Computing Science Group

On the Security of Internet Banking in South Korea

Hyoungshick Kim
Computer Laboratory
University of Cambridge
hk331@cl.cam.ac.uk

Jun Ho Huh
Computing Laboratory
University of Oxford
jun.ho.huh@comlab.ox.ac.uk

Ross Anderson
Computer Laboratory
University of Cambridge
ross.anderson@cl.cam.ac.uk

CS-RR-10-01

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD

On the Security of Internet Banking in South Korea

Hyoungshick Kim   Jun Ho Huh   Ross Anderson

Abstract

South Korean Internet banking systems have a unique way of enforcing security controls. Users are obliged to install proprietary security software – typically an ActiveX plugin that implements a bundle of protection mechanisms in the user's browser. The banks and their software suppliers claim that this provides trustworthy user platforms. One side-effect is that almost everyone in Korea uses IE rather than other browsers.

We conducted a survey of bank customers who use both Korean and other banking services, and found that the Korean banks' proprietary mechanisms impose significant usability penalties. Usability here is strongly correlated with compatability: Korean users have become stuck in an isolated backwater, and have not benefited from all the advances in mainstream browser and security technology. The proprietary mechanisms fail to provide a trustworthy platform; what's more, alternative strategies based on trustworthy computing techniques are quite likely to suffer from the same usability problems. We conclude that transaction authentication may be the least bad of the available options.

1 Introduction

The Internet has changed the way people bank. Banks have actively promoted online services to save costs and create new business opportunities; their customers benefit from being able to pay bills and transfer funds without having to go to a bank branch. Koreans have been particularly enthusiastic about online banking: the Bank of Korea reported that 57.29 (54.30 personal and 2.99 corporation) million online accounts had been registered as of September 2009 [3]. This report also showed that the number of Internet banking transactions per day, on average, is 29.03 million and the amount of money being transferred is 30.17 trillion Korean Won.[1]

One curious fact, though, is that Internet banking systems in Korea only support Microsoft Internet Explorer (and only the Windows version). Bank customers cannot use other web browsers like Safari, Firefox, Opera and Chrome. In consequence, Internet Explorer has become the dominant web browser in Korea. Recent market share trends for web browsers in different parts of the world [2] illustrate the near-monopoly of Internet Explorer in Korea (see Figure 1).

The reason is simple enough. In order to use Korean Internet banking services, the users are required to install external security plugins. As Internet Explorer is the dominant browser, the developers implemented these plugins as ActiveX controls [27], effectively locking the users in to Internet Explorer. To illustrate the ActiveX dependency of Korean web sites, we analyse the relative traffic around the keyword "ActiveX" using Google trends[2] (see Figure 2). This figure shows the relative traffic of top 10 countries around "ActiveX" from 2008 to 2009.

These plugins, however, are not included in the web standards developed by the World Wide Web Consortium (W3C)[3], and have severe compatibility and usability issues.

[1]about USD 26 billion
[2]`http://www.google.com/trends/`
[3]`http://www.w3.org/`

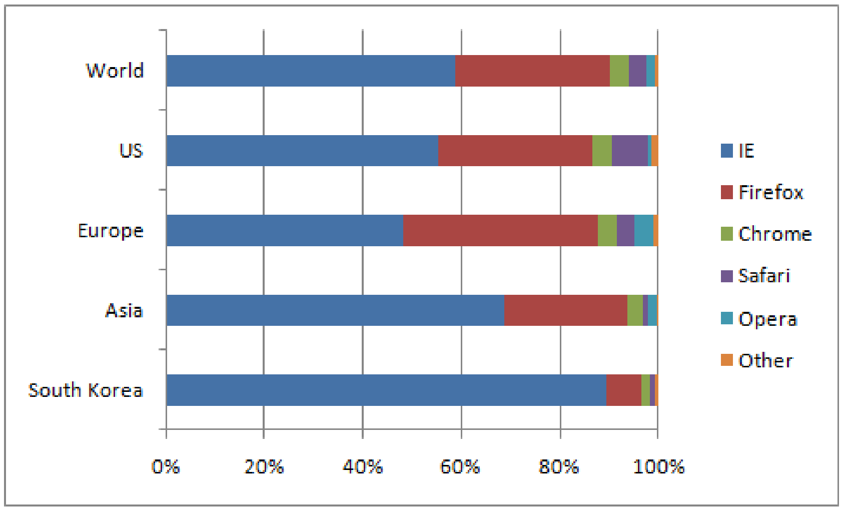


Figure 1: Web Browser Market Share by Geographic Regions (October 2009)

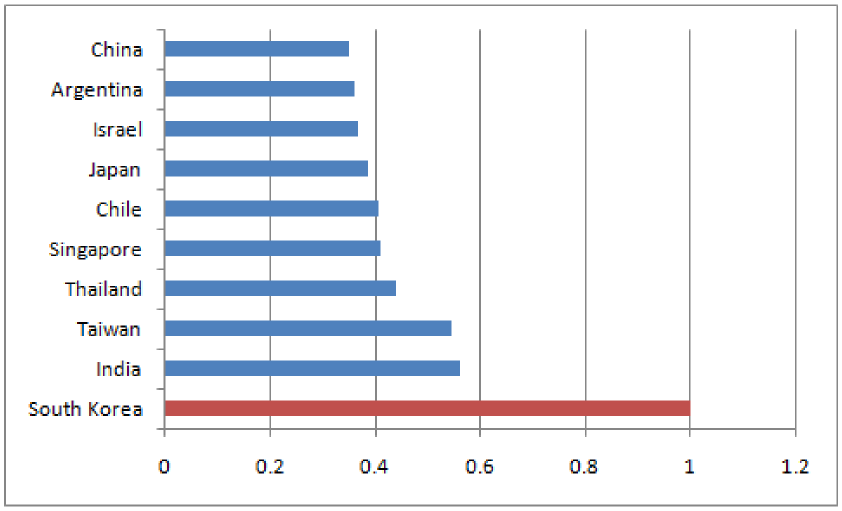


Figure 2: The Relative Traffic of Top 10 Countries Around "ActiveX" From 2008 to 2009

The majority of the Internet banking systems were developed by three Korean software companies: SOFTFORUM[4], INITECH[5] and Ahnlab[6]. Until recently, these companies developed systems that are only compatible with Windows[7].

Much debate [21, 1] has raged around the requirement that users install external security plugins, the lack of effort by the vendor companies to provide better compatibility with other operating systems and browsers, and the usability implications for bank customers. In 2007, a non-profit organization called "OpenWeb"[8] sued the Korea Financial Telecommunications and Clearings Institute (KFTC)[9] (a Korean government agency) for 415 million Korean Won for its

[4]`http://www.softforum.com/`

[5]`http://www.initech.com/`

[6]`http://global.ahnlab.com/`

[7]"Shinhan Bank" and "Korea Exchange Bank" now support Mac OS X through proprietary applications.

[8]`http://openweb.or.kr/`

[9]`http://www.kftc.or.kr/html/english/index.html`

implicit and anti-competitive endorsement of Microsoft products. They lost the case in 2009; we believe one of the main reasons for their loss was the lack of a security and usability evaluation for current Internet banking systems.

This paper aims to contribute: (1) an analysis of the strengths and weaknesses of the proprietary Korean security mechanisms compared with standard technologies, (2) a study of the usability issues raised by employing these proprietary mechanisms, and (3) recommendations for improving overall security as well as usability.

This paper is structured as follows. In Section 2 we describe how security is implemented and distributed in current Korean Internet banking systems. We evaluate these security mechanisms in Section 3 and discuss the usability implications in Section 4. We then recommend several enhancement strategies in Section 5. Finally, in Section 6 we discuss remaining work and conclude.

2 Security Mechanisms Used in S. Korea

There are four generally accepted security properties [26] that one may require when establishing a secure channel between the user and the bank:

- *user/server authentication* – before sending sensitive information over the Internet, the user should be assured that they are communicating with the right bank; the bank should also be able to verify the identity of the user before processing the requested transactions.
- *confidentiality* – only the authorized entities (i.e. the user and the bank) should have access to the content of the messages being exchanged.
- *data integrity* – the user and the bank should be able to detect any manipulation (including insertion, deletion and substitution) or replay of data by unauthorized parties.
- *non-repudiation* – neither the user nor the bank should be able to deny previous commitments or actions; for instance, in case of disputes, the bank should be able to prove to a third party that the user has performed certain transactions.

In Korea, there are additional requirements for the provision of *trusted platforms*. A trusted platform should detect and remove all malware, and prevent malware from reading the users' private banking information. We further define this term in Section 5.3. Although some banks from other countries also feel strongly about this requirement, and encourage their customers to install anti-virus software for instance, the customers are never forced to install anything [25].

Table 1 summarizes the security mechanisms being used in the Korean banking systems to fulfil these requirements. We also list the mechanisms that are commonly used in some banks in the UK and the US for comparison.

In most other countries, Secure Sockets Layer/Transport Layer Security (SSL/TLS) [14] is the de facto Internet banking standard for ensuring confidentiality and data integrity. The Korean banking systems, however, use proprietary protocols based on RSA, HMAC, and a block cipher called SEED (see Section 2.1). Some combination of ID, password and one-time passwords (OTPs) are commonly used worldwide for authentication [20]. Again, the Korean systems are unique in that they also use RSA to provide extra assurance. In fact, RSA is also used for non-repudiation, a property that is rarely found in other countries [11].

Looking at this table, it might seem at first glance that the Korean banking systems provide much stronger security than those in the UK and the US. Certainly, the feature count looks more impressive. But the extra mandatory mechanisms involve ActiveX plugins which the users are obliged to install. The rest of the paper studies these mechanisms in detail, and explains why – despite the hassle they impose on users – they do not add much security.

Table 1: Security Mechanisms for Online Banking

Requirements	All Korean banks	UK bank A	UK bank B	US bank C
Server authentication	proprietary –	SSL/TLS –	SSL/TLS –	SSL/TLS personal indicator [44]
User authentication	ID/password OTP private key (SW)	ID/password OTP –	ID/password – secret key (HW)	ID/password OTP –
Data integrity	proprietary	SSL/TLS	SSL/TLS	SSL/TLS
Non-repudiation	digital signature	–	digital signature	–
Confidentiality	proprietary	SSL/TLS	SSL/TLS	SSL/TLS
Malware detection	anti-virus	anti-virus [O]	anti-virus [O]	anti-virus [O]
Network access control	firewall	firewall [O]	firewall [O]	firewall [O]
Anti-keylogger	keystroke enc.	keystroke enc. [O]	–	–

([O] indicates that the feature is optional.)

2.1 Secure and Authenticated Communication Channel

In 1999, the Korean government launched an Internet banking system based on a Public Key Infrastructure (PKI). This was adopted rapidly by the banks; by the end of 2000, 20 of them were offering Internet banking services based on this PKI [10].

A user obtains a digital certificate through a licensed Certificate Authority (e.g. the Korea Information Security Agency (KISA)[10]) if they pass an online authentication test. The process is managed through proprietary software downloaded from a bank's website. The issued certificate is stored either on the user's hard disk (e.g. `C:\Program Files\NPKI`) or in an external device such as a USB stick. Every transaction is authorized by validating the user's certificate with the CA's public key.

A secure authenticated channel (SAC) is established between the user and the bank server by exchanging digital certificates. The protocols used to generate session keys are not published. We speculate that they include a digital signature mechanism that may be inspired by Secure Electronic Transaction (SET) [37], and an SSL variant using SEED [24] (see Appendix A for details). SEED is a 128-bit symmetric key block cipher developed by the KISA in 1998, with a 16-round Feistel structure. It was approved as a standard block cipher by the Internet Engineering Task Force (IETF) [24] and an ISO/IEC international standard. Since SEED and protocols that use it are not supported by mainstream web browsers – including Internet Explorer, Safari, Firefox, Opera and Chrome – an external plugin is required (see Figure 3).

It may seem strange to use plugins when SSL/TLS is already available and supported by mainstream web browsers. In fact, this is a by-product of the "Crypto wars" in the 1990s during which the US government tried to restrict strong cryptography.

When Internet banking systems were first being deployed in Korea, the encryption algorithms supported by the web browsers distributed outside the US appeared had key sizes limited by the US government to 40, or later 56, bits for symmetric encryption algorithms (RC4/DES).[11] Thus the Korean government funded the development of a new block cipher, SEED, as the country's proprietary standard for secure e-commerce. Ever since, SEED has been deployed as the standard for the Internet banking systems, responsible for encrypting sensitive financial transactions.

[10]`http://www.kisa.or.kr/maine.jsp/`

[11]Blaze and others had already argued convincingly in 1996 that products using 40- or 56-bit ciphers did not give sufficient security for business applications [7].

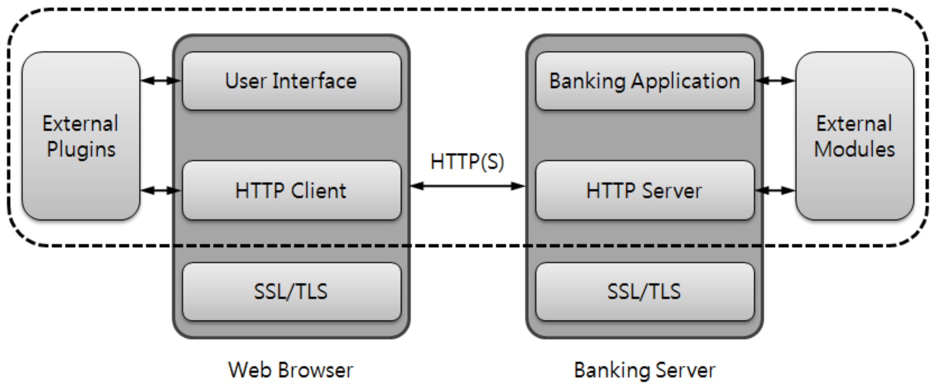


Figure 3: The SAC Enabled Through External Plugins

2.2 User Authentication

We now look at how the users are authenticated by bank servers. A wide range of technologies are currently used for authentication, including passwords, Personal Identification Numbers (PINs), digital certificates (PKI), physical tokens such as smart cards, One-Time Password (OTP) generators, transaction profile scripts, and biometric identification [20].

Korean systems use some combination of two or three of these techniques, based on the belief that this approach offers stronger security than relying on just one method. A commonly deployed authentication process runs as follows:

1. A customer firstly logs into the website using their user ID and password.
2. To carry out a banking transaction, some digits from an indexed Transaction Authentication Number (iTAN) [33] printed on the user's private security card or an OTP (generated by an OTP generator) are entered.
3. Online transaction records are digitally signed with the user's secret key stored in the user's PC or external memory.

Step 2 may involve the use of an "out-of-band" channel, such as postal delivery of a physical token, callback (voice) verification, e-mail approval or notification, and a mobile-phone-based challenge/response process.

A combined authentication mechanism, if carefully designed and implemented, should provide reasonable security. However the designer has to worry about man-in-the-middle attacks, social engineering and malware. The use of digital certificates, for instance, seems reliable in theory, but without strong protection for the user's private keys, it may buy less than you think. To mitigate key-stealing attacks, password-based encryption for the private keys has been legislated: an encryption key, derived from the customer's password alone, is used to protect the private keys. The KISA proposed a specification in 2004 – derived from the Public Key Cryptography Standard (PKCS) #5 [34] – where "SEED" is the standard encryption algorithm used. Triple DES was added as another standard encryption algorithm in 2007.

2.3 Trustworthy User Platforms

The user is the ultimate "client" of the system, not the web browser. The user connects to the bank server through the interfaces available on the web browser. The browser collects the user input, makes requests to the bank server to perform transactions, receives the server

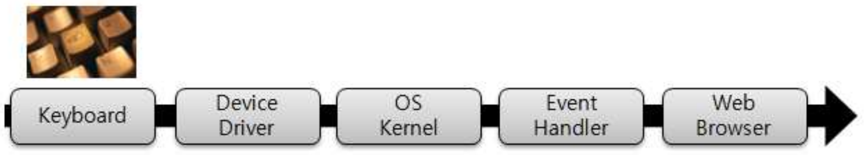


Figure 4: The Communication Channel Between the User and the Browser

response/data, and displays the output to the user. A trusted user platform, therefore, must secure this communication channel (see Figure 4) between the user and the browser.

This communication channel is frequently attacked by malware that tries to steal sensitive information, such as credit card details and other banking credentials. Some attacks are technical, involving zero-day exploits or other engineering tricks that install malware regardless of anything the user can do; others use social engineering tricks. For example, often websites often ask users to download a special codec, which actually turns out to be malware [32].

To mitigate such attacks, the Korean banks oblige users to install further security plugins when they users first access the banking service.

- An anti-virus program is installed to provide real-time protection against known malware. It detects malware by going through the files on the customer's disk and removing all the files that match the signatures in a blacklist. Therefore, it relies on timely updates and on the integrity of the blacklist. The Korean banks require the blacklist to be updated in real time.

- A personal firewall is designed to prevent the malware communicating with an external adversary. It monitors all incoming and outgoing traffic and permits only authorized connections while the user is doing banking. Its goal is to prevent man-in-the-middle or man-in-the-browser attacks being executed by malware.

- A keystroke encryption routine protects sensitive information from the moment it is typed into the keyboard until it reaches the browser (see Figure 4). This is intended to prevent the information from being read or tampered with en route to the browser.

3 Why are These Security Mechanisms Not So Effective?

At first glance, the Korean Internet banking systems seem more secure than the systems used in Britain or the USA (see Table 1). Certainly the feature count is higher: more security mechanisms have been sold to the banks and are distributed by them. In this section, we identify the fundamental problems with these mechanisms and show why they only offer a modest improvement.

3.1 No Protection Against Phishing Attacks

Phishing attacks involve masquerading as a bank and deceiving the users into disclosing their account details [17]. The users may be asked to update or re-enter their confidential information through fake web pages. These attacks have become sophisticated and it's now difficult for users to distinguish fake websites from real ones.

To avoid phishing attacks, users need to check whether the email sender or website owner is a trusted entity. Server authentication at the protocol level is not sufficient, as phishermen can also buy SSL certificates. The client software, installed on the user's machine, cannot easily decide whether the website represents the trusted entity the user intends to communicate with.

One possible approach is to deliver the server verification results to the users visually. Modern web browsers already provide security indicators or warning messages based on blacklists and extended-validation SSL certificates, identifying fake sites and presenting risk information visually. However, this feature cannot be used by Korean customers since SSL/TLS is not part of the Korean system. For the time being, the best one might do is to try to educate users about these attacks and the associated risks – but that has been found in Europe and the USA to not work very well; in general "blame and train" is not a good way to fix usability problems. It is preferable to make the system more usable.

3.2 Problems with Digital Certificates

In theory, digital certificates may provide a neat solution for user authentication. In practice, however, PKI has proved to be difficult to deploy due to the difficulty of protecting the private keys and the high maintenance costs [16]. In Korea, there seems to be a common misbelief that PKI is well established for e-commerce and provides high assurance for user authentication.

The protection of the private keys is critical part of PKI. An adversary, by stealing a private key, can impersonate a genuine customer and perform transactions online. Therefore, the private keys should always be protected using secure disk storage and memory mechanisms [20].

Korean banking systems store password-encrypted files of private key material on the user's hard disk. There are two problems with this approach: (1) a successful privilege-escalation attack would allow an adversary to read the decrypted key from memory; and (2) the security is only as good as the password, which malware can steal through brute-force attacks or key logging.

As discussed in Section 2.2, Korean banking systems require a combination of passwords, OTPs, and digital certificates for authentication. We argue that digital certificates are redundant here because the private keys are only protected with the passwords; by compromising the passwords, adversaries also get the private keys. In fact, this is the main reason why the OTPs were introduced in the first place – to limit the exposure to keyloggers. A good OTP system, like the German iTAN, forces the attacker to use session stealing or a man-in-the-browser attack.

If private keys could be strongly protected, however, PKI alone should be sufficient for authentication. In Section 5.3, we discuss whether this could usefully be achieved through the use of trustworthy computing capabilities [18].

3.3 Limitations of the External Plugins

In Section 2.3, we discussed how the customers are obliged to install four external plugins – in addition to the protocol crypto plugin, there's an AV product, a firewall, and keystroke encryption software – to use Internet banking. We identify the inadequacies of these plugins and argue that trusted user platforms cannot be achieved through this approach.

An anti-virus product typically uses either signature-based detection or heuristics-based detection. The former method cannot detect new malware or variants efficiently, and as malware writers get more organised, the proportion of malware that's detected is falling steadily. The latter is based on heuristics such as monitoring registry changes and the insertion of hooks into certain libraries or system interfaces; but as these are not based on any fundamental characteristics of malware, they often incur many false positives and false negatives [43].[12]

The firewall software shipped by Korean banks runs only while the user is using the Internet banking service. So malware that wants to bypass it can just wait until the user logs off to send

[12] Cohen [12] demonstrated that there is no algorithm that can detect all possible viruses.

sensitive information back home. The product may provide some protection against session stealing and man-in-the-browser attacks [15] but seems rather limited. But perhaps this is the most that can be done; if banking software interfered with the user's online activities when she was not engaged in banking, it could be unacceptable.

The keystroke encryption software has limitations too. First, it cannot prevent key scan codes from being intercepted by interrupt hooking methods; the encryption only happens afterwards. Second, successful privilege-escalation attacks or buffer overflow attacks would allow malware to modify or simply delete this software. Third, if not designed carefully, this software could prevent normal key inputs from being read by installed, benign programs. Fourth, it cannot prevent hardware keyloggers in situations where the user accesses the banking services through a public machine. Finally, the encryption keys themselves could be compromised. Without strong protection for the encryption keys (e.g. through a hardware mechanism such as the TPM [18]) and verification of the code responsible for managing these keys, keystroke encryption methods only offer slight improvement.

These external plugins may be effective against simple, naive malware. As soon as a malware becomes sophisticated enough to perform privilege-escalation, in-memory, and brute-force attacks, these plugins become less effective. This raises a question of how much they add. Engineers should consider the balance between widely-deployed protection mechanisms (such as those in the operating system), the benefits of user education (limited as they are), and what should be provided using proprietary mechanisms. (In Section 5.1, we further discuss the social engineering aspects.)

3.4 Lack of Security Proof

Korean Internet banking systems use proprietary authentication protocols. Some may argue that closed protocols will help prevent a number of known attacks; others argue that open protocols can become more secure than the closed ones through ongoing verification and patching efforts. Making the specifications available to the public could help both developers and attackers. Closed information on the other hand can slow down the attackers initially, but attackers eventually find and exploit vulnerabilities. Where does the balance lie?

We believe that prevention is better than cure. Protocols are not like operating systems, which will inevitably have many bugs; protocols are compact things, which are hard to design properly, and which commonly contain one or two bugs initially that get found once many people (including academics) study them. Protocol security should therefore be thoroughly analyzed by experts for a period before any protocol is seriously deployed. We are therefore concerned about the proprietary protocols used in the Korean systems. We have no idea what verification might have been done on them. Are there any formal proofs of correctness? If there were, then surely KISA can have no objection to publishing both the protocols and the proofs. SSL/TLS on the other hand has been studied for a long time, and formally verified [31] – its end-to-end security is equivalent to the cryptographic strength of the underlying algorithms if implemented properly [6]. The security offered by SSL/TLS has been the subject of comprehensive analysis [41, 29, 31, 38, 13, 19]. If and when flaws are found, they are generally fixed rapidly by the community. Security proofs are no panacea, but they do indicate that someone has studied and modelled a protocol carefully. Because SSL/TLS has a security proof and none has been supplied for the Korean protocols, former should be assumed to be more robust.

Arguments against the use of SSL/TLS due to published implementation vulnerabilities are unreasonable [9, 40, 8, 22, 5]. At least the basic protocol is sound; and attacks using timing channels, poor random number generators etc are at least as likely to affect proprietary protocols – if not more so, as the common implementations of SSL/TLS are open-source and widely studied.

4 What the Users Think About the S. Korean Services

To investigate the usability implications of employing the security mechanisms discussed here, we conducted an anonymous online survey (see Table 2) to study (1) how users feel about using Korean services compared to those from other countries; and (2) why users prefer, or do not prefer, using Korean services over services from other countries. We made the assumption that services from other countries do not require their users to install extra security software (which is generally the case).

We invited our friends and colleagues as well as several online communities in Korea to participate, and got a total of 401 participants. 80 of these participants had experiences of using banking services from Korea as well as from other countries (see Q2), out of a total of 379 people who had experience of Korean services (see Q1). The participants' IP addresses were checked to prevent multiple responses.

The results for Q3 show that 70% of those who have had experiences in using both services prefer to use the services from other countries. Q5 reveals that the two most common reasons for this is due to their simplicity and better compatibility with web standards. The results for Q8 and Q9 indicate that 68.9% of those who have used Korean services felt uncomfortable, mainly due to their complexity and lack of compatibility with web standards.

For Q4, Q5, and Q9, some participants offered other interesting reasons. In particular, for Q9, 25 participants commented about the system crashes that result from installing the ActiveX plugins. 4 participants commented about the inconvenience of having to carry around digital certificates to use Internet banking.

It must be said, though, that none of the users were impressed with the security of other countries' banks (Q5) while 58.3% said that the most important reason they prefer Korean services was "It feels more secure" (Q4). We argue in this paper that Korean services are not actually more secure, but perhaps unsophisticated users assume that services which are complex and difficult to use must also be complex and difficult to defraud. Perhaps this is another example of "security theatre", which tackles the feeling but not the reality.

In addition, we compared the network traffic of two Korean banking services (banks A and B) against a UK banking service (bank C) to analyse the relative performance (see Table 3). The network packets were collected and analysed during user login and authentication.

The results show that the number of packets sent/received using the Korean services A and B is roughly 6-7 times higher than that of the UK service C. Similarly, the total byte count is 5-10 times higher in the Korean services, illustrating a relatively high communication cost and low performance. It is also interesting to see that using Korean services involves communicating with multiple web servers: these extra web servers usually serve to distribute the mandatory external plugins. This may indicate a possibility of service-denial attacks or other additional failure modes.

5 Our Recommendations

Having identified the major usability and other problems of the proprietary security mechanisms, we now consider what can be done to improve the overall security and usability of Internet banking systems in Korea.

5.1 Providing Options to the Users

Users are obliged to install a number of security (ActiveX) plugins while using Korean banking services. As a result, they mostly have no option but to use Internet Explorer and Windows for online banking. Our survey shows that most customers find Korean services uncomfortable to

Table 2: Usability Survey Results

Q1. Do you have experience in using a Korean Internet banking service?	
Yes	94.5% (379)
No	5.5% (22)
Q2. Do you have experience using an Internet banking service from another country?	
Yes	21.1% (80)
No	78.9% (299)
Q3. If you had to select one service for Internet banking, which country's service would you prefer to use? (For those who have answered "Yes" to Q2)	
Internet banking service from Korea	30.0% (24)
Internet banking service from another country	70.0% (56)
Q4. What is the most important reason you prefer the Korean service?	
It's simpler	25.0% (6)
It's faster	8.3% (2)
It's more compatible with the Web standards	0.0% (0)
It feels more secure	58.3% (14)
Other	8.3% (2)
Q5. What is the most important reason you prefer the service from another country?	
It's simpler	50.0% (28)
It's faster	1.8% (1)
It's more compatible with the web standards	39.3% (22)
It feels more secure	0.0% (0)
Other	8.9% (5)
Q6. If all of the services provide the same level of security, which country's service would you prefer to use? (For those who have answered "Yes" to Q2)	
Internet banking service from Korea	15.0% (12)
Internet banking service from another country	63.8% (51)
I don't mind using either	21.3% (17)
Q7. Do you know why ActiveX controls are installed on your machine when you use the Korean banking service? (For those who have answered "Yes" to Q1)	
I know exactly	32.2% (122)
I know briefly	49.1% (186)
I have no idea	18.7% (71)
Q8. How often have you felt uncomfortable using the Korean banking service?	
All the time	41.7% (158)
Most of the time	26.9% (102)
Sometimes	27.2% (103)
Never	4.2% (16)
Q9. Why did you feel uncomfortable using the Korean banking service?	
It was complicated	20.9% (76)
It was slow	11.6% (42)
It had compatibility issues with the web standards	45.5% (165)
It felt insecure	7.7% (28)
Other	14.3% (52)

Table 3: Traffic Comparison

Metrics	Korean bank A	Korean bank B	UK bank C
Packet types	TCP, HTTP, SSL/TLS	TCP, HTTP	TCP, HTTP, SSL/TLS
Number of packets sent/received	8,669	8,961	1,301
Total bytes used	3,653,648	7,787,551	801,297
Number of communicating servers	9	5	1

use due to various compatibility and usability issues (see Section 4). Yet most of these plugins only offer a modest improvement, as they are designed to run on top of insecure operating systems and kernels (see Section 3.3). A successful privilege-escalation attack, for example, could turn them off or steal private keys.

Even if they somehow guarantee a significant improvement, it still seems unreasonable to oblige the users to install them without providing comprehensible information about the assumed threats and claimed benefits. Our survey indicates that many users have installed these plugins without knowing what the benefits might be.

A more user-friendly system should only recommend security plugins and allow the users to decide whether or not to install them. In fact, some banks in other countries already allow their users a choice. For example, the Royal Bank of Scotland's "staying safe online" webpage [39] informs their users about a security product that can be downloaded and gives an overview of the threats, benefits, and system requirements. Likewise, the benefits and risks should be clearly explained in terms that normal users can easily understand. This would allow the users who wish to use different web browsers or operating systems to make informed choices.

5.2 Adapting More Compatible Mechanisms

Another way to improve the overall usability is to replace the proprietary mechanisms with better engineered, compatible mechanisms. Since the proprietary plugins have serious weaknesses in the way they are designed or used (see Section 3.3), we believe the overall security should not be harmed (and could well be improved) by replacing them with the following mechanisms.

In Section 3.4 we discussed why SSL/TLS is likely to be safer than proprietary protocols. Using it would not only reduce the number of external components required but let users choose essentially any modern browser. It would also allow the use of industry-standard mechanisms for mitigating phishing attacks such as extended-validation certificates and collaboratively-generated blacklists.

If banks want to rely on digital-certificate-based user authentication, we recommend using SSL/TLS client certificate with standard cryptographic token interfaces. Netscape-based browsers already support PKCS #11 [35], and Internet Explorer supports Microsoft's Cryptographic Application Programming Interface (CAPI) [28]. These APIs would allow both banks and users flexibility in choosing the cryptographic component suitable for their operating environment: this component is called a Cryptographic Service Provider (CSP) in CAPI and cryptoki in PKCS #11.

The firewall plugin does not add much to security since it only runs during a banking session. The banks should think about whether session-stealing and man-in-the-browser attacks are real issues in Korea; if not, this component could be dispensed with.

5.3 More Trustworthy Computing Approaches

Adding extra security components on top of an insecure operating system or kernel will not result in a trusted user platform. If the threats discussed in Section 2.3 become critical in

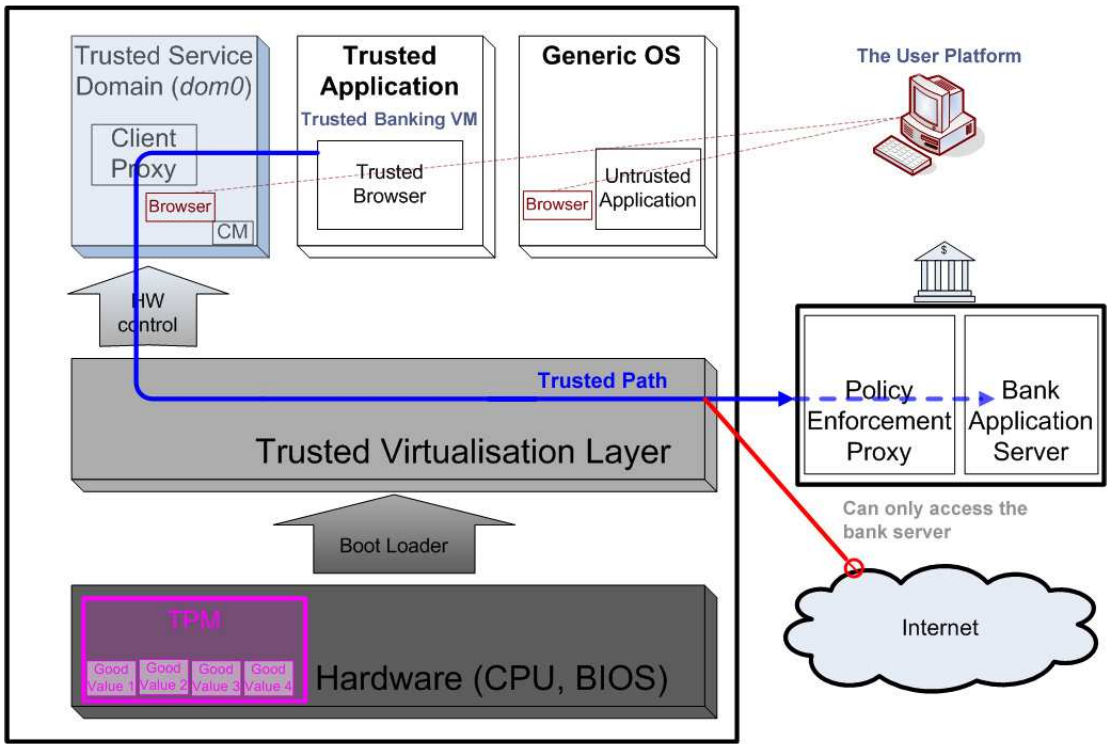


Figure 5: OpenTC's Secure Banking Prototype

the future, and the provision of trusted platforms becomes necessary, a more complete security solution should be deployed instead. What are the options?

By a complete solution, we refer to an online banking system that (1) runs on top of a trustworthy, integrity-protected kernel and security components, (2) is strongly isolated from the rest of the system, (3) has its private keys protected by both software and hardware, (4) allows a remote server to verify the integrity of these security properties, and (5) communicates with the bank server through a trusted path. Can we get a trustable, verifiable user platform from any technologies that already exist, or may be adapted in the near future?

The isolation requirement might at first sight be amenable to protections based upon operating-system level techniques like sandboxing. However, given the size and complexity of mainstream operating systems and their known vulnerabilities [36], it would appear that strong isolation is hard to achieve in this way. Virtualization [42] may be able to provide much stronger isolation thanks to the much smaller virtual machine monitor.

The Open Trusted Computing (OpenTC) consortium has implemented a "Secure Banking" prototype [23] based on Xen virtual machine monitor [42] (see Figure 5). Their prototype serves to demonstrate how virtualization and trusted computing could be used to prepare a trusted (TPM-measured) banking virtual machine, verify its state, and use it to access the bank server. If remote attestation [18] can be made to work, it might ensure that only a fully isolated, integrity-protected banking virtual machine can access the bank server to perform online transactions. The banking virtual machine could also verify the authenticity and integrity of the bank server. Using this virtual machine, the user can only ever reach the bank server – this would be sufficient to prevent phishing attacks. Any private key used is sealed to the trusted virtual machine, preventing other untrusted virtual machines from accessing the key.

This type of approach could provide a highly secure, verifiable online banking environment. However, it has several drawbacks. First, the early promise of trusted computing seems to have petered out; people have not been able to make remote attestation work (modern operating systems are too complex and change too much). Second, the TC approach shares many of the problems already displayed by the Korean online banking system – it requires essentially proprietary software, it would lock people into a particular platform, it would deprive them of the option of doing online banking via their choice of browser and operating system, and might well display similar usability issues. (The software developers would not be benefiting from all the worldwide effort invested in browsers and the user experience generally.)

A second possible solution might be something like a bootable USB stick developed by Australia's Commonwealth Scientific and Industrial Research Organisation (CSIRO). This "Trusted Extension Device" [30] allows a trustworthy environment to be initiated from any untrustworthy one. The trustworthy environment is defined and enforced by the issuer. Trust is established with a remote bank server through trusted computing attestation where both ends authenticate each other based on their integrity reports. If attestation is successful, the banking software built into the Trust Extension Device and the remote server perform further transactions through a secure communication channel.

This approach too has drawbacks. Each bank would create their own portable devices and distribute them to customers. A customer may have many accounts with different banks, and could end up keeping track of several devices. More costs arise when such devices need to be updated and patched: the banks would have to redistribute the devices and maintain a revocation list of compromised (or out-of-date) ones. And again, there would be the bundle of usability and compatibility issues to be expected with proprietary software.

A third possible solution might be to combine the USB idea with the idea underlying the CAP readers used in Britain and elsewhere [15]. The CAP reader accepts a smartcard (bank customers worldwide are being issued with these under the EMV programme) which can be used not just to generate OTP codes for logon but also to authenticate transactions using a MAC generated by a secret key in the smartcard. The main limitation of the CAP devices deployed in Europe is the lack of a connection to the PC: they are free-standing handheld devices into which the customer must retype transaction data. This retyping is clunky; many banks don't use it, while others only insist on it for high-risk transactions. The way to fix this is to make a 'CAP v 2' with a USB connection to the PC. Then, transaction data could be exported from the PC to the smartcard via the CAP and properly authenticated (with traffic protected by a MAC, for example). In this way each user could have a single USB CAP device into which he would plug the bank card issued on whatever account he wished to use online. A USB smartcard reader with a display and keypad might cost $10–20 if manufactured in quantity.

Something like the third option is, in our view, the most sensible medium-term solution to the problem of getting a complete, secure online banking solution into the hands of users at an acceptable cost – both in terms of the capital expenditure required by banks and the burden of usability and compatibility that they impose on their customers. Hence, if online bank fraud should get significantly worse, we would recommend that such options should be evaluated carefully. The biggest problem we anticipate is that if every banking session is split between a PC screen, which is not trustworthy, and a CAP screen which is, then attackers could try a number of social engineering attacks to persuade customers to authenticate wrong data using their CAP. How do you get customers to disregard a prominent screen warning such as "Customer notice – your CAP needs a security upgrade. Please authenticate the message number "349A3D" in order to install the new software" and focus instead on what the CAP is saying? Splitting transactions between a good device and a bad one is hard, and deserves serious research.

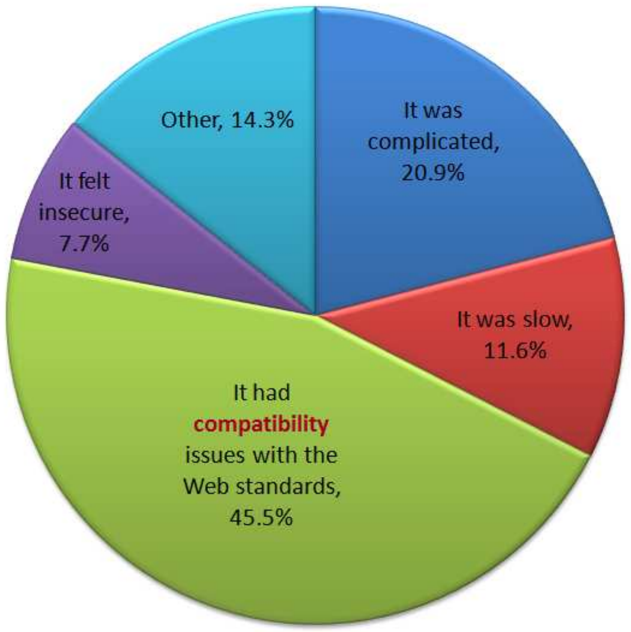


Figure 6: Why Did You Feel Uncomfortable Using the Korean Banking Service?

6 Conclusion

Korean banking offers an interesting natural experiment. While most banks worldwide offer online services to their retail mass market via web browsers, banks in Korea have for ten years insisted that customers use proprietary encryption software that is based on ActiveX controls, together with antivirus, firewall and keylogging countermeasures.

The effects have been rather mixed. On the one hand, Microsoft's Internet Explorer has an almost complete monopoly in Korea, as customers can't do banking using Firefox or Opera. And the Korean strategy is unpopular: we surveyed people who have used both Korean and other banking services, and found that over two-thirds of them felt uncomfortable using Korean services, mainly due to their complexity and lack of compatibility with the web standards. Curiously, most customers thought Korean systems more secure, even though they aren't really – presumably because they associate security with annoying complexity!

At the technical level, we have argued that proprietary security software offers at best a modest improvement, and certainly cannot provide the Holy Grail of a trustworthy user platform. We have recommended that banks should minimize the use of external plugins by replacing their proprietary solutions with SSL/TLS. The installation of additional security software should be optional, and the associated risks should be communicated in a comprehensible manner so that users can make their own security decisions.

The lessons learned in Korea may have much wider application. When security researchers discuss what options there might be if online bank fraud gets much worse, one of the possibilities is the use of proprietary systems are software – from variants on the 'trusted computing' theme to virtualization and proprietary bank client software. The Korean experience suggests that these should be treated with caution: the usability and compatibility costs they impose on users are likely to be nontrivial, and they may indeed be unacceptable (and undeployable in competitive markets). A more sensible medium-term solution may be to provide a separate trusted platform in the form of a CAP-like smartcard reader, with a trustworthy keyboard and display, into which the customer can insert her bank card to authenticate transactions. To provide usable transaction authentication, such a device should import transaction data automatically from the PC (for example, using a USB cable) rather than requiring it to be retyped. Even so, there is a remaining hard problem. Given a two-platform system, where one platform is not trustworthy (the PC) and the other platform is much more trustworthy (the CAP device, or for that matter a mobile phone), how do you mitigate the risks of social engineering attacks launched through the first platform?

In future work, we intend to conduct a threat and risk analysis on Korean Internet banking systems by studying the attack trends and evaluating their severity and likelihood. Very few countries publish full bank fraud figures (in Europe, only Britain, France and the Netherlands do); Korea is not there yet. We recommend that the government publish robust electronic crime statistics, as was for example recommended in a 2008 report to ENISA for the case of European banks [4]. In the meantime, perhaps a crime victim survey can compare customers' experiences across countries. We believe that inter-country comparisons are important to define the security requirements and determine whether in fact it is necessary to invest in a next generation of online banking security technology.

7 Acknowledgments

The authors would like to thank Joseph Bonneau, Andrew Martin, Eric Kerfoot, John Lyle, Cornelius Namiluko, Joe Loughry, Ronald Kainda, Mingqiu Song, and Wattana Viriyasitavat for their careful attention and insightful comments. The authors would like to extend their gratitude and thanks to everyone who has participated in the usability survey.

References

[1] Korean Home-brew on the Web, 2007.

[2] SatCounter GlobalStats, 2009.

[3] The Bank of Korea, 2009.

[4] R. Anderson, R. Böehme, R. Clayton, and T. Moore. Security Economics and the Internal Market. Technical report, The European Network and Information Security Agency, 2008.

[5] G. V. Bard. A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL. In *Proceedings of the International Conference on Security and Cryptography*, pages 99–109, 2006.

[6] L. D. Bisel. The Role of SSL in Cybersecutiry. *IT Professional*, 9(2):22–25, 2007.

[7] M. Blaze, W. Diffie, R. L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener. Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security. Technical report, January 1996.

[8] D. Brumley and D. Boneh. Remote Timing Attacks Are Practical. In *SSYM'03: Proceedings of the 12th Conference on USENIX Security Symposium*, pages 1–14, Berkeley, CA, USA, 2003. USENIX Association.

[9] P. Burkholder. SSL Man-In-The-Middle Attacks. Technical report, 2002.

[10] Y. T. Chang. Dynamics Of Banking Technology Adoption : An Application To Internet Banking. Technical report, 2003.

[11] J. Claessens, V. Dem, D. D. Cock, B. Preneel, and J. Vandewalle. On the Security of Today's Online Electronic Banking Systems. *Computers and Security*, 21(3):253 – 265, 2002.

[12] F. Cohen. Computer Viruses: Theory and Experiments. *Computers and Security*, 6(1):22–35, 1987.

[13] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *SSYM'01: Proceedings of the 10th Conference on USENIX Security Symposium*, pages 1–1, Berkeley, CA, USA, 2001. USENIX Association.

[14] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Informational), 1999.

[15] S. Drimer, S. J. Murdoch, and R. Anderson. Optimised to fail: Card readers for online banking. In *Financial Cryptography and Data Security*, February 2009.

[16] C. Ellison and B. Schneier. Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[17] D. Forte. Anatomy of a Phishing Attack: A High-level Overview. *Network Security*, 2009(4):17 – 19, 2009.

[18] D. Grawrock. *The Intel Safer Computing Initiative*, chapter 1–2, pages 3–31. Intel Press, 2006.

[19] A. Hess, J. Jacobson, H. Mills, R. Wamsley, K. E. Seamons, and B. Smith. Advanced Client/Server Authentication in TLS. In *Proceedings of Network and Distributed Systems Security Symposium*, 2002.

[20] A. Hiltgen, T. Kramp, and T. Weigold. Secure Internet Banking Authentication. *IEEE Security and Privacy*, 4(2):21–29, 2006.

[21] G. Kanai. the cost of monoculture, 2007.

[22] V. Klima, O. Pokorny, and T. Rosa. Attacking RSA-based Sessions in SSL/TLS. In *Cryptographic Hardware and Embedded Systems (CHES), 2003*, pages 426–440. Springer, 2003.

[23] D. Kuhlmann, S. L. Presti, G. Ramunno, D. Vernizzi, E. Bayer, and B. Gngren. Private Electronic Transaction (PET) Proof-of-concept Prototype Documentation, March 2008.

[24] H. Lee, S. Lee, J. Yoon, D. Cheon, and J. Lee. The SEED Encryption Algorithm. RFC 4269 (Informational), December 2005.

[25] M. Mannan and P. C. van Oorschot. Security and Usability: the Gap in Real-world Online Banking. In *NSPW '07: Proceedings of the 2007 Workshop on New Security Paradigms*, pages 1–14, New York, NY, USA, 2008. ACM.

[26] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[27] Microsoft. Description of ActiveX Technologies, 2007.

[28] Microsoft. Cryptography API, 2009.

[29] J. C. Mitchell. Finite-State Analysis of Security Protocols. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 71–76, London, UK, 1998. Springer-Verlag.

[30] S. Nepal, J. Zic, H. Hwang, and D. Moreland. Trust Extension Device: Providing Mobility and Portability of Trust in Cooperative Information Systems. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803/2010, pages 253–271. Lecture Notes in Computer Science, 2007.

[31] L. C. Paulson. Inductive analysis of the internet protocol tls. *ACM Transactions on Information and System Security (TISSEC)*, 2(3):332–351, 1999.

[32] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The Ghost in the Browser Analysis of Web-based Malware. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[33] RedTeam. iTAN Online-banking Security System. CAN-2005-2779, 2005.

[34] RSA Laboratories. PKCS #5: Password-Based Encryption Standard, 1993.

[35] RSA Laboratories. PKCS #11 v2.30: Cryptographic Token Interface Standard, 2009.

[36] A.-R. Sadeghi and C. Stüble. Taming "Trusted Platforms" by Operating System Design. In *Information Security Applications*, volume 2908, pages 1787–1801. Lecture Notes in Computer Science, 2004.

[37] Secure Electronic Transaction LLC. SET Secure Electronic Transaction Specification — Version 1.0, May 1997.

[38] M. Steiner, U. D. Saarlandes, P. Buhler, T. Eirich, and M. Waidner. Secure Password-Based Cipher Suite for TLS. In *Proceedings of Network and Distributed Systems Security Symposium*, pages 134–157, 2001.

[39] The Royal Bank of Scotland. Staying safe online, 2010.

[40] S. Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ... In *EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 534–546, London, UK, 2002. Springer-Verlag.

[41] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 29–40, Berkeley, CA, USA, 1996. USENIX Association.

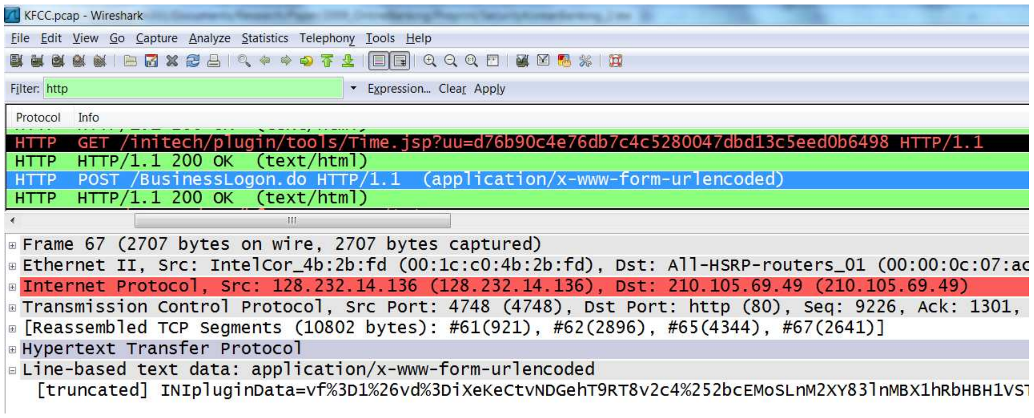
[42] Xen Source. Xen: Enterprise Grade Open Source Virtualization a XenSource White Paper. Technical report, 2005.

[43] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 116–127, New York, NY, USA, 2007. ACM.

[44] J. Youll. Fraud Vulnerabilities in SiteKey Security at Bank of America. Technical report, Challenge/Response, LLC, 2006.

A Examples of traffic

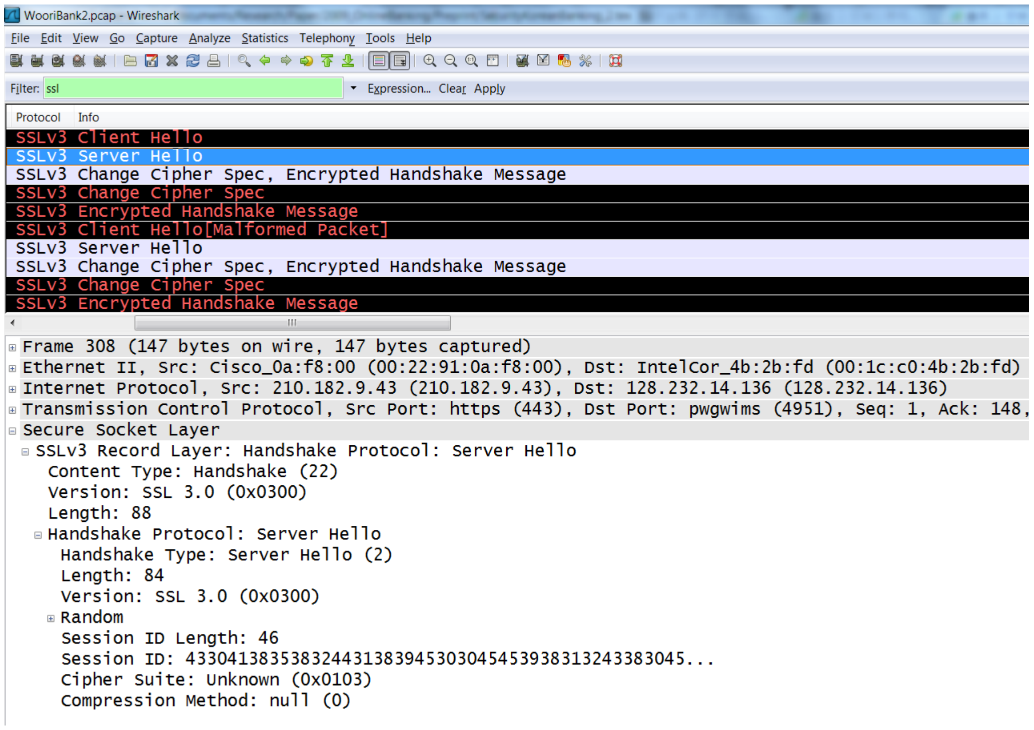
We analysed two key establishment protocols offered by different vendor companies, protocol 1 and protocol 2 (see Figure 7).


KFCC.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: http
Expression... Clear Apply

```
Protocol  Info
HTTP  GET /initech/plugin/tools/Time.jsp?uu=d76b90c4e76db7c4c5280047dbd13c5eed0b6498 HTTP/1.1
HTTP  HTTP/1.1 200 OK  (text/html)
HTTP  POST /BusinessLogon.do HTTP/1.1  (application/x-www-form-urlencoded)
HTTP  HTTP/1.1 200 OK  (text/html)

⊞ Frame 67 (2707 bytes on wire, 2707 bytes captured)
⊞ Ethernet II, Src: IntelCor_4b:2b:fd (00:1c:c0:4b:2b:fd), Dst: All-HSRP-routers_01 (00:00:0c:07:ac
⊞ Internet Protocol, Src: 128.232.14.136 (128.232.14.136), Dst: 210.105.69.49 (210.105.69.49)
⊞ Transmission Control Protocol, Src Port: 4748 (4748), Dst Port: http (80), Seq: 9226, Ack: 1301,
⊞ [Reassembled TCP Segments (10802 bytes): #61(921), #62(2896), #65(4344), #67(2641)]
⊞ Hypertext Transfer Protocol
⊟ Line-based text data: application/x-www-form-urlencoded
    [truncated] INIpluginData=vf%3D1%26vd%3DiXeKeCtvNDGehT9RT8v2c4%252bcEMoSLnM2XY83lnMBX1hRbHBH1VST
```

(a) Protocol 1


WooriBank2.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: ssl
Expression... Clear Apply

```
Protocol  Info
SSLv3  Client Hello
SSLv3  Server Hello
SSLv3  Change Cipher Spec, Encrypted Handshake Message
SSLv3  Change Cipher Spec
SSLv3  Encrypted Handshake Message
SSLv3  Client Hello[Malformed Packet]
SSLv3  Server Hello
SSLv3  Change Cipher Spec, Encrypted Handshake Message
SSLv3  Change Cipher Spec
SSLv3  Encrypted Handshake Message

⊞ Frame 308 (147 bytes on wire, 147 bytes captured)
⊞ Ethernet II, Src: Cisco_0a:f8:00 (00:22:91:0a:f8:00), Dst: IntelCor_4b:2b:fd (00:1c:c0:4b:2b:fd)
⊞ Internet Protocol, Src: 210.182.9.43 (210.182.9.43), Dst: 128.232.14.136 (128.232.14.136)
⊞ Transmission Control Protocol, Src Port: https (443), Dst Port: pwgwims (4951), Seq: 1, Ack: 148,
⊟ Secure Socket Layer
  ⊟ SSLv3 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 88
    ⊟ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 84
        Version: SSL 3.0 (0x0300)
      ⊞ Random
        Session ID Length: 46
        Session ID: 43304138353838324431383945303045453938313243383045...
        Cipher Suite: Unknown (0x0103)
        Compression Method: null (0)
```

(b) Protocol 2

Figure 7: Examples of traffic

In protocol 1, the user's certificate and the session key encrypted with the server's public key are delivered to the banking server using the HTTP "POST" method. We briefly describe this with the following notation. The symbols C and B represent the client software and bank server, respectively. For data input y, $S_X(y)$ and $P_X(y)$ denote the data values resulting, respectively, from the signature operation on y using X's private signing key, and the encryption operation on y party X's public encryption key. t_X is a timestamp generated by X. $cert_X$ is a certificate binding X to a public key suitable for both encryption and signature verification. E

is a symmetric encryption algorithm (e.g., SEED). $k_{X_1X_2}$ is a secret symmetric session key to be shared by X_1 and X_2.

$$\begin{array}{ll} B \longrightarrow C: & cert_B, t_B \\ C \longrightarrow B: & cert_C, P_B(t_B, k_{BC}), S_C(cert_C, P_B(t_B, k_{BC})), \\ & E_{k_{BC}}(cert_C, P_B(t_B, k_{BC}), S_C(cert_C, P_B(t_B, k_{BC}))) \end{array}$$

Protocol 2 is almost the same as the RSA-based SSL/TLS [41, 14] except SEED is used for encryption.