BSP: PAKE and decoy passwords

Student: Steve Meireles Tutor: Marjan Skrobot

2024

Table of Contents

The scientific question

How to detect if a password file is in possession of intruders and simultaneously time prevent phishing attacks?

References

- "Honeywords: Making password-cracking detectable", by Juels and Rivest, 2013
- "Encrypted key exchange: Password-based protocols secure against dictionary attacks", by Bellovin and Merrit, 1992
- "Password-Authenticated Public-Key Encryption", by Bradley, Camenisch, Jarecki, Lehmann, Neven and Xu, 2019
- "SweetPAKE: Key exchange with decoy passowrds", Arriaga, Ryan and Skrobot, 2023

Honeywords: Password File

Password File in Linux - /etc/passwd:

```
uuidd:x:68:68::/:/usr/bin/nologin
malga:x:1000:1000::/home/malga:/bin/bash
dhcpcd:x:974:974:dhcpcd privilege separation:/:/usr/bin/nologin
avahi:x:972:972:Avahi mDNS/DNS-SD daemon:/:/usr/bin/nologin
colord:x:971:971:Color management daemon:/var/lib/colord:/usr/bin/nologin
```

Honeywords: Honeychecker

- seperate hardened system
- store index of the correct passwords
- alarms system if password file is breached

Two functions:

$$Set(i, j)$$

$$Check(i, j)$$

Honeywords: Login Procedure

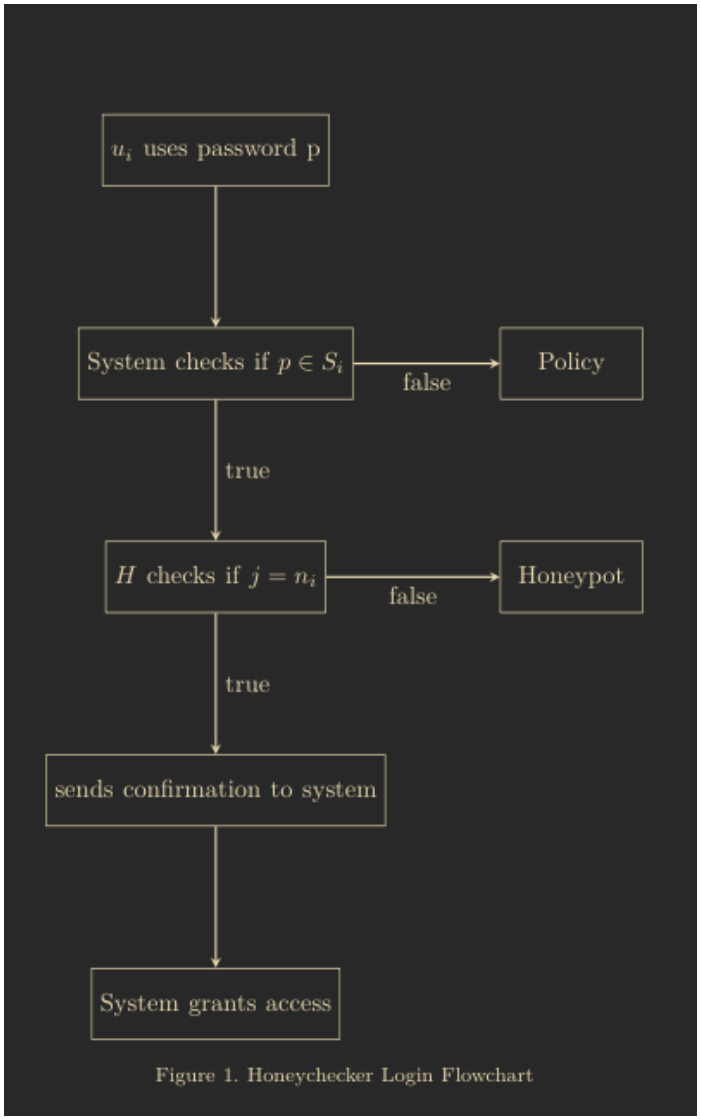

Figure: Honeychecker Login Flowchart

PAPKE

- Gen(pwd) -> (sk, apk)
- Enc(apk, pwd) -> $c = (c_1, c_2, c_3)$
- Dec(c, pwd) -> k

PAPKE

Decryption

- $k \leftarrow \{1,0\}^l$
- $y_2 \leftarrow Y_2 \cdot H_0(pwd)^{-1}$
- $R \leftarrow G$
- $(r_1, r_2) \leftarrow H_1(R, y_1, y_2, k)$
- $c_1 \leftarrow g_1^{r_1} g_2^{r_2}$
- $c_2 \leftarrow y_1^{r_1} y_2^{r_2} \cdot R$
- $c_3 \leftarrow H_2(R) \oplus k$
- $c = (c_1, c_2, c_3)$

SweetPAKE

- Gen(pwd) -> (apk, sk)
- Enc(pwd, apk) -> C
- Dec(sk, C) -> key
- Retrieve_key(i) -> key

SweetPAKE

Encryption part:

- $k \leftarrow \{0,1\}^*$
- $K = PRF(k, (id_a, apk))$
- $C[i] \leftarrow Enc(pwd, apk, K[i])$
- $(C', pmap) \leftarrow RP(C)$

Application: Requirements

- Implementation of BeePAKE
- Implementation of Honeywords generation algorithms
- will work on Python 2 or higher

Perfomance: Let k be the number of sugarwords in a password file Let T be the acceptable time required to share a key using the protocol:

$$T = 25\% \cdot k$$

Application: Requirements Template

The function generate() should use the following template:

- Description: First step of the BeePAKE protocol
- Parameter: No parameters
- Pre-condition: Protocol was started
- Post-condition: Generates a secret key, and a public key and returns the outbound message which will be sent
- Trigger: A party requests key-sharing with a server

Application: Design file structure

Repository named python-spake2 of warner in GitHub was used as base.

- honeyword_generation (directory)
  - gen.py (python file)
  - c_pws (text file)
- sweet_pake (directory)
  - file_operation.py (python file)
  - groups.py (python file)
  - ___init___.py (python file)
  - six.py (python file)
  - sweet_pake.py (python file)
  - util.py (python file)
- client_sweetPake.py (python file)
- pw_file (text file)

Application: Design data structures

The data structures of both classes SweetPAKE_Client and SweetPAKE_Server:

- password is the shared key stored as a byte object
- ida and idb being the ids of the sides which are communicating with each other stored as byte object
- params is the integer group being used as the object of the _Params class
- entropy_f the entropy used and stored as byte object

Additional data_structure of SweetPAKE_Server:

- database which is an associative array with all passwords as values and the usernames as values

Application: Production

Honeywords Generation Method, Chaffing-by-digits

```
def gen_chaff_digits(p, k):
    positions = []
    i = 0

    sugarwords = []

    for c in p:
        if c.isdigit():
            positions.append(i)
        i += 1

    sys_random = random.SystemRandom()
    for n in range(k):
        sugarwords.append(p)
        for x in positions:
            rand = sys_random.randint(0, 9)
            sugarwords[n] = sugarwords[n][:x]
           + str(rand)
       + sugarwords[n][x+1:]

    sugarwords.append(p)
    return sugarwords
```

Application: Production

```
    def gen(self):
        #gen function
        group = self.params.group
        self.random_exponent = group.random_exponent(self.entropy_f)
        self.y1_elem = group.Base1.exp(self.random_exponent)
        self.y2_elem = group.Base2.exp(self.random_exponent)
        Y2_elem = self.y2_elem.elementmult(group.password_to_hash(self.pw))

        #self.outbound_message = (self.y1+self.Y2) <-- apk
        y1_bytes = self.y1_elem.to_bytes()
        Y2_bytes = Y2_elem.to_bytes()
        self.outbound_message =  y1_bytes + Y2_bytes

        username_size = len(self.username).to_bytes()

        outbound_id_and_message = self.side + username_size + self.username + self
    .outbound_message

        return outbound_id_and_message
```

Conclusion

Question: How to detect if a password file is in possession of intruders and at the same Does time prevent phishing attacks?

- SweetPAKE is a good answer
- combines both strength of Honeywords and PAKE

Possible Technical Improvements:

- Integration of honeychecker
- Improved benchmark system
- Unit tests
- Improved comments
- Refractor according to one style guide
- Include MAC authentication
- Improve Error Handling

Thank you for your attention!