PAKE protocols und Decoy passwords

Friday 12th January, 2024 - 14:51

Steve Meireles Lopes
University of Luxembourg
Email: steve.meireles.001@student.uni.lu

This report has been produced under the supervision of:
Marjan Skrobot
University of Luxembourg
Email: marjan.skrobot@uni.lu

Abstract—Das Bericht antwortet auf die Frage: Wie erkennt man, ob eine Kennwort-Datei im Besitz von Eindringlingen ist und kann man gleichzeitig Phishing-Angriffe verhindern? Dies wird beantwortet mit SweetPAKE, das die Stärken von PAKE Protokolle und Honeywords kombiniert. Es stellt auch eine Implementierung eines SweetPAKE vor.

1.

Seit der Erfindung von Rechner, wird die Benutzername und Passwort Authentifizierungs Methode am häufigsten benutzt. Mittlerweile, weiss man dass Menschen sehr schlecht im erfinden und merken von komplexen Passwörten die man nicht vorhersagen kann. Es gibt etliche Methode diese Schwachstelle auszunutzen wie zum Beispiel die "Offline Dictionary" Attacke oder einfach das tausendfache probieren eines Passworts. In den letzen Jahre gab es auch Veröffentlichung von Millionen von Passwörter, weil Kennwort-Datei von Firmen an falschen Hände geraten. Ein weiteres Authentifikations Problem im Netz is das abhängige Vertrauen an Drittanbietern die Zertifikate austeilen.

Dieses Bericht widmet sich der Frage: Wie erkennt man, ob eine Kennwort-Datei im Besitz von Eindringlingen ist und kann man gleichzeitig Phishing-Angriffe verhindern?

2. Honeywords

Honeywords ist eine Methode das von Rivest und Juels empfohlen wurde[1]. Die Methode besteht darin mehrere "falsche "Passwörter zu haben, die sich als richtiges Passwort tarnen, die nennt man Honeywords. Diese Honeywords werden dann zusammen mit dem richtigen Passwort in der Kennwort-Datei gespeichert. Die Position wird dann einer seperaten System names Honeychecker gespeichert. Der Honeychecker checkt bei jedem Anmelde Versuch ob der Benutzer ein getarnten Passwort eingeben, wenn ja nimmt dementsprechende Massnahmen wie zum Beispiel dem Administrator zu alamieren. Die Autoren von Honeywords empfehlen auch mehrere Algorithmen um die Honeywords zu generieren, sie achten drauf, dass die Honeywords nicht zu differenzieren ist mit dem richtigen Passwort sodass ein ausenstehender nicht wissen kann was das richtige Passwort ist.

3. PAKE protocols

PAKE Protokolle sind Protokolle die es ermöglichen eine sichere Kommunikation herzustellen,obwohl man schwache Passwörter benutzt. Solche Protokolle benutzen zyklische Gruppen wie auch das Diffie-Hellman Prinzip.

Die Autoren von PAPKE bradley2019passwords, und Bradley, Tatiana und Camenisch, Jan und Jarecki, Stanislaw und Lehmann, Anja und Neven, Gregory und Xu, Jiayu, empfehlen zum Beispiel solch einen Protokoll. Jedoch fügen sie noch eine Primitive hinzu und zwar fokussieren sich mit dem generieren von Lang-Zeit Schlüsseln die eine sichere End-zu-End Verschlüsselung garantieren. Sie empfehlen zwei Schemas die das PAPKE Protokoll befolge, das PAPKE-IC und das PAPKE-FO.

4. SweetPAKE

SweetPAKE empfohlen von Arriage, Ryan und Skrobot, kombiniert die Stärken von Honeywords und PAKE Protokolle. Die Autoren von SweetPAKE empfehlen eine Naive Variante und eine sichere Variante. Die sichere Variante namens BeePAKE benutzt als Basis, das PAPKE protokoll.

Schritte:

Lass Alice und Bob zwei Parteien sein die ein Schlüssel teilen wollen.

1. Alice generiert ein öffentlicher Schlüssel und ein privater mit Hilfe von ihrem Passwort. Sie schickt das öffentliche Schlüssel zu Bob.

2. Bob erstellt dann ein Schlüssel für jedes Kennwort in der Kennwort-Datei mit Hilfe vom öffentlichen Schlüsseln. Dieses Liste von Schlüsseln wird dann an Alice geschickt.

3. Alice entschlüsselt alle Inhalte von der List bis sie ein gültigen Schlüssel bekommt und speichert die Position. Sie schickt dann die Position weiter an Bob.

4. Bob schickt nun die Position an den Honeychecker, der checkt ob ein Honeyword oder das richtege Passwort bentutz wurde.

5. Application

Diesen Abschnitt beeinhaltet den technischen Teil des Projektes. Es besteht aus der Implentierung des BeePAKE Protokolls marjan 2023.

Die Anwendung sollte eine Implementierung von BeePAKE [2] sein, die in dem vorherigen Abschnitt diskutiert und analysiert wurde. Die Implementierung sollte auf jedem Betriebssystem funktionieren. Es sollte mit der Python Version 2 oder höher laufen. Das BeePAKE-Protokoll wird mit dem PAPKE-FO [3] Protokoll implementiert, das in vorherigen Abschnitten erläutert wurde. Das Projekt wird eine Beispieldatei enthalten, die zeigt, wie die Implementierung verwendet und getestet werden kann. Es wird zwei Parteien haben und zeigen, wie beide einen Schlüssel teilen, indem sie den bereitgestellten Code verwenden.

Das Protokoll wird in vier Teile unterteilt, von denen jeder einen Schritt des Protokolls darstellt. Jeder Teil hat seine eigene Funktion:

- generate(): Erster Schritt des BeePAKE-Protokolls
- encryption(): Verschlüsselungsschritt
- decryption(): Entschlüsselungsschritt
- retrieve_key(): Schlüsselabrufschritt

Das Protokoll benötigt auch drei Funktionen des Verschlüsselungsschemas des PAPKE-Protokolls:

- papke_generate()
- papke_encryption()
- papke_decryption()

Die Funktion generate() sollte die folgende Vorlage verwenden:

- Beschreibung: Erster Schritt des BeePAKE-Protokolls
- Parameter: Keine Parameter
- Vorbedingung: Das Protokoll wurde gestartet
- Nachbedingung: Generiert einen geheimen Schlüssel und einen öffentlichen Schlüssel und gibt die ausgehende Nachricht zurück, die gesendet wird
- Auslöser: Eine Partei fordert den Schlüsselaustausch mit einem Server an

Die folgende gezeigte Funktion gen() ist der erst Schritt des BeePAKE Protokoll es generiert eine öffentlichen Schlüssel und eine privaten und speichert dies.

```
def gen(self):
        #gen function
        group = self.params.group
        self.rundom_exponent = group.
    rundom_exponent(self.entropy_f)
        self.y1_elem = group.Base1.exp(self.
    rundom_exponent)
        self.y2_elem = group.Base2.exp(self.
    rundom_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_und_message = self.side +
    username_size + self.username + self.
    outbound_message

        return outbound_id_und_message
```

Die Funktion enc() entschlüsselt jedes Passwort in der Datei mit Hilfe von der Nachrichter von der zweiten Partei und sendet die nötigen Information zu underen Partei.

```
def enc(self, inbound_message):
        #parse inbound_messahe
        self.inbound_message = self.
    _extract_message(inbound_message)

        #get username from message
        username_size = int.from_bytes(self.
    inbound_message[:1])
        self.working_with = self.inbound_message
    [1:username_size+1].decode('utf-8')

        self.inbound_message = self.
    inbound_message[username_size+1:]

        apk = self.parse_apk(self.inbound_message)
        group = self.params.group
        y1_elem = group.bytes_to_element(apk[0])
        Y2_elem = group.bytes_to_element(apk[1])
        client_pw_array = self.database[self.
    working_with]

        #PRF - get array of keys
        k = os.urandom(32)
        self.arr_K = group.secrets_to_array(k,
    y1_elem, Y2_elem, len(client_pw_array), 32)
        #self.arr_K = []
        self.ciphers = []

        #enc_function
        for i in range(len(client_pw_array)):
            gen_ciphers = self._papke_enc(group,
    y1_elem, Y2_elem, client_pw_array[i], self.
    arr_K[i])
            self.ciphers.append(gen_ciphers)

        #shuffle cipher array
        rp_ciphers, self.pmap = fisher_yates(self.
    ciphers)
```

```
        #message
        #self.outbound_message = c = (c1, c2, c3)
        self.outbound_message = b"".join(
    rp_ciphers)
        outbound_sid_und_message = self.side + len
    (rp_ciphers).to_bytes() + self.
    outbound_message
        return outbound_sid_und_message
```

Dec() nimmt eine Nachricht an und entschlüsselt die Liste von Schlüsseln bis sie eine gültige raus bekommt. Schlussendlich sendet sie die Position zur underen Partei.

```
def dec(self, inbound_message):
        #parse message
        group = self.params.group
        self.inbound_message = self.
    _extract_message(inbound_message)

        len_ciphers = int.from_bytes(self.
    inbound_message[:1])
        if len_ciphers < 0:
            raise ValueError("Invalid size")

        self.inbound_message = self.
    inbound_message[1:]
        ciphers = self._parse_array(self.
    inbound_message, len_ciphers)
        index = -1

        #dec
        for i in range(len(ciphers)):
            session_key_computed = self._papke_dec
    (group, ciphers[i])

            #checks if decryption is successful
            if session_key_computed != -1:
                index = i
                break

        if index == -1:
            raise ValueError("Could not decrypt")

        self.session_key = session_key_computed

        self.second_outbound_message = i.to_bytes
    ()

        return self.side + self.
    second_outbound_message
```

Die nächst gezeigte Funktion entnimmt aus der Nachricht und nimmt die nötigen Schritten um das Schlüssel zu bekommen.

```
def retrieve_key_ask_honeychecker(self,
    inbound_message):
        self.second_inbound_message = self.
    _extract_message(inbound_message)
        index = int.from_bytes(self.
    second_inbound_message)
        original_index = self.pmap[index]
        self.session_key = self.arr_K[
    original_index]

        # todo verify honeychecker

        return self.session_key
```

6. Conclusion

Als Schlussfolgerung kann man sage, dass SweetPake eine gute Antwort auf die Frage; Dieses Bericht widmet sich der Frage: Wie erkennt man, ob eine Kennwort-Datei im Besitz von Eindringlingen ist und kann man gleichzeitig Phishing-Angriffe verhindern? ist.

Mit Hilfe der Implementierung kann man dies auch austesten und möglicherweise Benchmarks ausführen.

References

[1] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013, pp. 145–160.

[2] M. Skrobot, "Sweetpake: Key exchange with decoy passwords," in SweetPAKE, 2023.

[3] T. Bradley, J. Camenisch, S. Jarecki, A. Lehmann, G. Neven, and J. Xu, "Password-authenticated public-key encryption," in Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17, Springer, 2019, pp. 442–462.