CS261 Scribe: Usable security

Motivation for usability in security

How usability is relevant

Usability in security is concerned with taking into account the human factor in security. This is a very important aspect that has been completely overlooked until recently; security experts either assumed that humans don't make mistakes (while they actually do) or just ignored them, building their system in terms of hardware and software only.

Usability is relevant in two ways:

1. **Task-oriented view**: How easy to use / intuitive / efficient is the security mechanism?
2. **Security-oriented view**: How does usability affect the security of the system?

The main argument for usability is that these two aspects are closely intertwined, and 2) cannot be achieved without 1).

How to cope with usability

There are two common approaches to deal with the usability of a system:

- **Change the user by training him**: this has been the historical approach taken by airline companies to ensure the safety of the passengers; planes were so secure that in case of crash the blame was put on the human side ("the pilot had misunderstood the instructions", "both parties did not agree on the unit to be used to measure distances", etc).
- **Change the product to make it more intuitive**: if the pilots repeatedly make the same error because the plane is unintuitive to use, isn't it because of a bad design rather than because of the lack of training of the crews?

Bolt-on security

A common mistake when it comes to security requirements in a system is to add them at the last moment, as if it was a secondary and passive magic layer. However, to be truly effective, security has to be incorporated in the design at the early stages of conception.

Human factors

A few key points to bear in mind when designing a usable secure schema:

- User studies are the only reliable way to assess what works and what does not work.
- For users **security is an aspect, not a goal in itself**, which makes it much harder to check.
- How can we measure the usability of a secure system?
    - **Subjective performance**: "How easy was it?", "How does it compare to… ?"

- **Objective performance**: What percentage of the users in a user study succeeded in doing a task? In what time?
- **Satisfaction**: "Would you use this software personally?"

Mistakes in usability

Phishing

File-sharing software

A file-sharing network's value is function of the amount of files shared. Hence to maximize the incentive for users to join, a file-sharing software had rather share as much of the user's files as possible; sometimes, this means sharing the whole hard drive, exposing private documents.

Possible solutions to this problem include:

- Blacklist some directories
- Require explicit authorization to share files
- Check the visibility of the files

The problem is that it is hard to come up with a generic solution: what works for me doesn't necessarily work for others.

Windows Firewall / SELinux

Passwords

What's wrong with passwords

Passwords are one of the biggest failures of the security community. They have a lot of weaknesses:

- Users can't remember an arbitrary number of passwords, so they most often use the same password everywhere. We all know that a security device is only as strong as its weakest component… hence a single weak system for which this password is used endangers all the other systems for which the user uses the same password. Worse: if one of the systems is malicious, the attacker has a very likely list of usernames and passwords for other websites.
- Users sometimes note passwords down on these infamous sticky notes on the screen…
- Passwords can be shared, guessed (social engineering)
- Passwords have a very low entropy (mostly based on dictionaries)

What replacement for passwords?

A number of replacements have been proposed:

- **Something you know**: challenge questions, …
- **Something you have**: hardware fob
- **Something you are**: biometrics
- **SSH, public key authentication**: the computer proves who he is

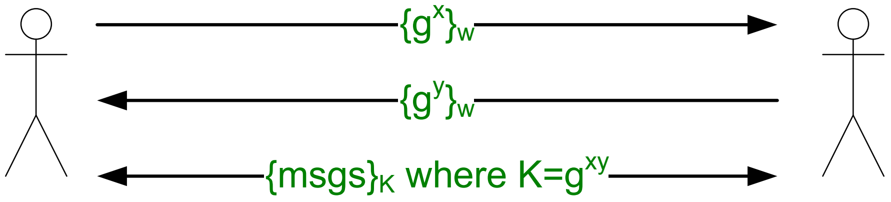
Two-factor authentication

Sometimes two of these factors (something you have / you know / you are) are combined; this has been adopted by online banks for instance, and it is called **two-factor authentication**.

It is not clear that the two-factor solutions that are actually deployed work very well. Studies prove that a non-negligible number of users will enter their password even when the sitekey of an online bank is not presented for instance.

Password authenticated key exchange

This is a nice application of passwords: two entities sharing a secret *w* can set up a common key *K* by encrypting the data exchanged in a standard Diffie-Hellman key exchange with *w* (see further in the document for an explanation of Diffie-Hellman):



The advantages of this technique are:

- Eavesdropping is impossible
- Spoofing the other computer is impossible (unless the attacker knows the password)
- Offline dictionaries attacks are impossible (one would have to guess the password and break Diffie-Hellman, which is considered a hard problem as of today)

Warning boxes & Confirm dialogues

The confirmation boxes of Windows Vista are so annoying that users do not look at them anymore and blindly click on “authorize”.

Similarly:

- Users don’t really pay attention to the confirmation boxes
- Users don’t pay attention to the SSL warning boxes (“Do you want to trust Foo?”)

Lack of encrypted emails

Whereas all the technology is here, unencrypted and unsigned emails still make up the vast majority of the mails! Emails are well-known to have absolutely no security embedded in them, and virtually almost anything in an email can be faked; hence emails lend very easily to phishing and spamming for instance.

Viruses

Double-clicking on attachments in email clients can spread the virus without the user's knowledge or consent…

Successes in usability

SSH (with public key)

In this case, the protocol is designed so that I can prove that I have (or that my computer has) possession of a private key without explicitly sending it. The key is typically generated by the machine for the user, so it has much better entropy and is virtually impossible to reverse-engineer.

However, this means that if the "device" (software or hardware) holding the private key is stolen then the attacker can impersonate me; to avoid this, a 2-factor authentication scheme is often used so that authentication does not solely rely on the couple public key / private key.

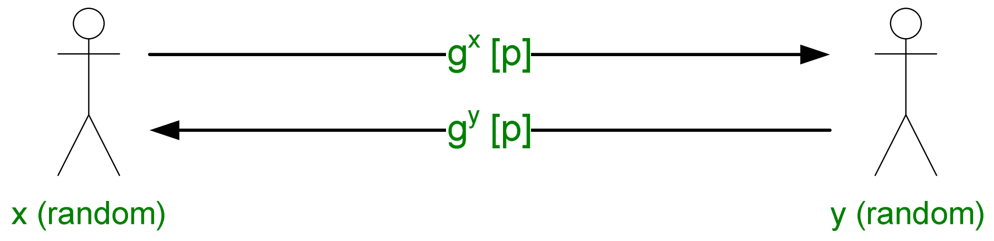
Key continuity management

This basically means checking that the key doesn't change through time. If we are sure that we did everything right the first time (when we retrieved the public key for instance) then we should be safe.

This is essential to guard against man-in-the-middle attacks.

Diffie-Hellman key exchange

The schema is (a [b] denotes a mod b):



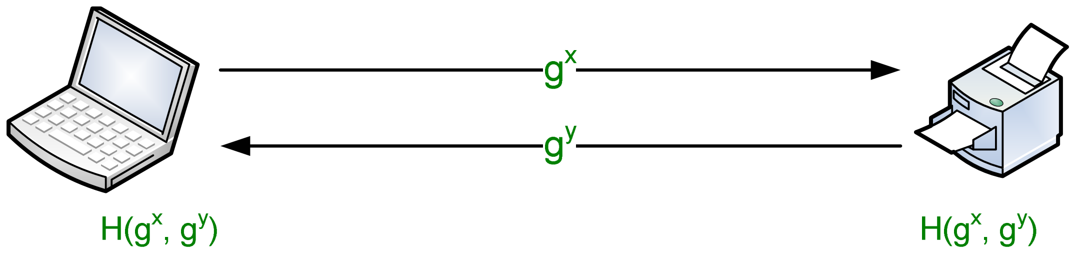
This protocol relies on the fact that $(g^y)^x[p] = (g^x)^y[p]$ (*g* and *p* are publicly known). A few remarks on this protocol:

- Upon completion of the exchange, and without exchanging x and y, the two actors share a common secret K = g^{xy} [p]. However, neither actor has any clue about who he is talking to.
- To prove one's identity, each actor has to sign with his private key the data that has been exchanged.
- The security relies on the fact that it is impossible to infer K = g^{xy} [p] from the observation of g^x [p] and g^y [p]. Note that this can be further strengthened (see "Password authenticated key exchange" above).

Ad-hoc networks (wireless)

Wireless networks are naturally prone to attacks and intrusion due to the very nature of the medium used to transport data. We consider here an example: suppose we want to print something on a network printer, but we want to subject the use of the printer to user confirmation, and we want to give the user a way to make sure that the document he's going to approve for printing is the one he intended to (and not a thousand pages document sent by an attacker).

A possible solution would be the following:



Then we have both the laptop and the printer display $H(g^x, g^y)$ and compare! However there is a pitfall: what interface should we present to the user on the printer?

- Having a confirmation screen ("Is 0x9876 the correct value?") will lead to most users clicking "YES" without paying attention (cf the paragraph on Vista warnings, etc)
- Asking the user to manually type in the confirmation code is cumbersome.
- Propose the user with several confirmation codes (the good one + several generated at random), and ask him to pick the right one

The third solution is the best one in that it makes it harder for the user to do the wrong thing.

Conclusion

Security systems affect human behavior, sometimes in an adverse way (adding more security leads to less overall security):

1. **"Shirking"**: as there are more people or systems in charge of something, they tend to individually become more sloppy because they rely on the others (analogy: in psychology, it has been proven that if there are say 20 witnesses to an accident, there is less chance that one of them will report it to the police than if there were only 2 witnesses, because each witness supposes that "there are many others witnesses, one of them will report it!"). But we know that a system is only as secure as its weakest point!
2. **"Bypass"**: too much security leads users to bypass it, especially when the security module produces a lot of false alarms ("cry wolf")
3. **"Overcompensation"**: users behave in a more risky way as the system is more secure. This one is not a security problem: rather than having security level requirements, users have a risk tolerance. So when the security of a system increases, they will adopt a (comparatively) more risky behavior that maintains their overall risk tolerance rather than keep their old habits.