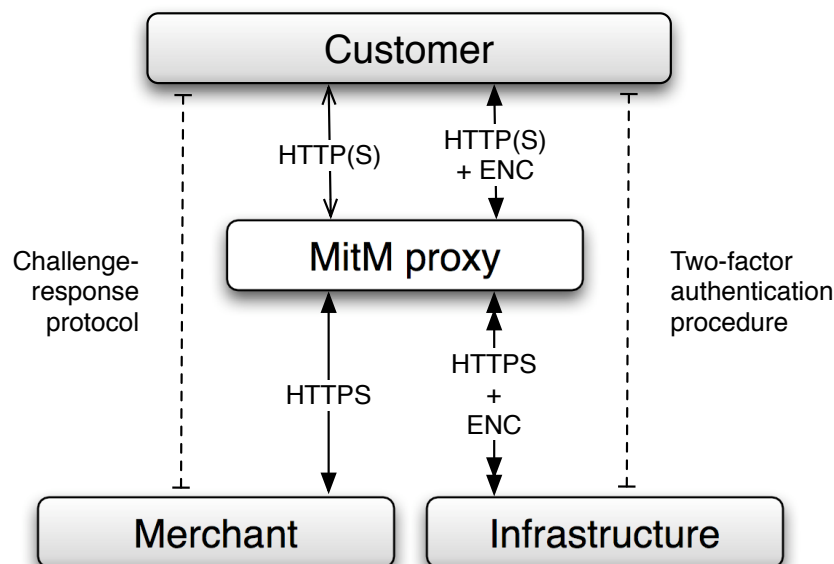


Fig. 1. The MitM proxy in the authentication protocol

Norwegian banks currently use OTPs and fixed passwords to authorize transactions. Therefore, the attacker must collect at least one OTP and the password to transfer money out of the account. This can be achieved by alerting the user at the end of the log-in procedure that the previously entered fixed password and OTP were incorrect, after which the attacker asks for them again.

Proof of Concept. The attack was tested against two randomly chosen Norwegian online banking systems. Both attempts gave access to a customer account in these banks. The vulnerability was first identified and tested in March 2007. The BankID community claims to have fixed the problem in November 2007.

3 Possible BankID Improvements

The MitM attack described herein must be addressed by the BankID community. The applet needs to properly authenticate its communication peers, enabling it to detect a MitM proxy. Also, the applet must require end-to-end encryption when communicating with both the infrastructure and the merchant.

In the long-term, the BankID community should evaluate the implications of moving to a traditional PKI where the clients possess their own credentials. This would improve the strength of the authentication, and yield a simpler design. Of course, such a change comes with a cost. However, a national security infrastructure must fulfill minimum security requirements, including resistance to MitM attacks.

As the system is now gaining serious momentum in Norway, its users need a better perception of the true level of security. In light of our attack, and the findings in [2], a thorough analysis of BankID is called for. The infrastructure and its documentation should be scrutinized by independent security experts to detect and resolve problems. This could increase the trustworthiness of BankID in the long run.

4 Related Work

A series of three articles analyze Norwegian banking systems [6,7,2]. Our attack builds on the above-mentioned article series and zooms in on weaknesses touched upon in the risk analysis of BankID [2]. In particular, our work further testify to the inefficacy of the banks' security-through-secrecy policy.

In [8], Anderson argues that a false threat model was accepted, due to the lack of feedback on why British retail banking systems failed. In doing so, the financial industry developed increasingly complex systems to protect against cryptanalysis and technical attacks, when it would have been wiser to focus on implementation and managerial failures. Analyses of banking systems published after Anderson's initial paper underscore the observation that systems fail not because of inadequate cryptographic primitives, but rather design flaws and implementation errors [9,10].

5 Conclusion

The new national security infrastructure for e-commerce in Norway, BankID, was vulnerable to a MitM attack in 2007. By changing initialization parameters to the BankID applet, an adversary could insert a proxy between a customer and a merchant. When BankID was used in Internet banking, an attacker could let a customer complete authentication, and later take over the banking session. The attack did not depend on malicious software being installed on the victim's computer. Proof of concept code was developed to demonstrate the attack.

The MitM vulnerability in BankID calls for immediate attention. The banks claim to have fixed the problem in November 2007. In the long run, the banks should carefully evaluate their development process, as their current methodology results in software that contradicts advice given in well-known security textbooks.

5.1 Final Remark

We would like to emphasize that *only* BankID accounts belonging to members of the NoWires Research Group were used to develop and demonstrate the MitM attack. No accounts belonging to others were involved in any way during our work with this paper.

References

1. Kent, S.T., Millett, L.I., eds.: IDs—Not That Easy: Questions About Nationwide Identity Systems. The National Academies Press (2002)
2. Hole, K.J., Tjøstheim, T., Moen, V., Netland, L., Espelid, Y., Klingsheim, A.N.: Next generation internet banking in Norway. submitted to IEEE Security & Privacy (2007) Available at: <http://www.nowires.org/Papers-PDF/BankIDevaluation.pdf>.
3. The Norwegian Banks' Payment and Clearing Centre: BankID FOI white paper. (Release 2.0.0) (2006) (in Norwegian).
4. Adams, C., Lloyd, S.: Understanding PKI—Concepts, Standards, and Deployment Considerations. 2nd edn. Addison-Wesley (2003)
5. Chikofsky, E.J., Cross II, J.H.: Reverse engineering and design recovery: A taxonomy. IEEE Software **7**(1) (1990) 13–17
6. Hole, K.J., Moen, V., Tjøstheim, T.: Case study: Online banking security. IEEE Security & Privacy **4**(2) (2006) 14–20
7. Hole, K.J., Moen, V., Klingsheim, A.N., Tande, K.M.: Lessons from the Norwegian ATM system. IEEE Security & Privacy **5**(6) (2007) 25–31
8. Anderson, R.: Why cryptosystems fail. In: ACM 1st Conference on Computer and Communication Security. (1993)
9. Berkman, O., Ostrovsky, O.M.: The unbearable lightness of pin cracking. (http://www.arx.com/documents/The_Unbearable_Lightness_of_PIN_Cracking.pdf)
10. Anderson, R., Bond, M., Clulow, J., Skorobogatov, S.: Cryptographic processors—a survey. Technical Report 641, University of Cambridge (2005) <http://www.cl.cam.ac.uk/~mkb23/research/Survey.pdf>.

**Paper VII:
Robbing Banks with Their Own
Software—an Exploit against Norwegian
Online Banks**

Robbing Banks with Their Own Software—an Exploit against Norwegian Online Banks*

Yngve Espelid, Lars–Helge Netland, André N. Klingsheim, and Kjell J. Hole

Abstract The banking industry in Norway has developed a new security infrastructure for conducting commerce on the Internet. The initiative, called BankID, aims to become a national ID infrastructure supporting services such as authentication and digital signatures for the entire Norwegian population. This paper describes a man-in-the-middle vulnerability in online banking applications using BankID. An exploit has been implemented and successfully run against two randomly chosen online banking systems to demonstrate the seriousness of the attack.

Key words: Public-key infrastructure, man-in-the-middle attack, online banking

1 Introduction

The Norwegian banking community has created a new infrastructure for secure e-commerce, called BankID.¹ As of October 2007, BankID has more than 700,000 users. This number is expected to approach 2.5 million come 2009. At the time of writing, the infrastructure is mainly used for authentication of Internet banking customers, but BankID is extending into other markets, such as the government sector and e-commerce in general. It has also been used in conjunction with e-voting in some companies. BankID won a European prize, namely the *eema Award for Ex-*

Y. Espelid, L-H. Netland, A. N. Klingsheim, and K. J. Hole
NoWires Research Group
Department of Informatics
University of Bergen, Norway
e-mail: {yngvee, larshn, klings, kjellh}@ii.uib.no

* A short version of our work is to be presented at Financial Cryptography and Data Security 2008 (FC '08) [10].

¹ Not to be confused with the Swedish BankID initiative.

cellence in Secure Electronic Business in 2006. Within a few years, the Norwegian banking industry wants BankID to become a nationwide identity system.

No detailed technical information about BankID has been released to the general public. Our request to see in-depth descriptions of the architecture and design was met with a non-disclosure signature prerequisite. Moreover, no publicly available independent third party evaluation of the system confirms that BankID meets a minimum of security and privacy requirements. This is worrisome due to a number of reasons: Firstly, a report by the US National Research Council [20] states that public review is essential when developing a nationwide identity system. The social costs of a poorly thought-out system are simply too high to justify.

Secondly, unlike in the US, liability has historically been assigned to the customer in disputes with Norwegian banks. In a previously analyzed court case [16], two expert witnesses turned the ruling in favor of a bank by claiming that the banking systems were very secure. No technical documentation was provided to support this claim.

Thirdly, the banking industry both owns the BankID infrastructure and provides financial services on top of the framework. It is not clear how potential conflicts of interest, involving the bank as a service provider and operator, will be resolved. Uncontested, the combination of no trusted third party and a security-through-secrecy policy could undermine the legal protection of Norwegian bank customers.

Finally, BankID relies on two-factor authentication with One-Time Passwords (OTPs), similar to earlier online banking systems. In 2005, Schneier warned that this form of authentication failed to address recent security challenges, such as phishing, identity theft, and fraudulent transactions [23].

A previous paper [18] describes a risk analysis of the BankID infrastructure. In that study, the authors give an overview of the architecture and design of BankID, and pinpoint several weaknesses. Our work was done in parallel with the mentioned evaluation, and examines the therein suggested Man-in-the-Middle (MitM) attack in detail. A short paper on our work has been accepted for publication at *Financial Cryptography and Data Security 2008* [10].

The remainder of this paper is organized as follows: Section 2 provides a short overview of BankID; Section 3 looks at BankID from an adversary's point of view; Section 4 describes a MitM vulnerability in BankID that has been turned into an exploit; Section 5 explains how to make the attack more effective by capitalizing on a bank customer's trust in BankID; Section 6 describes our disclosure process; Section 7 suggests solutions to improve BankID; Section 8 presents related work; while Section 9 concludes the paper.

1.1 Definitions

Individual authentication, referred to as *authentication* for simplicity, is the process of establishing an understood level of confidence that an individual is who he or she claims to be [21]. If the level of confidence is high, the authentication is said

to be strong. *Authorization* is the process of deciding what an individual ought to be allowed to do. A *vulnerability* is a weakness that violates the security goals of an information system. An *exploit* is a practical realization of a vulnerability in the form of detailed instructions or program code. The approach must have been implemented and successfully run to constitute an exploit.

2 BankID Overview

BankID is modeled after an X.509 Public-Key Infrastructure (PKI), where the banks themselves own and operate the central infrastructure. PKIs have been studied extensively by the computer security community. In addition, many vendors provide PKI solutions in the commercial marketplace. Hence, there is a strong theoretical foundation for important PKI principles as well as extensive practical experience gained from implementing and running PKIs. A good introduction to PKIs can be found in [1].

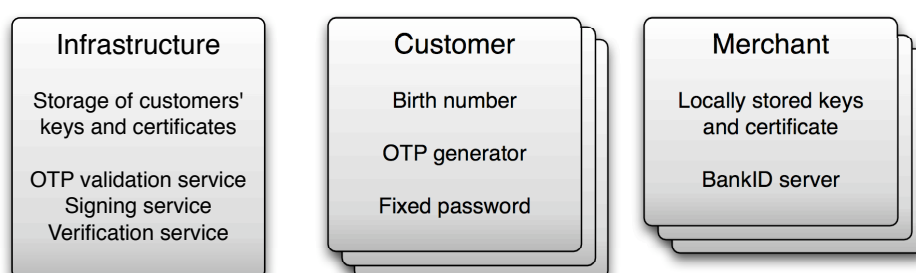


Fig. 1 Entities in BankID

Useful insights into BankID can be obtained from a white paper released by the BankID project [25], and by enrolling as a customer of the PKI. The system is built around three general entities: a central *infrastructure*, *customers*, and online *merchants*. The individual parties, their credentials, and duties are summarized in Fig. 1. In one of the participating banks, new customers sign up for BankID on the Web. Shortly after becoming a member, the customer receives an OTP generator and a fixed password by unregistered mail. When logging into the bank with BankID, customers use their Norwegian birth number² for identification, and an OTP in combination with the fixed password for authentication. People who sign up with more than one bank can choose freely among their OTP tools when using BankID. The PKI functionality provided by BankID is transparent to customers, as their private-public key pairs are stored and controlled by the infrastructure. PKI services, such

² Norwegian birth numbers uniquely identify Norwegian citizens. These are similar to the US Social Security numbers.

as creating and verifying digital signatures, are performed centrally on behalf of the users. Merchants store and control their own cryptographic keys and rely on server software distributed by the BankID project.

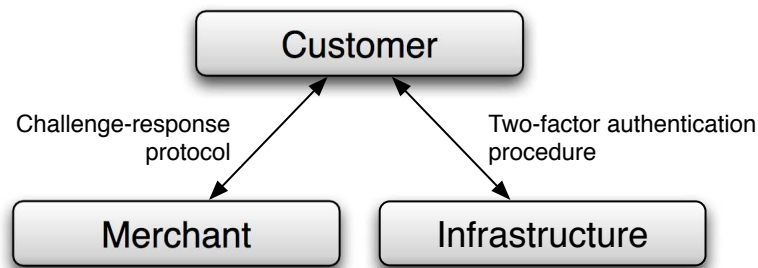


Fig. 2 BankID authentication procedure from the customer's point of view

The BankID design differs from a typical X.509 PKI, which requires private keys to be solely available to the entity identified in the matching public-key certificate [1]. Compared to textbook PKIs, BankID offers lower operational costs as the service providers don't have to roll out and maintain relatively expensive cryptographic hardware, but the design makes it harder to argue convincingly that a given private key can only be accessed by its rightful owner. The decision to store the customers' private and public keys on the infrastructure also results in an untraditional authentication protocol, that appears to be a hybrid of previous Internet banking schemes in Norway and X.509 PKI based authentication. Prior to BankID, authentication typically involved bank clients presenting the three customer credentials given in Fig. 1 to the banking system. The new design, depicted in Fig. 2, involves a longer protocol:

- The customer presents her birth number, OTP, and fixed password to the central infrastructure. This action unlocks PKI functionality on the infrastructure.
- The customer engages in a challenge-response protocol with the merchant. The infrastructure handles all PKI operations on behalf of the user.

A closer look at BankID's architecture and design can be found in [18].

2.1 The BankID Applet

A Java applet [24] is central in the authentication procedure depicted in Fig. 2. The applet is readily available from the central infrastructure.

The following describes a typical BankID session involving bill payment for Internet banking customers. First, the client visits her bank's log-in page, which instructs her browser to download the applet from the central infrastructure. The applet initiates the previously described authentication procedure. Upon successful

authentication, an HTTPS session loads in the customer’s browser. Now, the client can fill out payment details. Upon submitting the bill, the applet is reinitialized in the customer’s browser, prompting her for credentials to sign the transaction. After submitting a fresh OTP and the private-key password, the transaction is processed. Next, the customer can continue the HTTPS session or terminate the bank session by logging out.

3 An Adversarial View into BankID

Attackers use a variety of tactics to break the security of software systems. A common strategy is to start by gathering information about the target. A detailed profile on an application allows attackers to apply their resources in the potentially most rewarding places. Upon mapping out the target, adversaries are wise to consider common vulnerabilities, as studies show that systems often fail for the same reasons. Common techniques used by attackers have been documented at length by the software security community, e.g. [4, 15, 19].

In terms of BankID, an interesting observation is the bold design choice of centralized key storage that contradicts advice given in the security literature. The resulting authentication protocol should draw the attention of attackers, because the development of new secure cryptographic protocols is a difficult undertaking. So much so that security experts strongly discourage the practice of “rolling your own” cryptography [26].

3.1 Reverse Engineering the Authentication Protocol

An inspection of merchant web pages reveals that the BankID applet is initialized by HTML parameters. One parameter specifies the address to the infrastructure server running the two-factor authentication procedure. Another parameter controls the location of the merchant server carrying out the challenge-response protocol. Consequently, all merchants can use the same applet by configuring these initialization parameters.

The applet initialization parameters can be altered so that the applet communicates with BankID software through a proxy controlled by an attacker. This allows adversaries to produce a blueprint of the BankID authentication procedure.

A walk-through of the customer authentication is provided by Fig. 3. The broken vertical lines represent the three main entities’ lifelines. When entities interact, activation bars overlay their lifelines. The customer’s activation bar represents the browser running on her computer. The grey and partially overlapping activation bar symbolizes the applet running in her browser.

Steps 1 through 4 in the figure show how merchants bootstrap the authentication process. The customer downloads an HTML page from the merchant, which

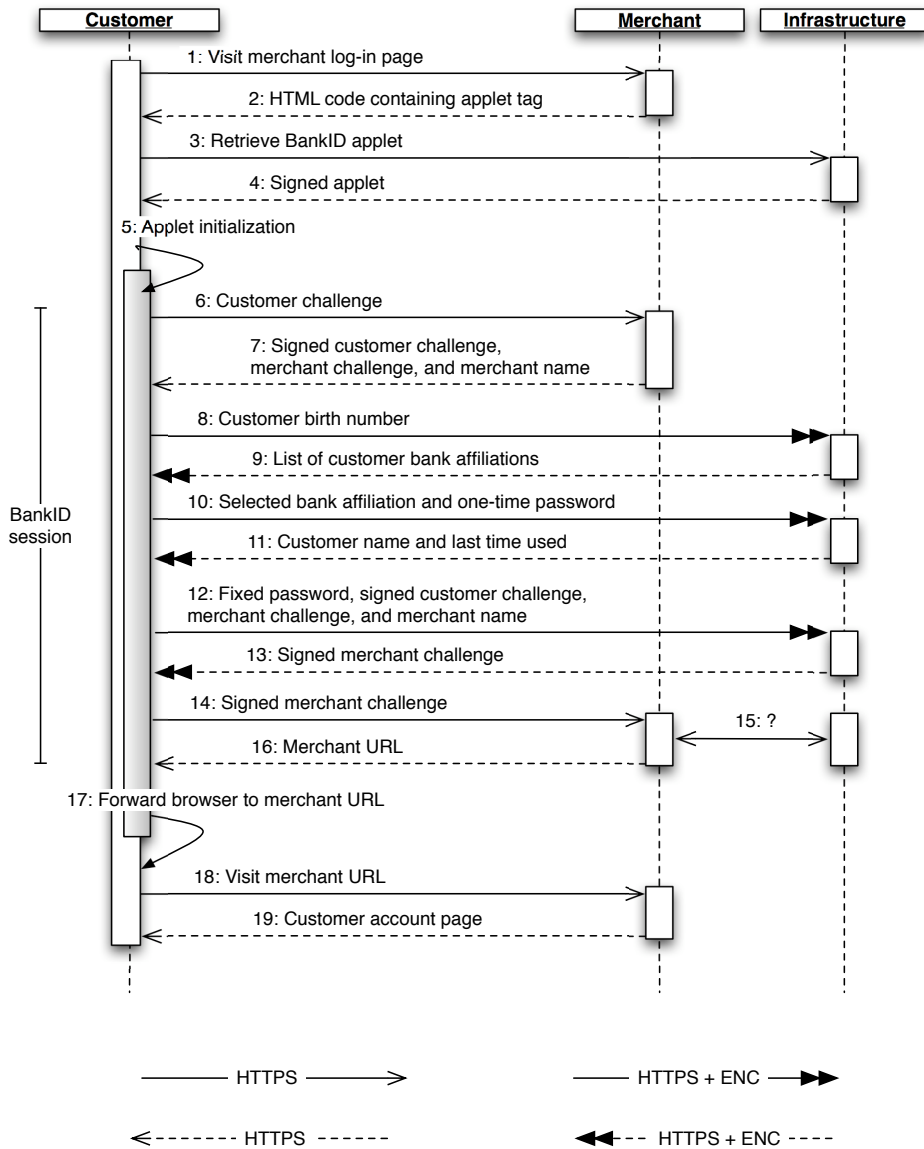


Fig. 3 Authentication process

instructs the customer’s browser to retrieve the signed applet from the central infrastructure. These interactions run over HTTPS. The browser automatically verifies the applet signature, and prompts the user to trust the applet.

On customer acceptance, the applet is initialized in step 5. The applet is the catalyst for the authentication protocol and manages the BankID session, depicted as steps 6 through 16. Two stages, numbered in accordance with Fig. 3, initiate the challenge-response protocol:

6. The applet generates and sends a challenge to the merchant.
7. The merchant signs the customer challenge and generates a challenge to the customer. These challenges, along with the merchant name, are sent to the customer.

Steps 8 through 13 make up the two-factor authentication procedure necessary to unlock the customer's PKI credentials on the central infrastructure and sign the challenge. This communication has an additional layer of encryption, denoted ENC in Fig. 3, which denies the proxy the possibility to determine the content of that part of the protocol. Hence, the following dialogue is partially guesswork based on observations of network activity and the applet's response to customer input:

8. The customer inputs her birth number for identification.
9. The infrastructure returns a list of her BankID affiliations.
10. The customer chooses a bank and enters an OTP.
11. On valid OTP, the infrastructure returns the customer's name and the time when BankID was last used.
12. The signed customer challenge, the merchant challenge to sign, the name of the merchant, and the fixed customer password, are sent to the infrastructure.
13. The infrastructure verifies the merchant signature and uses the customer's private key to sign the merchant challenge, which it returns to the customer.

This concludes our guesswork. The applet now completes the challenge-response protocol with the merchant:

14. The signed merchant challenge is returned to the merchant.
15. We assume that the merchant and the infrastructure communicate to determine the customer's identity and verify the signature.
16. If the challenge-response protocol is successful, the merchant sends a URL to the customer.

Step 16 completes the authentication protocol, and the customer continues an HTTPS session with the merchant as illustrated by steps 17 through 19.

3.2 Reverse Code Engineering

Java byte code is easily reverse engineered and can reveal a program's inner workings to an attacker. When reverse engineering the BankID applet, we found the additional layer of encryption in steps 8–13 in Fig. 3 intriguing and made some interesting observations. As it turns out, three public keys belonging to the infrastructure are hardcoded in the applet. These are linked to different infrastructure services, namely OTP validation, signing, and verification.

In a customer request to the infrastructure, the applet generates a symmetric key used to encrypt the query. This key is encrypted with the service's public key and appended to the request. Using its private key, the service can decrypt the symmetric

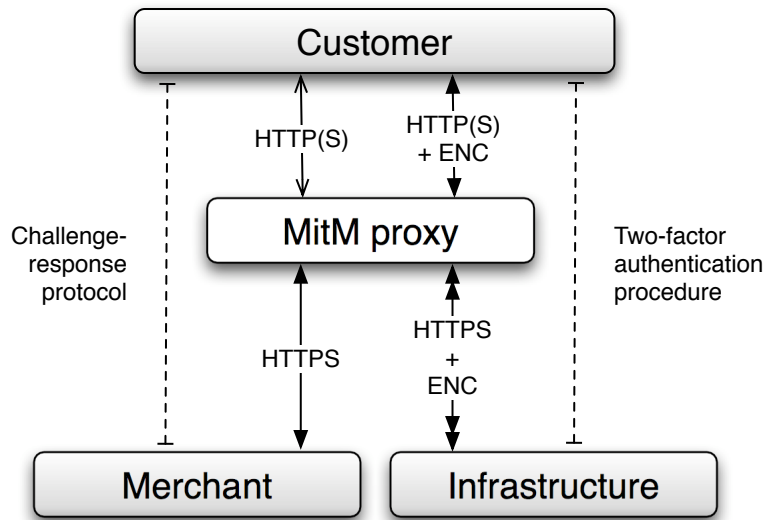


Fig. 4 The MitM proxy in the authentication protocol

key, and in turn decrypt the query. The same symmetric key is used to encrypt the response to the customer. A new key is created for each request.

Another observation from the code study is the use of the core Java class `SecureRandom`. This class provides a pseudo-random number generator. A study of the code segment generating symmetric encryption keys revealed a possible vulnerability. The running Java version affects whether the `SecureRandom` instance seeds itself or by an algorithm in the applet. Given Java version prior to 1.4, current time, amount of free memory, and loop counters are used to compute the seed. A cryptanalysis can determine the strength of both the seeding technique and the encryption algorithm. However, we chose to focus on session management issues in BankID.

4 An Exploit against BankID

As mentioned in Sect. 3, by changing two initialization parameters, the applet will—incommunicate—over either HTTP or HTTPS—with the MitM proxy depicted in Fig. 4. The proxy learns the communication between the applet and the merchant, which is sufficient to obtain an authorized session to the merchant. The attack is carried out through the following steps:

1. Trick the user into initializing the applet with malicious parameters.
2. Start the HTTPS session between the MitM proxy and the merchant to obtain a session ID (this identifier is not shown in Fig. 3.)
3. Relay the BankID session until the authentication completes.

4. Seize the HTTPS session to the merchant after the authentication is completed (step 16, Fig. 3.)

The attack can be explained in terms of session management. Conceptually, two sessions exist; a regular HTTPS session between the customer and the merchant, and the BankID session involving the infrastructure, the merchant, and the customer, as shown in Fig. 3. A discrepancy between these sessions enables the MitM attack. First, only the merchant server is authenticated in the HTTPS session, enabling the MitM proxy to initiate a session on behalf of the customer. Then the customer and the merchant are mutually authenticated through the BankID session, which is simply relayed by the MitM proxy. Finally, the authorization granted to the customer in the BankID session is transferred to the HTTPS session controlled by the MitM proxy, and the attack is successful (step 16, Fig. 3).

Note that the attack uses the signed applet from the infrastructure, turning it into an attack tool against BankID.

4.1 Proof of Concept

The previously described vulnerability was turned into an exploit against two randomly chosen Norwegian online banking systems in March 2007. Both attempts gave access to a customer account in these banks. The BankID community claimed to have fixed the problem in November 2007. A slightly modified version of the first exploit was successfully launched against BankID again in December 2007, using a version rollback attack. In short, an old version of the BankID applet was used to sidestep the countermeasures implemented in November 2007. The BankID community then introduced additional security measures in January 2008, thereby stopping our rollback attack.

5 Attack Considerations

The MitM attack can be bootstrapped in multiple ways, using well known attack strategies. Phishing attacks are already plaguing the banking industry, and can be used to trick some users into opening a webpage from the MitM proxy.

Nordic banks have been pestered over the last year by man-in-the-browser attacks [13]—trojan horses installed in web browsers. A trojan could change the parameters to the applet when the customer visits her online bank. This would be extremely difficult to detect for the average user.

5.1 Trust Management Capitalization

Trust can be defined as a positive expectation regarding the behavior of someone or something in a situation that entails risk to the trusting party [9, p. 77]. Risk is simply the possibility of suffering harm or loss. It is important to note that, unlike in the X.509 PKI specification [1], trust in this context is not a binary concept but involves levels of trust. For any given user, there is a certain amount of trust that is needed to be willing to transact. Let this level be denoted the *cooperation threshold* [22]. In order to get the most out of the attack against BankID, an adversary wants as many customers as possible to reach the cooperation threshold. The BankID design helps achieve this goal.

Assume that an attacker decides to bootstrap the BankID attack with an e-mail phishing scheme. A press release from Gartner indicates that approximately 19% of phishing targets click on a link in a malicious e-mail, and that 3% give financial or personal information to phishers [11]. The numbers illustrate that the success rate relies firstly on the cleverness of the phishing e-mail, and secondly on the trustworthiness of the phishing site. We discuss our attack in conjunction with the latter.

Recall that the signed BankID applet is loaded unmodified from the BankID infrastructure. Hence, the user carries out a seemingly regular authentication procedure. The browser successfully validates the applet signature and displays a certificate belonging to the BankID infrastructure. This is ideal in winning the trust of customers, as they are carefully instructed to look for this when using BankID [5]. Furthermore, the user is presented with this information before the webpage is rendered. The user's attention is drawn to the applet—not to the webpage or the MitM proxy's URL in the address bar. Hence, the important first step towards the cooperation threshold is taken before the malicious webpage is shown to the user.

The next step is to present a webpage visually indistinguishable from the merchant's authentic webpage. The only indication of an attack will then be in the address bar of the victim's browser. Phishers use a variety of tactics to manipulate this bar. If a merchant website has a cross-site scripting vulnerability [19], the success rate of the attack could rise further by capitalizing on the customer's trust in the merchant's own website. Upon completing the authentication, the attacker assumes control of the real BankID session. The information sent to the customer after this point can be crafted so that he still believes that the attack was a legitimate log-in attempt.

Norwegian banks currently use OTPs and fixed passwords to authorize transactions. Therefore, the attacker must collect at least one OTP and the password to transfer money out of the account. This can be achieved by alerting the user at the end of the log-in procedure that the previously entered fixed password and OTP were incorrect, after which the attacker asks for them again. Upon receiving the credentials, the attacker sends the customer one of the bank's standard error messages. This last step is a standard phishing technique. However, the customer has already reached the cooperation threshold and should take the bait.

6 The Disclosure Process

The discovery of the BankID MitM vulnerability and the subsequent exploit urgently called for countermeasures from the BankID community. Building on insights from the BankID risk analysis [18], our team needed less than a month to break into Norwegian Internet banks using BankID. Taking into account that the attack relies on techniques well-known to malicious hackers, it was reasonable to conclude that our attack posed a significant risk for BankID customers.

According to a survey [7], the world's largest software companies encourage some variant of *responsible disclosure* [8] when independent researchers find vulnerabilities in their products. Inspired by responsible disclosure, we informed major stakeholders in the BankID community, namely Bankenes Standardiseringskontor (BSK) and The Norwegian Banks Payment and Clearing Centre (BBS), about the MitM vulnerability in March '07. The Financial Supervisory Authority of Norway (FSAN) was also informed about the problem at this point. On request, BSK also received a technical description of how the BankID vulnerability could be turned into an exploit. They later responded that the vulnerability had been removed in January, i.e. before we developed the proof of concept code.

Unable to convince the system owners about the dangers posed by the exploit, we released the interim report "Next Generation Internet Banking in Norway" on May 16th [18]. This work points out weaknesses in BankID, and explicitly states that we had developed a proof of concept attack against BankID. Our discovery was reported by several media outlets, but did not spark a broad discussion around BankID as a nationwide identity system candidate.

In September and October '07 we demonstrated the MitM exploit for FSAN and a group of security experts with influence on the Norwegian banking industry, hoping that this would get the attention of the BankID community. A month later we distributed an early version of this paper to BSK, BBS, and the BankID coordinator. In November '07 we again told of the exploit in a large Norwegian newspaper [12]. In the subsequent debate, the banks claimed to have addressed our attack in a November patch, and questioned the lawfulness of our security testing. In early December '07 our attack caught the attention of members of the Norwegian Parliament, and was scheduled for a Question Time session.

By using a version rollback attack, the exploit was again successfully run against the previously vulnerable Internet banks on December 18th. Hence, the BankID community had spent eight months coming up with a fix that did not work. During this time period the banks neither sought our counsel nor asked for another test. Having failed to establish a productive dialogue with the system owners, we decided to look at other alternatives for improving the situation.

Full disclosure of an exploit on a live banking system with close to one million users seemed drastic. Still, banking customers had for at least nine months believed BankID to be secure, while our exploit showed that the user community had been and continued to be exposed to an unnecessary high risk. In the end, we chose to revisit responsible disclosure, but this time with FSAN as a coordinator. In January

'08 we were informed that new countermeasures had been introduced in BankID to prevent our rollback attack. We have not analyzed these security measures in detail.

7 Possible BankID Improvements

Due to lack of complete information on the BankID system we can only give some general recommendations on how to improve the security through changes to the applet itself. In future versions of the BankID applet, the session discrepancy discussed in Sect. 4 should be corrected to mitigate the risk posed by our exploit. The applet needs to properly authenticate its communication peers, enabling it to detect a MitM proxy. Also, the applet must require end-to-end encryption when communicating with both the infrastructure and the merchant. To achieve these goals the applet can require HTTPS when connecting to the infrastructure and the merchant, and explicitly check that the authenticated server is the correct one. Input validation [19], such as a whitelisting approach, can be useful to avoid hostile communication points. We leave it to the BankID community to evaluate the feasibility of this approach.

In the long-term, the BankID community should evaluate the implications of moving to a traditional PKI where the clients possess their own private-public key pairs. The move would improve the strength of the authentication, and yield a simpler design. Also, several of the problems identified in [18] could be reduced or solved. Such a change comes with a cost. However, a national security infrastructure must fulfill minimum security requirements, including resistance to MitM attacks, and offer a strong authentication. Many countries around the world have already put to use, or are contemplating national identity systems based on PKI and cryptographic smartcards. Hence, there are many experiences around the world that should be taken into consideration if the BankID community decides to offer a traditional PKI.

8 Related Work

A series of three articles analyze Norwegian banking systems [17, 16, 18]. The first paper shows that some Norwegian banks were vulnerable to a combined brute-force and distributed denial-of-service attack in 2003 and 2004. The authors go on to discuss the effects of the banks' security-through-secrecy policy, concluding that it prohibits learning and causes the same mistakes to be repeated. The second paper elaborates on the problems with a bank's non-disclosure policy in a Norwegian court case. The third paper contains a risk analysis of BankID from the customer's point of view. It concludes that users of BankID are exposed to a number of significant risks that should be mitigated. Our attack builds on the above-mentioned article series and zooms in on weaknesses touched upon in the risk analysis of BankID [18]. In

particular, our work further testify to the inefficacy of the banks' security-through-secrecy policy.

According to an IBM white paper [14], the two-factor authentication schemes widely adopted by online banks are insufficient in protecting against combined phishing and MitM attacks. An adversary first sends a phishing e-mail to the banking customer with a link to a proxy controlled by the attacker. If the victim takes the bait, the adversary plays an active role in the log-in process by relaying user input to the bank and the bank's responses back to the user. Upon completing authentication, the attacker can seize the session or mix fraudulent transactions with the users legitimate transactions.

Our attack uses elements of a combined phishing and MitM attack, but goes further by using the bank's own software, the BankID applet, to gain the victim's trust. The BankID attack starts as a phishing attack with a phishing e-mail to the bank customer, but continues with loading the unmodified and digitally signed BankID applet instead of a fake applet. By doing so, an adversary abuses a crucial point of trust in BankID. The unmodifiable applet, formerly a disadvantage to an attacker, becomes an advantage in terms of gaining the trust of banking customers. After finishing the applet log-in procedure, the attack proceeds as in the combined phishing and MitM attack, where the attacker must trick the victim into supplying his OTP and fixed password.

In [2], Anderson argues that a false threat model was accepted, due to the lack of feedback on why British retail banking systems failed. In doing so, the financial industry developed increasingly complex systems to protect against cryptanalysis and technical attacks, when it would have been wiser to focus on implementation and managerial failures. Analyses of banking systems published after Anderson's initial paper underscore the observation that systems fail not because of inadequate cryptographic primitives, but rather design flaws and implementation errors [6, 3].

9 Conclusion

The security of the Norwegian banking industry's new PKI solution, BankID, was repeatedly broken in '07. A MitM attack enabled attackers to access customer accounts in two online banks. Our attack used techniques well-known to cyber criminals and was based solely on public information. An exploit was demonstrated for FSAN and a group of security professionals to highlight the severity of the problem.

BankID's untraditional design hinders the system from providing a high level of security. The decision to store customers' private-public key pairs in a central location has resulted in a weak authentication protocol. A redesign of BankID is called for if the system is to offer the intended degree of security.

Our exploit underscores the importance of independent evaluation of national systems. BankID's design flaw contradicts advice given by security experts, and should have been detected and resolved long before the system was put into pro-

duction. An unbiased scrutinization of the infrastructure and its documentation by leading security analysts would most likely have identified the problem.

The BankID community needs to improve their risk management processes. Today, the system owners fail to identify and quickly resolve problems. This was demonstrated to us by the nine months it took the banks to address our initial exploit. The subsequent version rollback attack further testifies to the inefficacy of BankID's current risk management processes.

As BankID is now gaining serious momentum in Norway—and is pushed by the BankID community to become the main identity system in Norway—both government and citizens need a better perception of the true level of security. In light of our attacks and the findings in [18], a thorough analysis of BankID is called for. This could increase the trustworthiness of BankID in the long run. At the time of writing the gap is too big between the actual level of security, and how the BankID community describes their system (see www.bankid.no.)

9.1 Final Remark

We would like to emphasize that only BankID accounts belonging to members of the NoWires Research Group were used to develop and demonstrate the MitM attack. No accounts belonging to others were involved in any way during our work with this paper.

References

1. Adams, C., Lloyd, S.: Understanding PKI—Concepts, Standards, and Deployment Considerations, 2nd edn. Addison-Wesley (2003)
2. Anderson, R.: Why cryptosystems fail. In: ACM 1st Conference on Computer and Communication Security. Fairfax, VA, USA (1993)
3. Anderson, R., Bond, M., Clulow, J., Skorobogatov, S.: Cryptographic processors—a survey. Technical Report 641, University of Cambridge (2005). URL <http://www.cl.cam.ac.uk/~mkb23/research/Survey.pdf>
4. Andrews, M., Whittaker, J.A.: How to Break Web Software—Functional and Security Testing of Web Applications and Web Services. Addison-Wesley (2006)
5. BankID: Hva gjør kunden ved mistanke om at noe er galt? (2007). URL <http://www.bankid.no/index.db2?id=4066>. Last checked Jan. 2008 (in Norwegian)
6. Berkman, O., Ostrovsky, O.M.: The unbearable lightness of pin cracking (2006). URL http://www.arx.com/documents/The_Unbearable_Lightness_of_PIN_Cracking.pdf. Last checked Jan. 2008
7. Biancuzzi, F.: Disclosure Survey (2006). URL <http://www.securityfocus.com/columnists/415>. Last checked Jan. 2008
8. Christey, S., Wysopal, C.: Responsible vulnerability disclosure process (2002). URL http://www.whitehats.ca/main/about_us/policies/draft-christey-wysopal-vuln-disclosure-00.txt. Last checked Jan. 2008

9. Cranor, L.F., Garfinkel, S. (eds.): *Security and Usability—Designing Secure Systems That People Can Use*. O'Reilly (2005)
10. Espelid, Y., Netland, L.H., Klingsheim, A., Hole, K.J.: A proof of concept attack against norwegian internet banking systems. In: *Financial Cryptography and Data Security (FC)*. Cozumel, Mexico (2008)
11. Gartner: Gartner study finds significant increase in e-mail phishing attacks (2004). URL http://www.gartner.com/press_releases/asset_71087_11.html. Last checked Jan. 2008
12. Gjøsteen, K., Hole, K.J.: Nei, ennå ikke trygg. *Aftenposten* (29. Nov, 2007). URL <http://www.aftenposten.no/meninger/debatt/article2126133.ece>. Last checked Jan. 2008 (in Norwegian)
13. Gühring, P.: Concepts against man-in-the-browser attacks (2006). URL <http://www2.futureware.at/svn/sourcerer/CAcert/SecureClient.pdf>. Last checked Jan. 2008
14. Gundel, T.: Phishing and internet banking security (2005). URL ftp://ftp.software.ibm.com/software/tivoli/whitepapers/Phishing_and_Internet_Banking_Security.pdf
15. Hoglund, G., McGraw, G.: *Exploiting Software—How to Break Code*. Addison-Wesley (2004)
16. Hole, K.J., Moen, V., Klingsheim, A.N., Tande, K.M.: Lessons from the Norwegian ATM system. *IEEE Security & Privacy* **5**(6), 25–31 (2007)
17. Hole, K.J., Moen, V., Tjøstheim, T.: Case study: Online banking security. *IEEE Security & Privacy* **4**(2), 14–20 (2006)
18. Hole, K.J., Tjøstheim, T., Moen, V., Netland, L., Espelid, Y., Klingsheim, A.N.: Next generation internet banking in Norway. submitted to *IEEE Security & Privacy* (2007). URL <http://www.nowires.org/Papers-PDF/BankIDevaluation.pdf>
19. Huseby, S.H.: *Innocent Code*. Wiley (2004)
20. Kent, S.T., Millett, L.I. (eds.): *IDs—Not That Easy: Questions About Nationwide Identity Systems*. The National Academies Press (2002)
21. Kent, S.T., Millett, L.I. (eds.): *Who Goes There? Authentication Through the Lens of Privacy*. The National Academies Press (2003)
22. Marsh, S., Dibben, M.R.: Trust, untrust, distrust and mistrust—an exploration of the dark(er) side. In: *iTrust 2005, LNCS*, vol. 3477, pp. 17–33. Springer (2005)
23. Schneier, B.: Two-factor authentication: too little, too late. *Communications of the ACM* **48**(4), 136 (2005)
24. Sun Microsystems, Inc.: Applets. URL <http://java.sun.com/applets/>. Last checked Jan. 2008
25. The Norwegian Banks' Payment and Clearing Centre: BankID FOI white paper (Release 2.0.0) (2006). (in Norwegian)
26. Viega, J., McGraw, G.: *Building Secure Software—How to Avoid Security Problems the Right Way*. Addison-Wesley (2002)