Modifying an Off-the-Shelf Wireless Router for PDF Ballot Tampering

Daniel M. Zimmerman and Joseph R. Kiniry
Galois, Inc.
421 SW 6th Ave., Suite 300, Portland, OR 97204

November 7, 2014

Abstract

In order to highlight the dangers associated with Internet voting carried out over electronic mail with PDF forms, we show that an off-the-shelf home Internet router can be easily modified to silently alter election ballots. The modification is nearly undetectable and can be carried out in a way that leaves no evidence to be found in a post-election investigation.

1 Introduction

A number of governments, at various levels, have expressed interest in the establishment of Internet voting systems. Examples include the state of Alaska, which has carryed out an Internet voting trial for the 2012 and 2014 elections; Washington, D.C., which in 2010 developed an Internet voting pilot project with the OSET Foundation for absentee voters that was successfully attacked by an academic research group [6]; and the nation of Estonia, which has had Internet voting since 2005 that exhibits significant security flaws [3].

One mechanism proposed for Internet voting involves ballots rendered as standard Adobe Portable Document Format (PDF) forms. These forms are made available to voters on a web site; the voters then use standard software (e.g., Acrobat Reader, Preview, etc.) to fill out the forms and submit the completed forms via electronic mail to the appropriate election authority. The submitted ballots are then printed and counted, either by hand or with optical scanners.

This ballot-return mechanism was used as a fallback mechanism after the Washington, D.C. Internet voting system was successfully hacked and removed from service, it was used as an emergency measure in New Jersey elections in 2012 because of the impact of superstorm Sandy, and it is also available to current Alaska voters.

Unfortunately, this mechanism is vulnerable to fairly obvious attacks at several levels: malicious software on the user's computer could modify or invalidate a vote; a malicious election authority could intentionally miscount (or simply "lose") received votes; malicious third parties could masquerade as the election authority or perform denial of service attacks against the actual election authority to prevent votes from being cast or flood them with invalid ballots; and more. Here we describe a more subtle attack at the transport level, which changes the raw data traveling through the electronic mail system between the voter's computer and the election authority.

2 The Attack

Our attack transparently and untraceably alters a ballot to change a voter's choices once the ballot has been sent as an email attachment to the election authority. The end result is that the voter believes he has submitted a vote for some candidate, while the election authority receives a legitimate-looking vote for another candidate.

2.1 PDF Ballot Alteration

The goal of our attack is to modify a PDF file such that it reflects a different choice than the one made by the voter. We achieve this by directly modifying the data within the PDF file to change the vote.

In a PDF form with radio buttons, each button has an object identifier of one or more characters. Suppose the identifier for the selected object is "`ABC`". The selected button is then indicated in the PDF file with two strings: one of the form "`<< /V /ABC`" and one of the form "`/AS /ABC`". We call the "`/V`" and "`/AS`" strings the *vote string*

PDF Viewer	Test (1)	Test (2)	Test (3)
Adobe Acrobat Pro XI	Changed	Changed	Changed
Apple Preview	Changed	Changed	Changed
Google Chrome	Original	None	Changed
Gmail (Any Browser)	Original	None	Changed
Mozilla Firefox	Original	None	Changed
Safari	Changed	Changed	Changed
Skim	Changed	Changed	Changed

Table 1: A summary of our testing of various PDF viewers (on Mac OS X 10.10) with modified ballot PDFs. For tests (1), (2) and (3) we changed only the vote string, the vote string and the original selection string, and the vote string and both selection strings, respectively. "Original" indicates that the voter's original vote was shown after our modifications; "Changed" indicates that our changed vote was shown after our modifications; and "None" indicates that no vote was shown after our modifications.

and *selection string* respectively. There is a single vote string per set of buttons and one selection string for each identifier in the set; the latter appears as "`/AS /Off`" for each unselected button.

If we want to change a vote for the candidate with identifier "`ID1`" to a vote for the candidate with identifier "`ID2`" in a way that convinces all the PDF readers we have tested, we must replace the vote string "`<< /V /ID1`" with a new vote string "`<< /V /ID2`", replace the selection string "`/AS /ID1`" with a new selection string "`/AS /Off`", and replace the appropriate selection string for "`ID2`" (which was "`/AS /Off`" in the original file) with "`/AS /ID2`".

In fact, our testing revealed that it was not necessary to make *all* these changes in order to convince standalone PDF viewers of our changed vote. As shown in Table 1, we tested (1) changing only the vote string, (2) changing the vote string and changing the selection string for the previous selection to "`/Off`", and (3) changing the vote string, changing the selection string for the previous selection to "`/Off`", and changing the selection string for the new selection appropriately. All three standalone PDF viewers, including Adobe Acrobat (presumably the reference implementation for PDF form processing), showed our modified vote in all three cases; the Chrome, Gmail and PDF.js viewers, all of which are likely based on the same JavaScript implementation, showed our modified vote only when we changed all three strings and showed either the original vote or no vote at all in other cases.

Recall that election authorities print the submitted PDF ballots and count them on paper. In all our tests, the printed output from a PDF viewer matched its screen output. It is extremely likely that an election authority would use a standalone PDF viewer to print the votes, rather than printing them directly from a web browser, as they would hopefully not use a web-based email account to receive the votes. Since all the standalone PDF viewers we tested were convinced by a single string change, it is likely that a single string change (per targeted ballot) would be sufficient to change an election outcome. However, for completeness, we actually change all three strings in our implementation of the attack.

There is an additional subtlety regarding the lengths of identifiers. Since we carry out our attack on information as it is transmitted over the Internet, the attack is significantly simplified if we can modify the PDF file without changing its length (more specifically, without changing the length of any of the individual packets that comprise it).

Consider the case where one candidate (Candidate A) has identifier "`Aardvark`" and the other (Candidate Z) has identifier "`Zebra`". If we want to change a vote for Candidate A into a vote for Candidate Z, we replace the string "`<< /V /Aardvark`" with the string "`<< /V /Zebra␣␣␣`"; the 3 spaces at the end of our replacement string pad it to the same length as the original string. These extra spaces are ignored by PDF readers; though they might be noticed in a close inspection of the file, the result is still valid PDF and could conceivably have been generated when the voter initially filled out the form.

On the other hand, if we want to change a vote for Candidate Z into a vote for Candidate A we must perform multiple string replacements to change Candidate A's identifier throughout the file. First, we pick a new identifier to represent Candidate A with the same length as (or a shorter length than) the identifier for Candidate Z. In this case, perhaps we choose the identifier "`Aard`". Next, we replace the string "`<< /V /Zebra`" with the string "`<< /V /Aard␣`", and also replace two occurrences of "`<< /Aardvark`" later in the file with "`<< /Aard␣␣␣␣`". As before, the extra spaces pad the strings to the same length as in the original file and are ignored by PDF readers.

While replacing a chosen identifier with a shorter one is difficult to notice, replacing a chosen identifier with a longer one (and the resulting wholesale identifier change required to make the new identifier fit in the same number

of characters) is easy to spot. However, in any given PDF form, the identifiers are most likely already the same length. For example, the default identifiers assigned to buttons in a group when creating a form in Adobe Acrobat Professional are "`Choice1`", "`Choice2`", etc. If the attack is carried out in a way such that the selection strings are changed along with the vote string, the identifier for the candidate we choose must be limited to at most three characters so as to be able to replace "`Off`" in a selection string.

2.2 Implementation

The attack is carried out by modifying one or more TCP packets of the email attachment somewhere along its route between the voter's email client and the election authority. The modification replaces the byte sequences inside the PDF file that indicate a selected form element, as described above.

Since the PDF file is transmitted as an email attachment, the actual strings replaced are Base64 encoded versions of the strings within the PDF file. Base64, described in RFC 3548, [5] is the standard encoding mechanism for sending binary files as email attachments. It converts every three bytes of the binary file to four 6-bit fields. Each of these fields can take one of $2^6 = 64$ possible values, and can thus be sent as a single printable character chosen from a 64-character alphabet.

Using our previous "candidates", we see that the Base64 encodings of "`<< /V /Aardvark`" and "`<< /V /Zebra␣␣`" are "`PDwgL1YgL0FhcmR2YXJr`" and "`PDwgL1YgL1plYnJhICAg`", respectively; we compute similar encodings for the other strings that need to be changed. Depending on the offset in the PDF file at which the target string appears, the encoding actually present in the email might be different from that computed for the string in isolation. If the first character of the target string is the first in an encoded group of 3 characters, direct encodings like the ones shown above are correct (note that each of our target strings has a length, 15, that is a multiple of 3); however, if the first character of the target string is the second or third in an encoded group, or if the target string has a length that is not a multiple of 3, the encoding depends on the previous or subsequent characters in the PDF file. We use knowledge of the actual structure of the PDF ballot under attack to compile suitable encoded strings into our attack code; however, it is easy to imagine a more robust implementation that performs the Base64 decoding and re-encoding on-the-fly and therefore does not depend on known character offsets in the PDF file.

Our attack code monitors connections on standard email submission ports, replacing our target encoded strings with our desired replacement strings when they are encountered. The replacement is performed by modifying individual packets at the TCP protocol level, and cannot be detected in real time unless the connection is actively monitored at both ends to ensure that packet contents are transferred unmodified. It is possible that we get unlucky, for a particular ballot/email client/network combination, and our target strings end up split across TCP packets; in such cases our attack cannot modify the ballot (or may partially modify it, depending on which strings are split across packets). However, given typical TCP packet sizes and the relatively short lengths of the target strings, it is likely that our attack can successfully modify most ballots it encounters. Additionally, implementing an attack capable of performing the update across packets would not be technically challenging.

2.3 Deployment

For this demonstration, we chose to attack an off-the-shelf home wireless router purchased at a local big-box electronics retailer. Nearly all such routers on the market today are based on embedded versions of the Linux operating system and therefore, in accordance with the GNU General Public License, the source code for their firmware is freely available. We downloaded a complete distribution of our router's firmware source code and made a small modification to the kernel code that handles transmission of packets on network devices. The modification has fewer than 50 lines of code, of which about 1/3 comprise a simple (and completely unoptimized) string search and replace algorithm. We then built new firmware that, to any observer who does not perform detailed analysis of its traffic handling or a detailed inspection of the compiled code, is completely indistinguishable from the manufacturer's official firmware. It behaves identically to the manufacturer's firmware except in two respects: first, TCP connections to ports 25 and 587 (the standard email submission ports) are slower than on the manufacturer's firmware; second, particular sequences of bytes sent to these ports are replaced with different sequences of bytes.

To ensure that our attack had minimal impact on router performance, we ran several benchmarks to gauge its effect. On TCP connections to ports 25 and 587, we observed a slowdown of approximately 25%. This would definitely be a noticeable performance hit, but all commonly-used email clients send mail asynchronously in the background and end users typically do not monitor the speed of their outgoing email. Thus, we believe that it would either not be noticed in practice, or would be attributed to general network or server performance issues. On all other TCP connections, and with UDP-based protocols, there was effectively no impact on performance; in a few cases performance *increased* slightly with our modified firmware, probably due to compiler optimizations or other effects of building the firmware in a different environment than the manufacturer. Thus, for the applications where the end

user actually cares about Internet performance—web browsing, streaming video, file downloads—there is effectively no change in behavior.

In order to actually install the modified firmware on an end user's wireless router, we can take advantage of one of many security vulnerabilities. One common vulnerability is that most home routers ship with Universal Plug and Play activated; there are well-documented ways to take advantage of this to run arbitrary code on several Linux-based routers [2]. Some home routers are also vulnerable to "magic packets" sent to particular ports that enable administrator-level backdoors, while others have remote administration enabled by default with typically weak default passwords.

Some wireless routers, including the one we purchased, have a built-in firmware update mechanism that notifies users when new firmware is available and downloads it for them. On the particular router we purchased, the firmware update mechanism has effectively no security protections; it simply connects to an FTP server at a specified hostname, downloads a firmware information file, determines whether the firmware is new, and then downloads the firmware from the same FTP server. While the router does compute an MD5 checksum of the downloaded firmware and compare it to the checksum in the firmware information file, the fact that both the checksum and the firmware come from the same source means that DNS-based spoofing of the firmware upgrade server can easily fool the router into installing our modified firmware. The same is true for firmware update mechanisms from other manufacturers; for example, it was recently publicized that ASUS RT Series routers are vulnerable to man-in-the-middle attacks against their software update mechanism for much the same reason [4].

Another, more brute-force, technique is to simply "war drive" through neighborhoods where a campaign wants to change the vote, taking advantage of the default weak passwords on wireless networks and wireless router administrative interfaces that the vast majority of end users never change. Finally, social engineering attacks, either via DNS spoofing or via authentic-looking email notifications, are another promising avenue for placing modified firmware on users' home routers.

Once the modified firmware is installed, regardless of the method used, it is straightforward for it to (1) ensure that it remains installed until after the user has voted (in the absence of a hardware reset or other drastic action by the user), by hiding firmware update notifications and ignoring explicit update requests, and (2) eliminate evidence of tampering after the election, or after the user has voted, by replacing itself in the middle of the night with "good" firmware downloaded directly from the manufacturer's FTP or web site. We did not implement this functionality, but a real attacker would certainly do so.

3 Possible Mitigation Strategies

There are several ways to mitigate the effect of this attack, but all are impractical because of the difficulty in patching the large existing population of home routers, the fact that they would require unreasonable amounts of technical knowledge from voters, or inherent properties of the current electronic mail infrastructure.

Encrypted/Signed PDF. Perhaps the most obvious way to prevent the type of attack we have described here is to sign or encrypt the PDF file before emailing it. The PDF standard provides a built-in password protection system that, when properly used, gives a good level of cryptographic security. However, it is unlikely that voters would use the mechanism properly, and some PDF readers are incapable of using it at all. There are also clear issues with key and password distribution, especially if we already have control of users' home Internet routers; it is as easy to include a man-in-the-middle attack against the election authority in the router firmware as it is to transparently change PDF files in transit to email servers. For this same reason, requiring that the PDF files be submitted to a secure website also does not prevent vote tampering.

Encrypted SMTP. An encrypted connection to the SMTP server would prevent our attack on end user home routers from changing the PDF file in transit (unless our firmware also performs a man-in-the-middle attack against the SMTP server). However, this mitigation would do nothing to prevent the same attack from being carried out as the email passes through a backbone router somewhere between the end user's SMTP server and the election authority's SMTP server, as the vast majority of server-to-server SMTP connections are unencrypted. The discussion of exploits that can be carried out against backbone routers is beyond our scope here, but vulnerabilities that would allow us to carry out a similar attack to the one described here clearly exist; for example, many routers ship with GNU Bash installed and are therefore vulnerable to the recently-disclosed Bash environment variable command injection flaw.

More Secure Firmware Update Mechanism. One measure that could help, by eliminating one avenue of injection for the corrupted firmware, is to tighten the security of routers' firmware update mechanisms. This could be done in the standard way, with security certificates and SSL connections. However, this would leave open a number

of other avenues of firmware installation, including security vulnerabilities such as those we have discussed above, and would have no impact on most of the existing base of installed routers, which number in the millions.

4 Conclusion

Attacks on home networking devices have already been carried out at large scale. For example, in March 2012 over 4.5 million DSL modems in Brazil were found to have been compromised through a combination of malicious DNS servers and automated exploit scripts [1]. The goal of that particular compromise was to obtain customers' banking credentials for financial gain, and it was discovered in part because the illegitimate use of those credentials alerted customers to the security breach. Our attack, by contrast, would raise no immediate "red flags" since votes would be changed transparently in transit; moreover, because our modified firmware could erase evidence of its own existence after doing its work, it would be exceedingly difficult to find conclusive evidence of the vote tampering after the fact.

The overall conclusion is inescapable: unencrypted PDF ballots sent via electronic mail can be altered transparently, potentially with no obvious sign of alteration, and certainly with no way to determine where on the network any alterations took place or the extent to which votes have been corrupted. This method of vote submission is inherently unsafe, and should not be used in any meaningful election.

References

[1] ASSOLINI, F. The tale of one thousand and one DSL modems. http://securelist.com/blog/research/57776/the-tale-of-one-thousand-and-one-dsl-modems/, October 2012.

[2] ESNAASHARI, S., WELCH, I., AND KOMISARCZUK, P. Determining home users' vulnerability to Universal Plug and Play (UPnP) attacks. In *27th International Conference on Advanced Information Networking and Applications Workshops* (Barcelona, Spain, March 2013).

[3] HALDERMAN, J. A., HURSTI, H., KITCAT, J., MACALPINE, M., SPRINGALL, D., FINKENAUER, T., AND DURUMERIC, Z. Independent report on E-voting in Estonia. http://estoniaevoting.org/, May 2014.

[4] LONGENECKER, D. ASUS wireless router updates vulnerable to a Man in the Middle attack. http://dnlongen.blogspot.no/2014/10/CVE-2014-2718-Asus-RT-MITM.html, October 2014.

[5] SOCIETY, T. I. RFC3548: The Base16, Base32, and Base64 data encodings. http://tools.ietf.org/html/rfc3548.html, July 2003.

[6] WOLCHOK, S., WUSTROW, E., ISABEL, D., AND HALDERMAN, J. A. Attacking the Washington, D.C. Internet voting system. In *16th International Conference on Financial Cryptography and Data Security* (Kralendijk, Bonaire, February–March 2012).