Cryptographic Passwords and Authentication

stf

<2018-06-23 Sat>

NIST 800-63-3: Digital Authentication Guidelines [1]

- Minimum length: 8
- Minimum maximum length: 64
- allow all printable ASCII characters
- allow even all UNICODE characters, emoji inclusive
- No composition rules.

[1] https://pages.nist.gov/800-63-3/sp800-63-3.html

Offline Dictionary Attacks

- leaked password dbs
- millions of dictionary words / second checked

Password Managers

- Do not reuse passwords
- Do not use dictionary words
- High entropy (>80bit)

(sometimes)

Online Password Managers

pro

- easy syncing
- little installation overhead

con

- privacy
- attack surface (browser+3rd party)
- centralized, juicy target

classical convenience over security trade-off

Offline Password Managers

pro

- control
- verifiable

con

- syncing
- user is responsible for security

classical security over convenience trade-off

Cons of all passwords managers

- your master password is the key to the kingdom,
- offline bruteforce against your db
- keylogging
- many keep old user-chosen passwords, which are weak

Double Trouble

Double attack surface

- server user databases
- password storage

Crypto

magic silverbullets to the rescue \o/

SPHINX [2]

Setup

- *Group G.* The scheme works over a cyclic group G of prime order q, $|q| = \ell$, with generator g.
- *Hash functions* H, H' map arbitrary-length strings into elements of $\{0,1\}^\tau$ and G, respectively, where τ is a security parameter.
- *OPRF.* For a key $k \leftarrow Z_q$, we define function F_k as $F_k(x) = H(x, (H'(x))^k)$.
- *Parties.* User U, Device D, Server S.
- *Dictionary* Dict of size 2^d (a power of 2 is used for notational convenience only).

Initialization Phase *(assumed to be executed over secure links)*

- FK-PTR Initialization: U chooses password pwd $\leftarrow$ Dict; D chooses and stores OPRF key $k \leftarrow Z_q$; U interacts with D to compute rwd $= F_k(\mathsf{pwd})$.

Login Phase

- **User-Device Interaction (FK-PTR)**
  1. U chooses $\rho \leftarrow Z_q$; sends $\alpha = (H'(\mathsf{pwd}))^\rho$ to D.
  2. D checks that the received $\alpha \in G$ and if so it responds with $\beta = \alpha^k$.
  3. U sets rwd $= H(\mathsf{pwd}, \beta^{1/\rho})$.

[2] https://eprint.iacr.org/2015/1099

SPHINX Benefits

a password Store that Perfectly Hides from Itself (No eXaggeration)

- information theoretically secure password store
- manager does not know password
- manager salt independent from input/output passwords
- can use more than one "master" password

how does this work again?

Enter password

1. user enters password

User chooses random R

1. user enters password
2. "user" chooses random R

User blinds password with R

1. user enters password
2. "user" chooses random R
3. $a = H(pwd)^R$

User sends blinded password to storage

1. user enters password
2. "user" chooses random R
3. a = $H(pwd)^R$
4. User sends a to storage

Storage contributes its own "secret"

1. user enters password
2. "user" chooses random R
3. $a = H(pwd)^R$
4. User sends 'a' to storage
5. Storage returns $b = a^K$

User unblinds final password

1. user enters password
2. "user" chooses random R
3. $a = H(pwd)^R$
4. User sends 'a' to storage
5. Storage returns $b = a^K$
6. User unblinds b by $b^{(1/R)} = H(pwd)^K$

Security

- storage compromise: no problem
- network compromise: no problem
- offline dictionary against server: no problem
- storage+server compromised: offline dictionary against master pwd
- does not protect against compromised user (keylogging)

libsphinx et all

- `https://github.com/stef/libsphinx`
- `https://github.com/stef/pwdsphinx`
- `https://github.com/stef/websphinx-chrom`
- `https://github.com/stef/websphinx-firefox`
- `https://github.com/stef/winsphinx`
- also implemented in the PITCHFORK!!!5! \o/

testers, ports to smartphones, users welcome!

NIST 800-63-3: Digital Authentication Guidelines II

Server Side

- No expiration without reason (forgotten,phished,leaked)
- All passwords hashed (keyed), salted (>32bit) and stretched (pbkdf2 10.000)
- No password hints.
- No Knowledge-based authentication.
- No SMS in 2FA

OPAQUE [3]

Init: On input pw, p_U by U and k, PS by S, U computes $rw = H(pw, H'(pw)^k)$

and $c = AuthEnc_{rw}(p_U, P_U, P_S)$. S stores (k, p_S, c). U only keeps pw.

Login:

U (pw) | S (k, p_S, c)

$r, x \leftarrow Z_q$

$U \rightarrow S$: $\alpha = H'(pw)^r, \; X = g^x$

$y \leftarrow Z_q$

$S \rightarrow U$: $\beta = \alpha^k, \; c, \; Y = g^y$

- $rw \leftarrow H(pw, \beta^{1/r})$
- $p_U, PK_U, PK_S \leftarrow AuthDec_{rw}(c)$
- $K = KE(p_U, x, P_S, Y)$ $\qquad$ $K = KE(p_S, y, P_U, X)$

[3] https://eprint.iacr.org/2018/163

OPAQUE Init

the server

- generates and publishes public key
- generates a random salt k for user

the user or the server:

- generates public key pair
- calculates secret key $K = H(pw, H(pw)^k)$
- encrypts user keypair and the server public key with K

finally

- the server stores the encrypted keys

OPAQUE user initiates session

the user

- generates an ephemeral keypair and a blinding factor r
- calculates $a = H(pw)^r$
- sends a and the public ephemeral key over to the server

OPAQUE server response

the server

- generates an ephemeral keypair
- calculates $b = a^k$ where k is the random salt from the init
- calculates a shared secret S using the long-term and ephemeral keys
- calculates auth=HMAC(1,S)
- sends b, auth, the encrypted user keys & the public ephemeral key over to the user

OPAQUE user finish

the user

- calculates K by unblinding b -> H(pwd,$b^{(1/r)}$)
- decrypts the encrypted keys
- using the decrypted and the ephemeral keys calculates the shared secret S
- using S calculates and verifies auth=HMAC(1,S)
- if user needs to authenticate it sends HMAC(2,S) to server

OPAQUE Benefits

- forward secure
- precomputation doesn't help server compromise
- stretching happens on the client
- salt never leaves the server
- password never leaves the client
- is an AKE $\rightarrow$ shared key

cons:

- explicit user authentication is an extra message

OPAQUE in libsphinx

OPAQUE implemented in

`https://github.com/stef/libsphinx`

ports to PAM, ningx auth module, javascript, php, etc warmly welcome.

The End

Questions?