CAPTCHA-based Code Voting

Rolf Oppliger[1], Jörg Schwenk[2], Christoph Löhr[2]

[1]eSECURITY Technologies
CH-3073 Gümligen
rolf.oppliger@esecurity.ch

[2]Ruhr-University Bochum
D-44780 Bochum
{joerg.schwenk|christoph.loehr}@rub.de

Abstract: Code voting provides an appropriate technology to address the secure platform problem of remote Internet voting, but it is not particularly user-friendly. In this paper, we propose the use of CAPTCHA- an acronym standing for Completely Automated Public Turing tests to tell Computers and Humans Apart - to improve the user-friendliness of code voting, discuss the security of CAPTCHA-based code voting, and elaborate on a possible implementation.

1 Introduction

Elections and votes are fundamental processes for the proper operation of democratic states and their (democratically legitimated) governments. In the literature, the term *electronic voting* (or *e-voting* in short) is used to refer to elections and votes that are supported by electronic means. With the proliferation of the Internet, its use for e-voting has been proposed by many people (mainly politicians) as a way to make voting more convenient andas it is hopedto increase participation in elections and votes. The term *Internet voting* is therefore used to refer to election or voting processes that enable voters to cast their ballots over the Internet. This basically means that the ballots must be represented electronically, and that the electronic ballots must be transmitted to election officials using the Internet as a transport medium.

There are many possibilities to implement Internet voting, and poll-site Internet voting, Kiosk voting, and remote Internet voting are usually distinguished in the literature (e.g., [Cal00]). In this paper, we only focus on remote Internet voting, i.e., Internet voting where the voter (or a third party acting on behalf of the voter) uses his personal computer (PC) to cast a ballot over the Internet. From a security viewpoint, remote Internet voting is the most challenging possibility to implement Internet voting. In states that support absentee balloting, such as all-postal voting, any other form of Internet voting (i.e., poll-site Internet voting and Kiosk voting) is likely to fail. This is because the other possibilities require the voter to visit a voting place, and this is probably too inconvenient compared to the simplicity of casting ballots from home. In Europe, for example, a few states have started to employ remote Internet votingbe it in geographically restricted pilot projects, such as in three cantons of Switzerland [Ber08], or for official use, such as in Estland.

Against this background, it is possible and likely that the use of remote Internet voting tends upwards, and that the security of remote Internet voting will become a major issue. Security, in turn, has many aspects, and there are several partly complementary security technologies, mechanisms, and services that can be used to address them. As argued in [Opp02], code voting, i.e., voting by providing randomly-looking codes instead of YES or NO in the case of a vote and candidates' names in the case of an election, is an appropriate technology to address the secure platform problem of remote Internet voting. Unfortunately, code voting is not particularly user-friendly, and in this paper we exlore possibilities to use *Completely Automated Public Turing tests to tell Computers and Humans Apart* (CAPTCHAs) also known as *Reverse Turing Tests* (RTTs) or Human *Interactive Proofs* (HIPs) to improve the user-friendliness of code voting. We think that CAPTCHA-based code voting provides an interesting possibility to implement code voting in a real-world setting.

The rest of the paper is organized as follows: The security requirements of (remote) Internet voting are summarized in Section 2. Code voting and CAPTCHA-based code voting are introduced and discussed in Sections 3 and 4. A preliminary security analysis is given in Section 5. Finally, conclusions are drawn and an outlook is given in Section 6.

2 Security Requirements

There are many investigations and studies that elaborate on the security of Internet voting in general, and remote Internet voting in particular (e.g., [Cal00, Rub01]). The results all give evidence that security (including privacy and reliability) is among the most important preconditions for the successful deployment of Internet voting. The current paper ballot systems set a standard that is adopted as a security baseline for Internet voting. They represent certain tradeoffs between voter convenience and protection against fraud and abuse. It is generally required that elections and votes conducted over the Internet are at least as secure as the current paper ballot systems. In states that support absentee balloting in the form of all-postal voting, however, it is this voting technology that sets the security standard for Internet voting.

There are many lists of security requirements for (remote) Internet voting that can be found in the literature[55]. There is even an e-voting protection profile for the Common Criteria drafted in Germany[56]. The following list of security requirements is not intended to be complete or comprehensive:

- Completeness and soundness of the Internet voting protocol(s);
- Correctness of the results;
- Authenticity of both the voter (or the voting client acting on behalf of the voter, respectively) and the voting server;
- Secrecy of the ballots (including, for example, anonymity of the voter);
- Integrity of the ballots (including, for example, protection against malicious software);
- Non-duplication of the ballots;
- Availability and reliability of the voting process (including, for example, protection against denial-of-service attacks).

Some security requirements are complementary and don't interact with each other (e.g., integrity and non-duplication of the ballots). Other security requirements, however, are (or at least seem to be) contradictory in some sense. For example, one way to attest the correctness of a voting process is auditability, meaning that the entire voting process can be audited in some reasonable way. Auditability, however, sometimes contradicts to the secrecy of the ballots. In fact, there is a lot of research going on in the cryptographic community to address this apparent contradiction and to guarantee ballot secrecy and the correctness of the results simultaneously. Most of this research elaborates on schemes and protocols for verifiable secret sharing and secure multi-party computation as pioneered by Yao [Yao82].

[55] In 2004, for example, the Committee of Ministers of the Council of Europe adopted Recommendation Rec(2004)11 that specifies “legal, operational and technical standards for e-voting.” These standards, among other things, also comprise security requirements.
[56] http://www2.dfki.de/fuse

Many security requirements of (remote) Internet voting can be addressed with existing technologies, mechanisms, and services. For example, there are many technologies that can be used to secure the server side. Examples include firewall technologies and intrusion detection systems (IDS) or intrusion prevention systems (IPS). The authenticity of the voter and the voting server can be addressed with public key certificates. Similarly, the secrecy and integrity of the ballots can be guaranteed with a cryptographic protocol, such as the Secure Sockets Layer (SSL) [FKK96] or Transport Layer Security (TLS) [DR06] protocol. It is, however, important to note that the use of the SSL/TLS protocol protects the secrecy and integrity of the ballots only during the transmission over the Internet. The ballots are not automatically protected on the client or server side. In fact, additional security technologies, mechanisms, and services are required to protect the secrecy and integrity of the ballots before and after they are transmitted over the Internet. There are additional risks for the secrecy of the ballots (i.e., privacy risks) related to the use of spyware (in the home setting) or remote system administration tools (in the institutional setting).

Due to the fact that a remote Internet voter uses his PC to cast a ballot and that this PC may be subject to malware, the insecurity of the client-side platform represents the major vulnerability (and Achilles heel) of remote Internet voting. Rivest coined the term secure platform problem to refer to the problem of protecting an inherently insecure client-side platform against malicious software and corresponding attacks [Riv01].

Due to the fact that the *secure platform problem* is hard and difficult to solve, there are several e-voting research and development projects that don't even address it. For example, in the FAQ document of the European CyberVote project[57], the question “Can a virus or Trojan horse attack CyberVote?” is answered in the following way:

“Yes, like any other client software in an insecure PC environment.

Anti-virus software should be used and strict security guidelines followed to limit the risk of a virus or Trojan horse attack.

Secure user interface techniques can be applied to the CyberVote client to prevent Trojan horses.”

Unfortunately, the FAQ document does not further explain the term “secure user interface techniques.” It turns out that there are not many security technologies, mechanisms, and services that can be used to effectively address the secure platform problem of remote Internet voting. In fact, we think that code voting as introduced next is one (if not the only) technology that may work in a real-world setting.

[57] http://www.eucybervote.org/faq_security.html#q35

3 Code Voting

The term *code voting* is used to refer to an e-voting technology in which the voter casts his ballot by providing a voting code instead of YES or NO (in the case of a vote) or a candidate's name (in the case of an election). The voting code, in turn, looks like a random string. If the alphabet consists of all decimal digits 0...9, then the voting code basically represents a number. In general, however, any alphabet can be used and the voting codes can be arbitrarily long.

To the best of our knowledge, the first code voting system was proposed by Chaum [Cha01]. In such a system, each voter is equipped with a code sheet (i.e., a sheet that itemizes all voting codes) and he must enter the appropriate voting codes to cast his ballot. An exemplary code sheet for an election is illustrated in Table 1. If the voter wants to vote for Bob, then he must enter 990234 (instead of "Bob").

Candidate	Voting code
Alice	236412
Bob	990234
Carol	141290
Dave	782755
Eve	774892
…	…

Table 1: A code sheet with voting codes

Due to the fact that voting codes look like random strings, code voting effectively protects against passive and active attacks:

- In a passive attack, the adversary sees a voting code sent over a network (using, for example, a network management or system administration tool), and must then be able to tell whether this code represents YES or NO (in the case of a vote) or to which candidate the code actually refers to (in the case of an election). If the voting codes are chosen with a good random bit generator or a cryptographically secure pseudorandom bit generator (PRBG), then the best the adversary can do is guessing. In this case, seeing the voting codes sent over the network does not help the adversary.

- In an active attack, the adversary does not only see a voting code sent over a network, but he can also manipulate it. For example, the adversary may employ malware or a client-side remote system administration tool to turn a voting code representing YES into a voting code representing NO (in the case of a vote) or a voting code of one candidate into a voting code of another candidate (in the case of an election). Again, if the voting codes are chosen with a good random bit generator or a cryptographically secure PRBG, then the adversary does not know the other voting codes, and hence the best he can do is again guessing.

In either case, the success probability of an adversary is not better than guessing, meaning that the best an adversary can do is guessing. This is indepedent from the adversary's computational resources and available time. Consequently, the security that is achieved is unconditional or information-theoretic. There are, however, two conditions that must be fulfilled to achieve this level of security:

- As mentioned before, the voting codes must be random, i.e., they must be chosen with a good random bit generator or a cryptographically secure PRBG;
- The code sheets must be personal and distributed out-of-band[58], using, for example, a trustworthy postal mail delivery service.

Also, it is important to note that code voting requires a modified voting behavior, and that there may be some legal constraints to consider (not addressed in this paper).

In spite of the fact that code voting as discussed so far is able to provide unconditional or information-theoretic security, it may still be the case that an (active) adversary simply deletes a voting code in transit. To protect against this attack, it may be worthwhile to have the server send back a verification code and have the voter verify this code.

Table 2 illustrates an exemplary code sheet with voting and verification codes. Again, if the voter wants to vote for Bob, then he must enter 990234 and wait for the server to send back the verification code 672345. If another verification code is sent back, then something illegitimate is going on and the voter is well advised to stop voting (needless to say that some dispute-resolving mechanisms must also be put in place here).

Candidate	Voting code	Verification code
Alice	236412	124355
Bob	990234	672345
Carol	141290	045686
Dave	782755	687432
Eve	774892	234115
…	…	…

Table 2: A code sheet with voting and verification codes

If the voter verifies the verification code, then it makes a lot of sense to communicate the result of the verification step to the server (otherwise, the server does not know whether the result is correct). This is where the confirmation code comes into place. Table 3 illustrates an exemplary code sheet with voting, verification, and confirmation codes. In our toy example, the voter would confirm the successful verification of the verification code 672345 by sending the confirmation code 574546 to the server. At this point, there is no need to continue the recursion (and send more codes back and forth).

[58] It is important that the code sheets must be provided outside the voter's PC (i.e., the PC that is used by the voter to cast his vote). If the code sheets were inside the PC, then malicious software could get and use them to change the ballots. Also, the voting codes must be randomly or pseudo-randomly chosen from a sufficiently large set of possible values to make the probability that malicious software can correctly guess them sufficiently small (i.e., negligible).

Candidate	Voting code	Verification code	Confirmation code
Alice	236412	124355	252435
Bob	990234	672345	574546
Carol	141290	045686	124145
Dave	782755	687432	243521
Eve	774892	234115	967468
…	…	…	…

Table 3: A code sheet with voting, verification, and confirmation codes

The bottom line is that there are many possibilities to implement code voting. In addition to casting a vote by simply entering a voting code, the voter may verify a verification code sent back from the server (to verify that he has casted the vote to an authentic server, and that the vote has been properly registered by the server). Also, the voter may acknowledge proper verification of the verification code by sending out a confirmation code.

In Table 4, we summarize the 2^3-1=7 possibilities to implement code voting. Among these possibilities, we think that the following four possibilities are meaningful in practice:

- Voting code-only implementation;
- Verification code-only implementation;
- Voting and verification code implementation;
- Full implementation (i.e., voting, verification, and confirmation codes).

Possibilities	Voting code	Verification code	Confirmation code
Voting code-only implementation	X		
Verification code-only implementation		X	
			X
Voting and verification code implementation	X	X	
	X		X
		X	X
Full implementation	X	X	X

Table 4: Possibilities to implement code voting

In a voting code-only implementation, the voter casts his ballot by simply sending a voting code to the server. In a verification code-only implementation, the voter casts his ballot as usual, but waits for a verification code sent back from the server. It is then up to the voter to verify this code. A verification code-only implementation is particularly interesting, because the voter has to minimally change his behaviour (i.e., he can still enter YES or NO and only validate the verification number sent back from the server). This advantage, however, may also be disadvantageous, because it is possible and likely that some voters don't care about the validity of verification codes sent back. As its name suggests, a voting and verification code implementation employs voting and verification codes. Last but not least, a full implementation employs voting, verification, and confirmation codes. It goes without saying that this is the preferred choice from a security viewpoint, and that all other choices represent tradeoffs.

A practically relevant question refers to the length of the various codes. Obviously, the length must make the probability to correctly guess a code sufficiently small. For example, if the number includes 10 binary digits (bits), then the probability of correctly guessing a code is $1/2^{10} = 1/1{,}024 = 0.000975562$. Due to the fact that the numbers cannot be verified off-line (without access to the code sheets), this seems to be sufficient. 10 bits can be represented with $\log 2^{10} = \log_{10} 1{,}024$ decimal digits which is slightly more than 3 digits. Consequently, 4 decimal digits can be used to encode a code and some redundancy to detect errors (error detection is particularly important for voting and confirmation codes that are entered by the user).

In theory, 10-bit code numbers can be randomly generated, using a random bit generator. In practice, however, the code numbers are more likely generated with an appropriately seeded pseudorandom bit generator (PRBG) or a construction that employs a keyed hash function, such as the HMAC construction [KBC97]. In either case, the generation of the code numbers is not further addressed in this paper.

Last but not least, we note that a guessing attack may have an equalizing effect on the outcome of an election or vote. If, for example, a candidate only gets a few votes under “normal” circumstances, then he may get an average number of votes under a guessing attack. This is because it is equally likely to guess a voting code for an unpopular candidate as it is to guess a voting code for a popular candidate. Hence, the outcome of an election or vote that is subject to a guessing attack may be equalized to some extent. Because we do not further address guessing attacks, this point is not further discussed in this paper.

4 CAPTCHA-based Code Voting

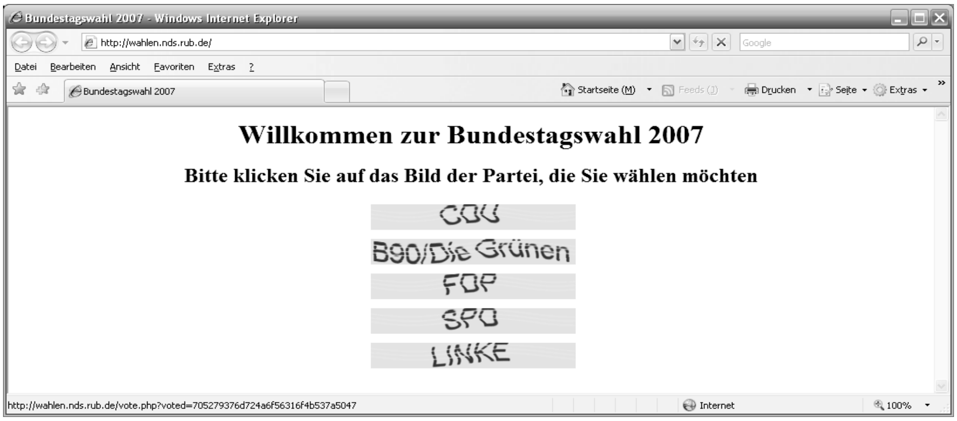
The potential difficulty of differentiating humans from computers pretending to be humans was addressed already in 1950, when Turing described his now-famous test. In short, the *Turing test* is a proposed test for a machine to demonstrate intelligence [Tur50]. It proceeds as follows: a human judge engages in a natural language conversation with one human and one machine, each of which are trying to appear human. If the judge cannot reliably tell which is which, then the machine is said to pass the Turing test. In order to keep the test setting simple and universal (to explicitly test the linguistic capability of the machine instead of its ability to render words into audio), the conversation is usually limited to a text-only channel such as a teletype machine as Turing suggested, or, more recently, Internet-based messaging.

In the mid-1990s, people came up with the idea of using a reverse Turing test to have a machine test whether a user is human. For example, in 1995, Lam of The Chinese University of Hong Kong implemented a reverse Turing test in a voting application written for Radio Television Hong Kong. The public was able to vote for their favourite singers and songs online for the first time in the annual "Top Ten Chinese Songs Award." To prevent automatic and machined submissions, users were required to correctly input a 6-digit number that was represented as an image. In 1996, the first reference of automated tests, which distinguish humans from computers for the purpose of controlling access to Web services, appeared in a manuscript of Naor [Nao96]. Other primitive reverse Turing tests seem to have been developed in 1997 at AltaVista to prevent bots from adding URLs to their search engine.

In 2000, von Ahn and Blum developed and publicized the notion of a *Completely Automated Public Turing test to tell Computers and Humans Apart* (CAPTCHA), which included any program that can distinguish humans from computers. They invented multiple examples of CAPTCHAs, including the first CAPTCHAs to be widely used on the Internet (at Yahoo!) [vABL04]. The acronym CAPTCHA is trademarked by Carnegie Mellon University. Alternatively, a CAPTCHA is sometimes called *Reverse Turing Test* (RTT) or *Human Interactive Proof* (HIP).

In general, there are many possibilities to implement CAPTCHAs, RTTs, or HIPs. A common type of (visual) CAPTCHAs requires that the user type in the letters of a distorted image, sometimes with the addition of an obscured sequence of letters or digits that appears on the screen. Such CAPTCHAs are also used in this paper (as an example). But there are many other visual CAPTCHAs and CAPTCHAs based on audio or video. More recently, for example, Microsoft Research has come up with a HIP called ASIRRA (Animal Species Image Recognition for Restricting Access) that works by asking users to distinguish between photographs of cats and dogs [E+07]. Audio CAPTCHAs, in turn, have been developed and are being deployed for handicapped persons. In essence, any task that can be efficiently solved by a human but is not known to be efficiently solvable by a machine can be turned into a CAPTCHA, RTT, or HIP. There are many opportunities for research and development here.

In CAPTCHA-based code voting, the voter does not cast his ballot directly by providing an appropriate voting code, but indirectly by clicking on an appropriate CAPTCHA. Clicking on a CAPTCHA, in turn, causes a random-looking voting code (representing a cryptographic hash value) to be sent from the browser to the server. Let us consider an exemplary (and simplified) election in Germany, in which the voter can select between five political parties. If, for example, a voter visits http://wahlen.nds.rub.de, then the voting server sends back a dynamically generated Web page in which the parties' acronyms are rendered as CAPTCHAs and visually presented to the voter in random order.


Bundestagswahl 2007 - Windows Internet Explorer
http://wahlen.nds.rub.de/
Google
Datei Bearbeiten Ansicht Favoriten Extras ?
Bundestagswahl 2007
Startseite (M) Feeds (J) Drucken Seite Extras
Internet
100%

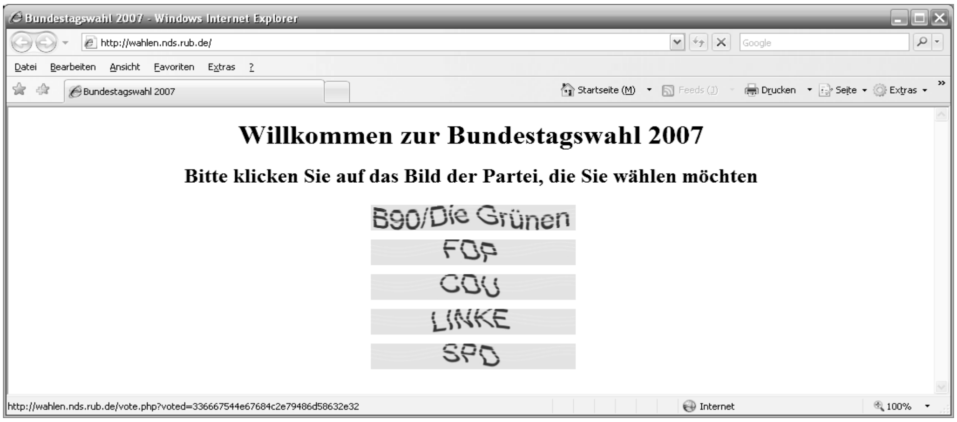
Willkommen zur Bundestagswahl 2007

Bitte klicken Sie auf das Bild der Partei, die Sie wählen möchten

CDU
B90/Die Grünen
FDP
SPD
LINKE

http://wahlen.nds.rub.de/vote.php?voted=705279376d724a6f56316f4b537a5047

Figure 1: First screen for CAPTCHA-based code voting


Bundestagswahl 2007 - Windows Internet Explorer
http://wahlen.nds.rub.de/
Google
Datei Bearbeiten Ansicht Favoriten Extras ?
Bundestagswahl 2007
Startseite (M) Feeds (J) Drucken Seite Extras
Internet
100%

Willkommen zur Bundestagswahl 2007

Bitte klicken Sie auf das Bild der Partei, die Sie wählen möchten

B90/Die Grünen
FDP
CDU
LINKE
SPD

http://wahlen.nds.rub.de/vote.php?voted=336667544e67684c2e79486d58632e32

Figure 2: Second screen for CAPTCHA-based code voting

Figures 1 and 2 illustrate two possible screens. If, in this example, the voter selected CDU on the first screen (choice 1), then the voting code sent to the server would be:

705279376d724a6f56316f4b537a5047.

Similarly, if the voter selected CDU on the second screen (choice 3), then the voting code would be:

336667544e67684c2e79486d58632e32.

In either case, the voting code represents a cryptographic hash value and is visible in the browser's status line. Note that the two codes are different and unlinkable despite the fact that the selected party is the same. Also note that in CAPTCHA-based code voting, there is no urgent need to minimize the length of the voting code. The voting codes are sent by the browser to the server in a way that is transparent to the user, i.e., the user does not have to type it in. This simplifies things considerably, and the discussion held at the end of Section 3 is obsolete in this setting. So from a usability perspective, CAPTCHA-based code voting is perfectly fine. The user experience does not significantly deviate from what he knows and is accustomed to. In the following section, we address the question whether CAPTCHA-based code voting is also fine from a security perspective.

5 Preliminary Security Analysis

If one considers the use of code voting to overcome the secure platform problem, then one is mainly concerned with the possibility of automated client-side attacks mounted by malware. More specifically, one wants to make it impossible for an adversary to write malware that can modify a vote in some meaningful way. This must be true even if the malware has access to all information that is available in the client's operating system or browser. Note, for example, that such malware has access to the browser's state and content of Web pages, and hence that it is able to read out the voting codes. But it does not know what code belongs to what choice, and hence it can only make random guesses. In the example given above, the malware is likely to be able to read out the voting code 705279376d724a6f56316f4b537a5047 for the first choice on the first screen, but it is not able to associate this code to the CDU party (because this association is done outside the client system in the brain of the voter). Consequently, it cannot decide whether this selection is the appropriate one, and hence whether it should modify the vote. Also, in the case of an election with more than two options, if the malware knew that it should modify the vote, it would still not know which other option to select.

The bottom line is that CAPTCHA-based code voting remains secure (in the sense sketched above) as long as the CAPTCHAs in use remain secure, i.e., cannot be solved by a machine. If somebody can write a piece of software that can break the security of the CAPTCHAs, then this software can also be used to trivially break the security of CAPTCHA-based code voting. So we have to make the critical assumption that the CAPTCHAs in use are secure. This assumption is critical, because the security of CAPTCHAs has come under fire and many researchers are trying to compromise them.

Based on the assumption that the CAPTCHAs in use are secure, one can argue that CAPTCHA-based code voting remains secure as well. But there are still a few subtle attacks that must be considered with care. Let us briefly elaborate on two examples.

1. If an adversary has introduced himself in the communication channel between the client and the server, then he is representing a man-in-the-middle (MITM) and can display any CAPTCHA or CAPTCHA-like image. It is then simple for him to circumvent or bypass CAPTCHA-based code voting (because he can create the CAPTCHAs and therefore knows what they represent). Consequently, the use of technologies and mechanisms that protect against MITM attacks seems to be mandatory. There are a few such technologies and mechanisms available; examples include ciphersuites for the TLS protocol that support authentication based on pre-shared keys [BH06], SSL/TLS session-aware (TLS-SA) user authentication [OHB08], the use of client-side public key certificates, and a few more. Unfortunately, these technologies and mechanisms are not yet widely deployed, and hence, any currently available infrastructure for remote Internet voting and CAPTCHA-based code voting is vulnerable to MITM attacks. It is best to make this vulnerability explicit.

2. Since an increasingly large number of e-commerce application providers employ CAPTCHAs to make sure that their users are human, an adversary could collaborate with these providers to exploit the human resources (and capabilities) of their users. For example, an adversary could set up a free Web-based CAPTCHA service for e-commerce application providers. If invoked, this service could use CAPTCHAs found on compromised client systems and provide them to the users of the service. The responses could then be used by the malware to modify the vote in some meaningful way. In the example given above, the malware would input the five CAPTCHAs found on the first screen to the service. The service would dispatch the CAPTCHAs to individual users, and return the strings representing the names of the parties to the malware. The malware would then be able to decide if and how to meaningfully modify the vote. There is hardly anything that can be done technically to protect against such a distributed attack. Consequently, one must carefully monitor the CAPTCHAs that are used by service providers, especially during the time frames of the elections and votes that are supported by CAPTCHA-based code voting. Too many occurrences of strings that represent political parties or names of politicians should be taken as an alert.

We think that both attacks are relevant and must be considered with care. In particular, we think that the use of technologies and mechanisms to protect against MITM attacks and a careful monitoring of CAPTCHAs in widespread use are mandatory in a real-world deployment of CAPTCHA-based code voting.

6 Conclusions and Outlook

The secure platform problem is severe for remote Internet voting. The malware-based client-side attacks that are currently mounted against Internet banking (e.g., [ORH08]) can easily be turned into attacks against remote Internet voting. The attack vectors are essentially the same, i.e., it does not matter whether malware manipulates an Internet banking transaction or a remote Internet voting transaction. In either case, the manipulation occurs after user authentication and can be made transparent to the user. This should be kept in mind when people argue about the (in)security of remote Internet voting.

Against this background, we think that code voting provides an appropriate technology to address the secure platform problem of remote Internet voting, but that it is not particularly user-friendly. There are different possibilities to implement code voting, and these possibilities have specific advantages and disadvantages.

In this paper, we proposed the use of CAPTCHAs to improve the user-friendliness of code voting, briefly discussed the security of CAPTCHA-based code voting, and elaborated on a possible implementation. CAPTCHA-based code voting can only be as secure as the CAPTCHAs that are used. Alternatively speaking, if an adversary is able to break the CAPTCHAs in use, then he is also able to break the security of CAPTCHA-based code voting. Consequently, the current state-of-the-art in breaking CAPTCHAs should be closely monitored and observed. For example, there is a recently published low-cost attack on CAPTCHAs employed by Microsoft[59]. In spite of the progress that has been made in order to break the security of CAPTCHAs, we still think that CAPTCHA-based code voting provides an interesting possibility to implement code voting in a real-world setting, and that it has potential for the future. It is certainly worthwhile to implement it, and to explore its use (and usability) in a field study.

[59] http://homepages.cs.ncl.ac.uk/jeff.yan/msn.htm

References

[Ber08] Beroggi, G.: Secure and Easy Internet Voting. IEEE Computer, Vol. 41, Number 2, February 2008, pp. 52-56.

[BH06] Badra, M.; Hajjeh, I.: Key-Exchange Authentication Using Shared Secrets. IEEE Computer, Vol. 39, Number 3, March 2006, pp. 58-66.

[Cal00] California Secretary of State, California Internet Voting Task Force, Final Report, January 2000, http://www.ss.ca.gov/executive/ivote/.

[Cha01] Chaum, D.: SureVote: Technical Overview. Proceedings of the Workshop on Trustworthy Elections (WOTE '01), August 2001, http://www.vote.caltech.edu/ wote01/ pdfs/surevote.pdf.

[DR06] Dierks, T.; Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.1, RFC 4346, April 2006.

[E+07] Elson, J. et al.: Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization. Proceedings of the 14th ACM Conference on Computer and Communications Security (ACM CCS 2007), 2007, http://research.microsoft.com/ asirra/papers/CCS2007. pdf.

[FKK96] Freier, A.O.; Karlton, P.; Kocher, P.C.: The SSL Protocol Version 3. Internet-Draft, 1996.

[Nao96] Naor, M.: Verification of a human in the loop or Identification via the Turing Test. 1996, citeseer.ist.psu.edu/naor96verification.html.

[OHB08] Oppliger, R.; Hauser, R.; Basin, D.: SSL/TLS Session-Aware User Authentication. IEEE Computer, Vol. 41, Number 3, March 2008, pp. 59-65.

[Opp02] Oppliger, R.: How to Address the Secure Platform Problem for Remote Internet Voting. Proceedings of the 5th Conference on “Sicherheit in Informationssystemen” (SIS 2002)}, Vienna (Austria), October 3-4, 2002, vdf Hochschulverlag, pp.~153-173.

[ORH08] Oppliger, R.; Rytz, R.; Holderegger, T.: Internet BankingClient-Side Attacks and Countermeasures. Submitted for publication.

[Riv01] Rivest, R.L.: Electronic Voting. Proceedings of Financial Cryptography '01, February 2001, http://theory.lcs.mit.edu/~rivest/Rivest-ElectronicVoting.pdf.

[Rub01] Rubin, A.D.: Security Considerations for Remote Electronic Voting over the Internet. Proceedings of the 29th Research Conference on Communication, Information and Internet Policy (TPRC 2001), October 2001, http://avirubin.com/e-voting.security.html.

[Tur50] Turing, A.: Computing machinery and intelligence. Mind, Vol. LIX, No. 236, October 1950, pp. 433-460.

[vA+04] von Ahn, L. et al.: Telling Humans and Computers Apart Automatically-How Lazy Cryptographers Do AI. Communications of the ACM, Vol. 47, No. 2, February 2004, pp. 57-60.

[Yao82] Yao, A.C.: Potocols for Secure Computations. Proceedings of 23rd IEEE Symposium on Foundations of Computer Science, Chicago, Illinois, November 1982, pp. 160-164.