RIES - Rijnland Internet Election System

very quick scan of published source code and documentation

Rop Gonggrijp

Willem-Jan Hengeveld, Eelco Hotting, Sebastian Schmidt, Frederik Weidemann

Stichting "Wij vertrouwen stemcomputers niet"
("We do not trust voting computers" foundation)
Linnaeusparkweg 98, 1098 EJ Amsterdam, The Netherlands

e-mail: rop@gonggri.jp

Abstract

The Rijnland Internet Election System (RIES) is a system for voting in elections over the internet. RIES processed around 90.000 votes in public elections in The Netherlands in 2004 and 2006. Based on total votes processed in public elections, RIES is one of the largest internet voting systems worldwide. As an interesting feature, RIES offers cryptographic end-to-end verifiability. This enables the voter to use cryptography to verify that her or his vote was counted as cast. On June 24th 2008, the source code to RIES was published. A rather cursory scan of this source code shows a significant lack of security-awareness among the programmers which - among other things - appears to have left RIES vulnerable to very simple attacks.
Given what we found in the scope of this quick study, it is very worrisome that a previous version of RIES has been actually used in the context of a real-world parliamentary election. Even if one believes - as the authors of this paper do not - that remote electronic voting over the internet is a good idea, it is clear that at the very least more attention should be given to secure programming. The fact that we discovered deficiencies of this magnitude in very a cursory examination of such a high profile voting system also raises questions regarding the RIES project and regarding the way in which governments are implementing electronic voting in general.

Introduction

This paper describes the result of a few days of looking at the source code and documentation of a rather complex internet voting system. This study began when the source code for RIES was published, on June 24th 2008. The first review version of this paper was available four days later on June 28th. This paper can by no means be understood as an exhaustive study. Such a study would require much more time and effort as well as an in-depth understanding of the inner workings of RIES.

Permission

Verifying some of the problems we found in the source code on the actual system without permission from the people operating RIES would probably be prosecutable as a computer crime. So in the early evening of Friday, June 27th we asked nicely and kindly got permission to attempt penetrating the RIES portal server at `https://portal.ries.surfnet.nl` from Simon Bouwman at "Het Waterschapshuis", a national cooperation of Water Boards that plans to operate the servers for the Water Board elections. He also kindly added one of our IP-numbers to the list of sites allowed to approach this server, a protection measure they were apparently just installing that very evening.

As a condition for getting permission, we accepted to print a brief reaction of "Het Waterschapshuis" along with our findings.

History of RIES

Water Boards

The Water Boards ("Waterschappen" in Dutch, sometimes also translated as "District Water Control Boards") are 27 different regional authorities dealing with water management in The Netherlands, a country that has long been highly dependent on a complex infrastructure of pumps and dykes to stay dry. They rank among the oldest democratic structures in The Netherlands. Since they are separate bodies of government the boards of these authorities are each directly elected by the people that live and/or own property in their area. These elections are typically postal elections, and

turnout is traditionally very low, often far below 30%.

RIES was developed by one of these Water Boards in conjunction with a number of private companies. It was used experimentally in the 2004 election by two of the Water Boards, and roughly 70 thousand voters cast their vote via the internet in that election.

2006 parliamentary election

RIES, in a version called RIES-KOA[1]. was also used for the 2006 national parliamentary election to augment the postal voting available to Dutch citizens residing outside of the Netherlands. (The Netherlands do not offer postal voting to voters not residing abroad.) Roughly 20.000 votes were cast using RIES in that election. Because the Netherlands have proportional representation, these votes were in effect added to the national totals for each candidate.

RIES-2008

A lot has happened with regard to electronic voting in The Netherlands in the past few years. The country was 100% e-Voting in 2006, and has since abolished all electronic voting in polling stations. Our previous research [1] into the security of the Nedap system in use in 90% of the precincts played an important part in the decision making process. The use of RIES for these low-turnout Water Board elections would make RIES the last remaining electronic voting system in use in The Netherlands.

The Water Boards would like to deploy RIES in its present incarnation, called RIES-2008, for the Water Board elections, which will all take place simultaneously in November 2008. The ministry of Transport and Water Works has drafted legislation allowing for this. Our foundation has lobbied with parliament for publishing the source code as well as for clear technical requirements and a procedure to formally test whether a proposed voting system meets these requirements. As a consequence the ministry created one and a half page of requirements [2] which for the greater part simply point to the recommendations issued by the Council of Europe [3]. The source code to RIES was published on the website www.openries.nl on 24 June 2008.

Claims regarding RIES

The www.openries.nl website lists a large number of documents on RIES. On the page 'What do others think?' we read (translated from Dutch):

> *Various prominent institutions have tested and positively evaluated RIES:*
>
> - *TNO Human Factors from Soesterberg tested usability of the voting interface;*
> - *A team of specialists from Peter Landrock's Cryptomathic (in Aarhus, Denmark) tested the cryptographic principles;*
> - *Madison Gurka from Eindhoven tested the server and network setup and security;*
> - *A team under supervision of Bart Jacobs (Radboud University Nijmegen) did external penetration tests.*

Scientists [4] [5] [6] [7] [8] as well as other parties [9] [10] have looked into various aspects of the design and/or security of (parts of) RIES.

Apart from purely scientific work, RIES has been subject to an accessibility test, a browser compatibility test, a modules test, a disaster recovery test, a functional acceptance test, a chain test, a regression test, a risk analysis, a security and usage evaluation, a server audit, evaluations of the various elections held with it and a report [11] to see how RIES matches up to the 112 recommendations of the Council of Europe with regard to e-Voting [3] and many, many more studies and reports.

Code examination

Duly impressed by the enormous amount of work on RIES security, we quickly browsed over the source expecting to find not even a hint of a single problem. As it turned out, we found quite a few problems, many of them rather serious issues. Below is a list of things we spotted in a first cursory look at the code.

XSS - Cross-Site Scripting

There are locations in the code where information supplied by the user is passed back on the page that is output by the system. For example we can see[2]:

```
document.location="start.jsp?elid=
  <%= request.getParameter("elid") %>";
```

[1] KOA stands for 'Kiezen Op Afstand', which is the Dutch government's remote e-voting project

[2] `riesvotewin_source_v1.0/admin/index.jsp`, line 29

as well as[3]:

```
<c:set var="section" value="<%=
  request.getParameter("section")%>"/>
```

This probably means that when a user's browser can be made to view a URL, an attacker can execute Javascript statements within the context of a user session on the election site. In the case of RIES, the cryptographic routines that perform the actual act of voting are implemented as client-side Javascript, making it impossible for a users to protect themselves against such attacks by turning off Javascript.

We found out that we were not the only ones who had spotted this problem. As mentioned above, the RIES website lists an impressive number of studies into various aspects of the system. Among them is a report by GOVCERT.NL, the Dutch government computer emergency response team. They found this problem and reported it in September of 2006 when they did a 'web application scan' [12]. They found the same '`elid`' variable to be vulnerable, and recommended that the input and parameters be validated to eliminate the risk of Cross-Site Scripting. They also ominously said (translated from Dutch):

> *REMARK: The lack of sufficient input validation can also lead to vulnerabilities such as SQL-injection which are more serious in nature. During the scan we have not found any such vulnerability.*

We are surprised that the makers of RIES proudly present a two year old report of a quick scan on their website without having implemented the recommendations contained within.

Random tokens

The code contains a method of authenticating a user via her/his mobile phone. The code calls this challenge/response, however it is technically a response only. When a user wants to log in, the system generates a random password which is sent to the user via SMS. The user must than enter this password via the internet. Below is the piece of code[4] that does the actual generation of that 'random' token:

```
Random rGen=new Random(new Date().getTime());
String sResult="";
int i=0;
while(i<6) {
    sResult+=rGen.nextInt(10);
    i++;
}
```

Random() will present the same output when given the same output of `Date()` and `getTime()`. Even though the latter is in milliseconds, an attacker would only need a few thousand guesses to figure out the key sent to a phone that she does not own. Since the code does not prevent someone from trying a few thousand tokens, this would not prevent the attacker from gaining access.

Note that an attacker can probably acquire a very accurate idea of system time from the http headers provided by the system or, if the goal is privilege escalation, from the token received by SMS following a valid login.

SQL injection

In 2006, GOVCERT.NL had warned RIES: if the programmer doesn't check the inputs to his/her code, the program may end up vulnerable to SQL injection attacks. During the interaction with the program, a user typically enters all sorts of text strings, such as her username when prompted like this:

SQL queries involving user-supplied information in the RIES source code are **all** generated by simply inserting whatever the user entered into a query, without any checking. One of the queries that follows is the one where the code associated with the login box above tries to find the telephone number for a user to send a special SMS token to allow that user to log in[5]:

```
sbBuffer=new StringBuffer();
sbBuffer.append("select PHONE from OPERATOR
  where OPERATOR_ID='"+sUsername+"'");
oRs=oStmt.executeQuery(sbBuffer.toString());
```

---

[3] `riesvotewin_source_v1.0/admin/sectionlinks.jsp`, line 3

[4] `riesportal_source_v1.0.zip/WEB-INF/src/java/org/openries/portal/jaas/JAASHelperServlet.java`, line 280

[5] `riesportal_source_v1.0.zip/WEB-INF/src/java/org/openries/portal/jaas/JAASHelperServlet.java`, line 347

As is visible from this code, the SQL statement to be processed by the database server is formed with the `sUsername` string. The code does not contain anything to sanitize that string first. If one enters `rop` in the username box the query to the SQL server would become:

```
select PHONE from OPERATOR where
  OPERATOR_ID='rop'
```

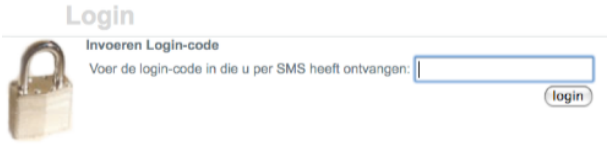
Since the program finds no corresponding entry in the `OPERATOR_ID` table it outputs 'login failed':

However, we can make the SQL statement succeed by entering a string as follows:

The resulting SQL statement now looks like:

```
select PHONE from OPERATOR where
  OPERATOR_ID='rop' OR 1=1; --
```

And as a result we now get:



The system has apparently sent the special 'random' access code (for problems with this code see above) via SMS. Since the SQL statement succeeded on the first user, we suspect this user will have received the SMS.

To actually enter the system and prove further vulnerabilities, one needs to play around a little more. We were still experimenting with this when the system suddenly said:

A few minutes later it said 'technical problem':

And then it finally said 'closed for the weekend':

We guess that since we were testing on a Friday night, indeed the system could be down for the weekend. It did however briefly re-open on the following Saturday, but after a few attempts it was again 'closed for the weekend'.

Exception handling

Often when exceptions are handled in the code, a message is logged, but no action is taken. For example, in `_sendResponseSMS`[6], the exception handlers are (in pseudo-code) very often structured like the one in `sendRessponseSMS`:

```
_sendResponseSMS()
{
try {
 executeQuery
 if result {
   try {  sendsms }
   catch(all) { return false }
 }
 else { return false }
}
catch (sqlexception) { logmessage }

return true
}
```

Since the code contains no '`return false`' with the `catch(sqlexception)`, an exception from the `executeQuery` will still cause the calling function `sendResponseSMS` to succeed and cause the server to display the 'enter-SMS-response' page. One of the possible reasons why `executeQuery` would throw an `sqlexception` would be a syntax error in

[6] `riesportal_source_v1.0.zip/WEB-INF/src/java/org/openries/portal/jaas/JAASHelperServlet.java`, line 332

the SQL statement, for instance, caused by an SQL-injection attempt. But since SQL syntax errors are ignored, it becomes much easier to construct SQL-injections. An attacker can target specific SQL statements, without the need to keep all the other statements free of errors.

The problem is that RIES tries to trap exceptions from the library functions, and translate them in true/false but often fails to do a '`return false`'.

In the case of the SQL exceptions, it might be even better not to trap these exceptions at all, but let the JSP server handle this. In most cases an exception should be a reason to abort any pending operations, not to 'log message, and continue'.

Problem deriving key?

Then there's a part in `org.openries.ripocs.config.ConfigManager` where the code is apparently retrieving a stored 'salt' value from a file to create , through XOR, a smartcard key of some sort. The final lines of the code[7] that is supposed to be generating the value are:

```
// derive AbelPiKey (16 bytes)
return PKCS5.PBKDF1(sPassPhrase, abSalt,
  PKCS5_ITERATIONS, 16);
```

However, the entire function is commented out and instead it now reads:

```
//temp for ketentest 1 because of existing
  keys in smartcards
return Utils.stringToHex ("0123456789ABCDEF
  FFFFFFFFFFFFFFFF0123456789ABCDEF");
```

We're not sure whether this presents an actual security problem and what the magnitude of the problem would be without learning much more about what's going on here. We'll leave it at saying this looks rather suspicious in production code.

Other issues and comments

The code exposes a development CVS server that appears to be running on a regular home ADSL connection[8]:

```
:pserver:arnout@cozmanova.xs4all.nl:4202/
  usr/local/cvs-ries-rep
```

A public mail server is used[9]:

```
private static String EMAIL_SERVER =
  "smtp.xs4all.nl";
```

It is not clear under what circumstances the system sends mail and whether one could perform attacks if one could destroy, intercept, modify or interject batches of these e-mails. The concept of phishing for voting credentials comes to mind.

The code also contains an SMS gateway with a valid account[10]:

```
private String _sServiceURL="https://
  secure.mollie.nl/xml/sms/";
private String _sUsername="mdobrinic";
private String _sPassword="riesdemo";
private String _sGateway="2"; // development
  default; 1=more reliable; 2=cheaper
```

Assuming the authors want their voting system to be optimized for 'more reliable' as opposed to 'cheaper', the setting of the `sGateway` value shows how easy it is for undesirable development artifacts to make it into production code.

Fundamental issues

Limited security against insiders

Elections like the ones performed with RIES legally require secrecy of the vote. In RIES this requires the operators to destroy information they held at some stage during the process. If anyone manages to hold on to this information the publication of a verification file at the end of the election allows whoever has this information to tie every vote to an individual voter. Hubbers et al [6] also conclude that the cryptography used in RIES offers no protection against insiders.

> *RIES is built on certain cryptographic primitives, like one-time signatures. Keys for individual voters are generated centrally. There are no anonymous channels. The structural protection and safeguards offered by cryptography are therefore rather limited. Many of the guarantees in RIES thus rely on organizational controls, notably with respect to (voter) key generation, production of postal packages, insider attacks (especially at the server), integrity and authenticity of the software, and helpdesk procedures.*

The CIBIT rapport [7] concludes (translated from Dutch):

> *Vulnerability of the STUF-C10 file, all temporary variants hereof and KGenVoterKey.*

---

[7] `riesripocs_source_v1.0.zip/src/org/openries/ripocs/config/ConfigManager.java`, line 23

[8] `rieslogin_source_v1.0.zip/org/CVS/Root`

[9] `riessystem_source_v1.0.zip/source/org/openries/system/messaging/EmailMessage.java`, line 52

[10] `riessystem_source_v1.0.zip/source/org/openries/system/messaging/SMSConfig.java`, line 22

Using the STUF-C10 file one can influence the election and break vote secrecy. These objects need to be destroyed as soon as the necessity for the presence of these objects expires.

Compared to a postal election performed in accordance with proper procedures, one must conclude that violating vote secrecy on a massive scale is now in the hands of one or at least very few individuals.

Household PCs assumed secure

Hubbers et al [6] also describe a central assumption during the design of RIES:

RIES assumes that the voter's PCs are secure. Attackers may however employ malware or even 'man-in-the-browser' attacks to capture voter's PCs. Powerful attackers may thus change votes, and so this involves a unique potential risk for Internet elections.

Given the prevalence of attacks against client PCs , for example with regard to electronic banking, it would seem inevitable for attacks to appear once electronic voting becomes common. The fact that candidates have apparently tried to submit faked signatures to be on the Water Boards in the past[11] proves there is an apparent potential for fraud regarding these elections.

Conclusions

The scope of this paper is not to completely understand the RIES system but only to outline a number of immediately-visible security problems. As a reality check, we are happy to have proven that SQL injection actually works on the live system. Further examination of RIES, including actually attempting to disturb/manipulate elections would likely require further study of the inner workings of RIES and is beyond the scope of this first examination.

We are amazed to find a system so apparently well-studied yet so fundamentally and undeniably insecurely programmed. Computer security appears not to be part of the mindset of the people programming RIES. For example, in the case of the SQL-problems, it would have been better to use `prepareStatement` in addition to sanitizing user input. Scientific studies of RIES seem to have concentrated on the more scientifically 'sexy' theoretical security offered by the inventive cryptographic protocols while largely ignoring the threats posed by very straightforward and down-to-earth attack methods that are much more likely to be used in the real world.

To create a system that appears secure, there are two approaches. The proper approach is to design a system with security in mind. The other approach is to retrofit an insecure system with security-measures that make a system look secure but which in fact add little security. Such measures are usually intended to impress onlookers. There are situations where adding an SMS-token like the one used in RIES is a useful addition to other security measures. However in combination with the proven lack of security awareness during implementing the system, RIES' SMS-token appears to fall in the impress-the-onlookers category.

The www.openries.nl site says: "*Various prominent institutions have tested and positively evaluated RIES"*. This research shows one must be very wary if scientific and other studies into some part of a not yet published and changing system are used to implicitly claim the entire system is secure.

No amount of voluntary studies of some part of a voting system - often paid for by stakeholders wanting to see the system in use - can ever replace clear and stringent government-imposed requirements that include independent source code review. Such reviews need to pragmatically and 'holistically' look for security problems as well as test the code against more formal coding standards and practices.

In their 2004 article 'Stemmen via Internet geen probleem' [5] Hubbers and Jacobs 'vote in favour' of use of this system when they state (translated from Dutch):

Summarizing, [RIES] is a relatively simple, original and understandable system that has been implemented with the appropriate care and transparency. [...] When the use of RIES during these Water Board elections is involved, we clearly vote in favour!

We pose that RIES has clearly not been implemented with 'appropriate care'. Given the dependence of society on their judgment, scientists should probably refrain from endorsing electronic voting systems at least until the entire system has been examined.

The Water boards need to be commended for the publication of source code and all documentation relevant documents as well as for allowing outside researchers to study the security of the system. It seems that although they are struggling with

[11] Hoogheemraadschap van Rijnland, Bestuursverkiezing Rijnland moet door fraude deels over, Persbericht 12 november 2004, `http://www.rijnlandkiest.net/asp/get.asp?xdl=../views/rijnlandkiest/xdl/Page&ItmIdt=00001440`

obvious and serious code quality problems, they are at least trying to do the right thing.

The amount of problems we found, as well as the class of problems, mean RIES must undergo much more testing and quite possibly partial code rewrites. Use of this code base in real-world elections this year would seem highly irresponsible. A particularly dangerous - yet expected - reaction to this study would be to quickly fix the problems we found, pretend RIES is now sufficiently secure and use it in elections.

The Dutch government had to be forced by a majority in Parliament to develop any standards at all for internet voting. The resulting minimalist legal requirements [2] (covering a whole page and a half) or the recommendations of the Council of Europe [3] that these requirements point to contain absolutely nothing that would prevent problems of this magnitude.

The security of RIES is highly dependent on proper procedures to be followed. Given that the whole point of an election is to be able to independently verify the outcome as well as the secrecy of the vote, protection needs to be strong against insiders as well as against outsiders. The common notion that insiders are somehow more trustworthy does not match the reality of election problems worldwide. This makes RIES, and many systems like it, unsuitable for use in public elections. The assumption of a secure PC in every voter's home simply does not match reality.

Recommendations

Recommendations to government

RIES as it is should not be used for any elections. We strongly believe the system in its present state does not meet any imaginable responsible criteria for a system of this importance. Even if all our recommendations were followed, we feel the fundamental problems listed in this paper mean RIES cannot be used in elections that require secrecy of the vote.

The Water Boards must not be believed if they say RIES can be quickly fixed. The problems described in this paper point to a serious lack of security awareness at the time RIES was programmed. The vulnerabilities found in this quick study sufficiently warrant thorough and independent study to determine whether the current code base of RIES is suitable at all for use in elections.

For critical applications such as election systems, responsible coding standards and other criteria need to be developed and independently tested against. No systems should be approved or used in elections until such tests are part of the applicable legal requirements.

The fundamental shortcomings of RIES and systems like it need to be given more weight. Certain attacks, such as breaking the secrecy of the vote for the entire population, are much harder to perform in a postal system. RIES offers very limited protection against insider attacks, which in our view is not appropriate for public elections.

Recommendations regarding RIES

We feel that fixing RIES can only be of scientific interest. Even if one were to implement the following recommendations:

All problems described in this document need to be fixed. In some parts, re-writing the code may make more sense than trying to 'retrofit' security.

Further problems need to be identified and fixed. We did the most cursory of examinations without a deep understanding of the interactions between the different parts of RIES. Given these circumstances, the probability that we spotted everything that would need to be spotted is very small.

The apparent management issues that led to these problems need to be addressed. Who hired programmers that put unchecked user-supplied strings into SQL-queries? Who organized code quality assurance? How come the XSS problems pointed out by GOVCERT.NL almost two years ago were not sufficiently addressed?

The resulting system would still not be suitable for use in elections because it is based on overly optimistic assumptions regarding the security of home PCs and (more importantly) because it fundamentally lacks adequate protection against insiders.

Acknowledgements

The authors would like to thank Simon Bouwman of "Het Waterschapshuis" for giving permission to test some of our attacks against the system. Furthermore they would like to thank the entire crew of "Wij vertrouwen stemcomputers niet" for their insight and for helping in proofreading this paper.

Reaction Het Waterschapshuis

As stated in the introduction, we agreed with "Het Waterschapshuis" to include a brief reaction with our findings. They responded as follows:

An advantage of open source is that the code can be reviewed and improvements of the code can be made. The published code of RIES is not yet the production code. Internal reviews and tests have to be made. Recommendations from external - as in this paper - are welcome. New versions of the code will be published in the coming weeks.

As mentioned in the paper, RIES is a rather complex (internet) voting system. RIES contains several sub-systems. Each sub-system is a combination of software, configuration and administrative procedures. Each sub-system has very different tasks and settings and also different security requirements. For instance, the VoteWindow is the only sub-system which is public to the world-wide Internet. All other sub-systems are limited access only, not accessible through the internet. The RIES-Portal access will be controlled by VPN, RIES-RIPOCS is only accessible via RIES-Portal, and RIES-ROCMIS is an offline machine used within a proper set of administrative procedures. Therefore, RIES cannot be evaluated from source code alone to measure the security strength.

Keep in mind; this part is NOT production code yet. Many of the issues are related to proper input validation. And we agree that proper input validation is required and we will fulfill that requirement. Mainly Struts input validation mechanism will be used. In the published source packages and the development system investigated, the feature was not switched on for development reasons. Therefore again: we're in a state of functional sequence test (ketentest) and not yet in production.

The original response was slightly longer and added a list where each issue we found was discussed separately. Since this was a little too long to be included in this paper, we agreed to include a link to the full response, which will be available on the RIES website at `http://www.openries.nl/wvsn-paper` .

There is a lot to be said regarding this reaction, but that would turn this paper into a discussion forum. Suffice it to say that we stand by our original conclusions and recommendations and that the debate on whether or not RIES is suitable for public elections is very likely to continue.

References

[1] Rop Gonggrijp and Willem-Jan Hengeveld, Studying the Nedap/Groenendaal ES3B voting computer, a computer security perspective, 2007, Proceedings of the USENIX/Accurate Electronic Voting Technology workshop, `http://portal.acm.org/citation.cfm?id=1323112`

[2] Ministerie van Verkeer en Waterstaat, Regeling waterschapsverkiezingen 2008, 15 mei 2008/Nr. CEND/HDJZ-2008/587, Staatscourant 23 mei 2008, nr. 97 / pag. 11, `http://www.wijvertrouwenstemcomputersniet.nl/images/e/e7/SC85731.pdf`

[3] Council of Europe, Recommendation Rec(2004)11 of the Committee of Ministers to member states on legal, operational and technical standards for e-voting, 2004. `https://wcd.coe.int/ViewDoc.jsp?id=778189`

[4] E.-M. Hubbers, B. Jacobs, and W. Pieters. RIES - Internet Voting in Action. In R. Bilof, editor, COMPSAC'05, Proceedings of the 29th Annual International Computer Software and Applications Conference, COMPSAC'05, pages 417-424. IEEE Computer Society, 2005. 26-28 July 2005, `http://www.cs.ru.nl/~hubbers/pubs/compsac2005.pdf`

[5] E.-M. Hubbers, B. Jacobs. Stemmen via internet *geen* probleem, Automatisering Gids #42, 15 October 2004, p.15, `http://www.openries.nl/aspx/download.aspx?File=/contents/pages/77743/stemmenviainternetgeenprobleem.pdf`

[6] Engelbert Hubbers, Bart Jacobs, Berry Schoenmakers, Henk van Tilborg and Benne de Weger, Description and Analysis of the RIES Internet Voting System, 24 June 2008, `http://www.win.tue.nl/eipsi/images/RIES_descr_anal_v1.0_June_24.pdf`

[7] Ir. Jaap van Ekris, CIBIT, Beoordeling KOA, Een beoordeling van de integriteit van "Kiezen op Afstand", 11 September 2008, `http://www.openries.nl/aspx/download.aspx?File=/contents/pages/77743/eindrapportcibit.pdf`

[8] Nijmegen University - Security of Systems, Server Audit van RIES, 23 July 2004, `http://www.openries.nl/aspx/download.aspx?File=/contents/pages/77743/reportkun.pdf`

[9] Jens Groth, CryptoMathic, Review of RIES (v 0.3), Cryptomathic A/S, 21 January 2004, `http://www.openries.nl/aspx/download.aspx?File=/contents/pages/77743/reviewofries.pdf`

[10] Lucas Kruijswijk. Internetstemmen met RIES onder de loep, 2006. `http://www.wijvertrouwenstemcomputersniet.nl/Internetstemmen_met_RIES_onder_de_loep`

[11] Aanbevelingen van de Raad van Europa, Evaluatie voorziening internetstemmen RIES, conform artikel 5 onderdeel b Regeling waterschaps-verkiezingen 2008, version 6, June 2008, `http://www.openries.nl/aspx/download.aspx?File=/contents/pages/77726/evaluatieaanbevelingenraadvaneuropa.pdf`

[12] GOVCERT.NL, Webapplicatie-scan, Kiezen op Afstand, 1 September 2006, `http://www.openries.nl/aspx/download.aspx?File=/contents/pages/77743/webapplicatie-scan.pdf`