Mobile Auth for Payments

Glenn C. Everhart

April 25, 2013

If we read conference reports, it is clear that avoiding malware infestations in customer endpoints is not moving in a promising direction for anyone but evildoers.

For that reason I have been interested in finding ways to perform transactions which will work even where there is active malware on the customer device. (I do presume and require that the authenticator device be secure however.)

An ideal endpoint would have dedicated resources which could not be infected, and which would be able to communicate to the user and to the authenticator without any malware being able to access this communication (at least, not in cleartext). This would be the old “secure path” scheme on a dedicated processor not directly visible to a phone CPU and with covert channel protection so the phone CPU could not observe timing or power of this device.

At the moment, vendors of phones may still have little interest in devising such. Thus the question is: what can be done with a mobile device with no dedicated secure processors or memory, by an organization which I will presume has no special access nor influence on phone hardware or OS.

It turns out that a somewhat viable auth scheme can be devised, which I will describe. (Since one application is payments, the description will at times talk about the authenticator as a “bank” and the mobile device as a “phone”. The intent is that this not limit the usefulness of the scheme.)

For payments, we want a method by which payment decisions will ensure that the customer is known to auth end, amount and payee are specified, and customer certifies that he is as claimed and that payment information is confirmed.

If we get customers to solve a simple puzzle and do some simple obfuscating of their results, we can get authenticators which change with each use, and don’t provide enough information on a mobile device for malware to determine what the authenticating pattern is.

Let me give an example.

Suppose the customer needs to pay $65 to FooBar, Inc, and is contacting his bank through FooBar on the internet.

The bank:

1. Gets information from the mobile to identify which mobile (and customer) is being contacted here
2. Sends a seed value which will be combined with phone information (and which will embed timestamp information to avoid replay) to produce a display
3. This display is then sent to screen (possibly from the authenticator, else from the mobile device) to produce a series of what will amount to random numbers (but the authenticator end will be able to generate these for its checking also). (Payee info would make sense to include in whatever hash is made so the results implicitly depend on that too.)
4. The display will look something along these lines:

 $0-$40 2 3 5 7 0 4 9 6 2 8 6 7 1 3 2

 $40-$60 0 3 5 1 2 7 8 5 6 3 9 9 4 7 0

$60-$80 3 6 1 2 9 0 6 7 8 3 4 6 2 7 1

$80-$120 2 7 0 9 3 5 6 1 4 8 7 5 4 4 9

$120 + 0 4 8 2 0 7 5 3 1 9 0 6 8 4 2

 a b c d e f g h i j k l m n o

1. The user will have previously selected perhaps 3 or 4 positions (indicated by the red letters at the bottom row) so that he will be able to select the positions and transmit the digits shown at those positions (using a separate keypad: the selection **must not** be done in place on the picture).
2. Thus suppose the user has chosen positions n a m e. (It may be good human factors to have the label column include the whole alphabet; that is an adjustment that can be done at will.)
3. He is told to select the digits in the row corresponding to the amount of his transaction. If his transaction is $65, that means he selects digits in the third row and would select 7 3 6 9.
4. The customer is asked to type a string now, which contains his selected digit pattern (the 7 3 6 9) with some other digits before, between, and/or after his selected digits. Thus he might type 42793536198. At the authenticating end, a search for the digits the customer should have used will find the 7369 pattern in there readily. (There must be some limit on the string size, of course, and if a customer enters only 4 characters he should be told to please select a new pattern before his next use.)
5. The resulting string gives no clear indication where the selected digits are, even if the malware knows the transaction amount. If multiple auths sending only the selection were done, the positions would be evident after several tries. With this system, though, it will take far more information to eliminate any positions as parts of the customer’s chosen authenticating pattern. It should be good enough to avoid giving malware information on the customer’s chosen pattern for some time, long enough that periodic resets will keep it safe.
6. The seed used by the bank should include some part that is payee specific, so that payee will implicitly be part of the pattern reported.
7. Note that the authenticator must use its knowledge of the seed sent to ensure that the selected digits are from “this” transaction. However this will ensure that someone cannot get a transaction forged successfully by misdirecting a customer somewhere other than the bank, and replay protection will keep any replays from working. This contrasts with the situation with display cards (which I have previously referred to as “whoami” cards) in that the authentication of bank to customer is not needed. A phony “bank” will not cause any money to be moved. Obviously the payee that was included in the hash should be checked to be sure the transaction pays that entity.
8. To make these displays harder for malware to analyze, use of pictures rather than text displays, and possibly captcha techniques, would be good practice to approximate some of the protection that a “secure channel” would have.
9. To approximate the protection of any secrets desired in the mobile (if any are used), metamorphism of code of the “payment” application can be used. By rearranging code and data somewhat, and rearranging instructions, such action would make (in principle) each copy of a “bank payment app” have its secrets somewhere different, making it harder for malware to find them. To the extent such secrets get incorporated in the number strings in the displays, they add an element to the auth that depends on the device, as well as the customer pattern.

It should be noted that this same system could be used where a secure processor is controlling a display where the main phone CPU cannot observe it, and then malware would not be able to tell what was going on. (Then too, the customer could simply type his selected pattern digits and not bother with the obfuscating string.)

As it is, this system requires only the entry of a string of perhaps 6 to 10 characters by the customer after he picks out his pattern by eye. An actual display should use color or other elements to facilitate the vertical picking out of positions, and might have a few more rows if such seemed to produce good readable selections. The pictures here are suggestive only.

You will note that the secret here is the pattern the user uses to pick out numbers. This does not show up directly anywhere, and even if the customer enters only the digits of his pattern, that won’t get back to the pattern in a single go. With extra digits being entered, and no indication on the mobile which are the “real” ones, reversing the behavior to figure the “real” pattern can be nontrivial. It might be easiest to look for timing differences to try to figure this out over several auths. Manifestly, an ever-changing number like this won’t lend itself to theft, and the hard attaching of payee into the mix will make stealing all the harder here. Note too that no mobile design hardware need be altered here.

To go from this kind of system to something that could be entered in a website, it might be simplest to use a one-time card number or perhaps a one-time CVV2 or PIN, good for a few minutes, which an auth of this kind might feed. In that case the payee and amount would need to match the actual transaction, a detail which will take a bit of design but is feasible. If a retailer is cooperative, there ought to be ways to pass this information between merchant and authenticator using known techniques.

This is of course usable with all kinds of e-commerce.