CLIMBING THE SPHINX

THE JOURNEY OF PORTING IT TO ANDROID AND THE DETOURS OF FIXING DE-SIGN VULNERABILITIES

silent signal

András Veres-Szentkirályi **2020-08-22**

$ whoami

András Veres-Szentkirályi

- OSCP, GWAPT, SISE
- Silent Signal co-founder
- pentester, toolmaker

Fahrplan

1. The basics
2. Distribution is hard
3. Secure protocol design is hard
4. Final thoughts

What is SPHINX?

- EN: `https://www.youtube.com/watch?v=JF-ivzWqha4`
- HU: `https://www.youtube.com/watch?v=dP-Pnr7pdpM`
- a password **S**tore that **P**erfectly **H**ides from **I**tself (**N**o **X**aggeration)
- distributed yet more secure than naïve approaches
- free software implementation
  - `https://github.com/stef/libsphinx`
  - `https://github.com/stef/pwdsphinx`

What did I do?

silent signal

- a distributed password manager is worth more if it runs on smartphones
- ported SPHINX to Android
- in other words: introduced the first alternate implementation
- thanks for the funding from NLnet as part of the NGI0 PET fund

SPHINX basics needed for this talk

silent signal

From Stef's original SPHINX presentation:

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 $b^{\frac{1}{R}} = H(pwd)^K \Rightarrow$ we'll call this $\mathtt{rwd}$ in this talk

Fahrplan

1. The basics
2. Distribution is hard
3. Secure protocol design is hard
4. Final thoughts

Malicious server tracking users

- the original version had an Ed25519 key for signing management requests
- this key was the same for every account they stored
- a malicious server could link accounts that belong to the same person

```
$ find data -name pub | xargs sha256sum | cut -c 1-64 | sort | uniq -c
     12 e63a01d67bd96d4607e4643e9122f071523d3b1c1d9f42fbad9790b34127726a
...
```

Solution preventing user tracking

silent signal

- generate a random 32-byte “master key”
- use keyed hash to derive seeds for account-specific signing keys
- easy to generate, hard to correlate

```
$ find data -name pub | xargs sha256sum | cut -c 1-64 | sort -u | wc -l
1265
```

Device compromise

silent signal

- the original version had an Ed25519 key for signing management requests
- how do we synchronize this key?

Device compromise

- the original version had an Ed25519 key for signing management requests
- how do we synchronize this key?
  - QR code offers easy yet secure sharing
  - for compactness, Base64 and such should be avoided
- what if the key gets compromised? $\Rightarrow$ DoS

Device compromise

- the original version had an Ed25519 key for signing management requests
- how do we synchronize this key?
  - QR code offers easy yet secure sharing
  - for compactness, Base64 and such should be avoided
- what if the key gets compromised? $\Rightarrow$ DoS
  - what if we include `rwd` in the computation?
  - tradeoff: offline brute force possible in case of a compromised/malicious server

Device compromise – solution

silent signal

```
fun auth(socket, hostId, rwd = ByteArray(0)) {
    val nonce = socket.getInputStream().readExactly(AUTH_NONCE_BYTES)
    socket.getOutputStream().write(getSignKey(hostId, rwd).sign(nonce))
}

fun getSignKey(id, rwd = ByteArray(0)) =
    Ed25519PrivateKey.fromSeed(key.foldHash(Context.SIGNING, id, rwd))

fun foldHash(context: Context, vararg messages: ByteArray): ByteArray =
    context.foldHash(*(listOf(asBytes) + messages.toList()).toTypedArray())

fun foldHash(vararg messages: ByteArray): ByteArray =
    messages.fold(value.toByteArray(), ::genericHash)
```

List of usernames

- the original version stored the list of usernames for each domain locally
- every device needs R/W access to this list

List of usernames

- the original version stored the list of usernames for each domain locally
- every device needs R/W access to this list
- solution: (E2EE) BLOB storage
- no `rwd` $\Rightarrow$ no authentication – but I guess it's fine?

OPAQUE

- another solution would be to use OPAQUE for management
- it could have some nice additional properties
- but no tradeoff option to avoid brute force

Fahrplan

1. The basics
2. Distribution is hard
3. Secure protocol design is hard
4. Final thoughts

Encrypted rule – original version

silent signal

- problem: `rwd` is just a bunch of high-entropy bits while we need passwords that fit various policies regarding length and character set
- original solution: pack character set and length into 16 bits, encrypt this and upload/store/retrieve along with the SPHINX process
- original protocol runs directly over plain TCP
  - SPHINX itself doesn't necessarily require encryption
  - requests are encrypted using the server public key
  - response contains SPHINX result and E2EE rule

Encrypted rule – problem and solution

- this doesn’t prevent *tracking which account was requested* when by *eavesdroppers*
- intermediate solution: convert Ed25519 key to Curve25519 and encrypt the already encrypted rule again by the server
  - outside asymmetric layer protects against traffic analysis
  - inside symmetric layer protects against compromised/malicious server

```
  try:
    rule = readf(path+'/rule')
  except ValueError:
    return b'fail' # key not found

+ with open(path+'/xpub','rb') as fd:
+   xpk = fd.read()
+ rule = pysodium.crypto_box_seal(rule, xpk)
+
  try:
    return sphinxlib.respond(chal, secret)+rule
```

Further problems

- requests are encrypted using the server's Curve25519 public key
  - replay attacks are trivial to perform
  - no forward secrecy
  - random protocol/port is easier to track and/or block (public Wi-Fi et al)

Further problems

- requests are encrypted using the server's Curve25519 public key
  - replay attacks are trivial to perform
  - no forward secrecy
  - random protocol/port is easier to track and/or block (public Wi-Fi et al)
- TLS solves all of these and is not that much worse
  - solves replay attacks (c.f. TLS 1.3 0-RTT) and forward secrecy
  - usually allowed at least on TCP/443 (HTTPS)
  - PKI makes server public key distribution an optional hardening
- recent versions are not that different from the original protocol
  - EC certs are a reality (although CA/B limits this to NIST curves)
  - ECDHE key exchange supports X25519
  - ChaCha20-Poly1305 cipher suites exist since RFC 7905
  - session resumption can improve performance

Fahrplan

1. The basics
2. Distribution is hard
3. Secure protocol design is hard
4. Final thoughts

Future plans

- rule could contain a XOR mask that should be applied to `rwd` before applying the password derivation phase
  - useful for storing passwords that can't/shouldn't be changed
  - Android offers standard interface to store CC info, this could be used for that as well
  - why not BLOB? SPHINX returns a valid-looking answer for every passphrase

Future plans

- rule could contain a XOR mask that should be applied to `rwd` before applying the password derivation phase
  - useful for storing passwords that can't/shouldn't be changed
  - Android offers standard interface to store CC info, this could be used for that as well
  - why not BLOB? SPHINX returns a valid-looking answer for every passphrase
- current version encrypts rule with integrity protection using a key derived from `rwd`
  - since it depends on `rwd`, there's a reliable method to tell whether the passphrase was right
  - plausible deniability? on-line brute force?

Future plans

- rule could contain a XOR mask that should be applied to `rwd` before applying the password derivation phase
  - useful for storing passwords that can't/shouldn't be changed
  - Android offers standard interface to store CC info, this could be used for that as well
  - why not BLOB? SPHINX returns a valid-looking answer for every passphrase
- current version encrypts rule with integrity protection using a key derived from `rwd`
  - since it depends on `rwd`, there's a reliable method to tell whether the passphrase was right
  - plausible deniability? on-line brute force?
- better rule encryption plans:
  - remove explicit integrity protection
  - remove implicit integrity oracle(s)
  - add (optional?) "check digit": n bits of `rwd` $\Rightarrow$ validity oracle with $P_{FP} = 2^{-n}$

Outro

- source code and binaries under MIT: `https://github.com/dnet/androsphinx`
- most of it is Kotlin $\Rightarrow$ iOS port should be easier
- GUI is kind of complete
- core functionality WORKSFORME
- pull requests welcome

THANKS!

ANDRÁS VERES-SZENTKIRÁLYI

vsza@silentsignal.hu

facebook.com/silentsignal.hu

@SilentSignalHU

@dn3t

silent signal