Advanced Topics in Online Privacy and Cybersecurity - 67515

PKI– Public Key Infrastructure implementation

Reut Ora Oelbaum

הדרישות לפרוייקט הן:

Programming task 2: Built a PKI system

- CAs
  - issue certificates to other entities (either make them CAs or not)
  - CA never sees entity's secret key
  - Include validity dates
  - Design and implement a revocation mechanism
- Entities
  - Sign objects using their secret keys
- Relaying parties (validators)
  - given object and cert chain, check object is valid

API +Structure + Examples

ישויות במערכת:

1. שרת- שהוא CA , בפרט השורש.
2. שרת שאינו CA
3. ישות קצה לקוח

בנוסף לישות נוספת שאני הוספתי- ישות שהיא מנהל על.

כדי להוסיף ישות למערכת יש להריץ את הקוד הבא: (לשנות את הIP, PORT, והשם של הישות שמוסיפים בהתאם). שיורש מהקובץ new_entity_interface

```python
import new_entity_interface
import GLOBALS

if __name__ == '__main__':

    my_host = GLOBALS.GRANDSON_IP
    my_port = GLOBALS.GRANDSON_PORT
    my_name = "GENERIC_NAME"
    new_entity_interface.run(my_host, my_port, my_name)
```

תוצאת ההרצה היא:

```
Do you want to serve as [1]ca or [2]end_server [3]end_client
```

וזה נותן למשתמש לבחור איזה סוג ישות הוא.

נראה דוג' לכל ישות:

1. **הוספת CA**

לאחר קנפוג המערכת ניתן להוסיף CA נוסף. כדי להיות CA צריך לבקש (ע"י תקשורת סוקטים) מ CA קיים (בפרוייקט זה אין בדיקת אימות למי שמבקש להיות CA).
מהישות המנהלת אנחנו מקבלים את רשימת הישויות הקיימות במערכת וכך ניתן לפנות לאחד מהם ולבקש להיות CA.

<u>דוג' הרצה:</u>
בשלב הזה- ה CA הקיימים במערכת הם:

```
CA_ROOT
└── CA1
```

המשתמש החדש מקבל גם את הכתובות שלהם-

```
Do you want to serve as [1]ca or [2]end_server [3]end_client1
server
The valid ca's
[('CA_ROOT', '127.0.0.1', 12355), ('CA1', '127.2.0.0', 12344)]
The valid server's
[]
The SERVER IP127.2.0.0
The SERVER PORT12344
```

השרת בחר ל"רשת" מ CA1 את הזכות להיות CA הוא מקבל CERT, נוסף למערכת ורץ בתור CA בעצמו.

```
CA_ROOT
└── CA1
    └── GENERIC_NAME
```

2. הוספת שרת

שרת בבואו להתחבר למערכת מבקש מCA קיים להגיש לו CERT. ולאחר מכן רץ בעצמו בתור שרת- נותן שרות ללקוחות ומגיש להם את ה CERT שלו (שקיבל חתום מה CA שפנה אליו)– כדי שמשתמש הקצה יוכלו מקומית לאמת אותו.

<u>דוג' הרצה:</u>

אני מתייחסת אל השרת במקרה זה כאל אתר כמו YNET.

```
my_host = "127.3.1.0"
my_port = 12344
my_name = "ynet"
new_entity_interface.run(my_host, my_port, my_name)
```

בשלב הזה- ה CA הקיימים במערכת הם:

```
CA_ROOT
└── CA1
    └── GENERIC_NAME
```

המשתמש החדש מקבל גם את הכתובות שלהם ובוחר לבקש CERT מהשרת CA בשם GENERIC_NAME

הוא מקבל את הCERT (שאח"כ יגיש למשתמש שניגש לאתר) ובמערכת המנהלת יש מבנה שמכיל גם CA'S וגם שרתים שאינם CA'S שניתן לראות שהוא נוסף לשם.

```
CA_ROOT
└── CA1
    └── GENERIC_NAME
        └── ynet
```

בדוג' הבא נראה איך לקוח פונה לשרת זה- מקבל את הCERT ומאמת אותו.

3. הוספת לקוח קצה פונה לשרת

הלקוח פונה לשרת (שמשמש גם כCA או לא) מקבל את הCERT של השרת ומאמת אותו מקומית בעזרת הפונקציה

<u>דוג' הרצה:</u>

הלקוח פונה לשרת של YNET שהוסף בדוג' הקודמת:

```
Do you want to serve as [1]ca or [2]end_server [3]end_client3
The valid ca's
[('CA_ROOT', '127.0.0.1', 12355), ('CA1', '127.2.0.0', 12344), ('GENERIC_NAME', '127.3.0.0', 12344)]
The valid server's
[('ynet', '127.3.1.0', 12344)]
The SERVER IP127.3.1.0
The SERVER PORT12344
```

הוא מקבל את הרשימה של השרתים הקיימים במערכת וכך מקבל את הכתובות של האתר.

הוא פונה אל YNET, מקבל את הCERT שלו ומאמת אותו..

להלן הקוד הרלוונטי:

```
cert = request_cert_from_server(HOST, PORT)
print(cert)
validity_check = get_and_validate_chain(cert)
if not validity_check:
    print("Error. Not valid cert for the desired Server")
    exit()
else:
    print("The Server has valid cert")
```

לאחר ההרצה מתקבלת הפלט הבא כרצוי:

```
The Server has valid cert
```

*הקבצים הרלוונטיים להרצה נמצאים בהגשה.

קנפוג המערכת:

Run CA_TREE.py

Run CA_ROOT.py

Run CA1.py

הקבצים המתאימים לדוג' הנ"ל:

Run new_entity_CA.py

Run new_entity_server.py

Run new_ynet_client.py

כאשר 3 הקבצים הנ"ל יורשים מהקובץ new_entity_interface.py

פונקציות שנמצאות ב new_entity_interface.py

```
def get_all_valid_ca_and_servers()
def get_revocation_list()
def request_cert_from_server(ip, port)
def request_public_key_from_ca(signer_ip, signer_port)
def get_and_validate_chain(cert)
```

ניתן להבין מה הפונקציות עושות משמן.

אפרט על האחרונה-

מקבלת CERT– ובודקת את תקינותו ואת התקינות של החותמים עליו עד לישות הראשונה (בדיקת תקינות כוללת גם תפוגה והאם הCERT נמצא ברשימה של הCERT החסומים (revocation_list)

```python
def get_and_validate_chain(cert):
    while cert.signer_name is not None:
        if get_revocation_list() != "The list is empty":
            if cert in get_revocation_list():
                print("The cert in revocation list")
                return False
        signar_public_key = request_public_key_from_ca(cert.signer_IP, cert.signer_PORT)
        try:
            if verify_using_public_key(cert.signature, cert.msg, signar_public_key) is None:
                ...
            else:
                print("error")
                return False
        except cryptography.exceptions.InvalidSignature as e:...
        except Exception as e:...

    return True
```

בקצרה במילים:

בודקים את תקינות הcert. (תאריך ורשימת המורחקים)

אם הCERT תקין- בודקים את התקינות של הCERT של מי שחתום עליו.

חוזרים על התהליך עד שמגיעים לCERT של השורש.

CERT תקין רק אם כל הCERT בדרך לשורש תקינים.

דוג' להרצה זדונית-

מתייחס לקבצים הבאים:

FAKE_CA.py

REQUEST_FROM_FAKE.py

במידה ושרת מבקש להיות CA מ CA לא מאומת- (שאינו צאצא חוקי ובתוקף של השורש) כמו במקרה זה- הבקשה לא תאושר.

```
Connected by ('127.0.0.1', 61944)
update data
 Eroor! .Invalid Signature
```

מנגנוני בדיקת תקינות

תאריך

כל CERT שמונפק מכיל שדה של תאריך הנפקה ותאריך תפוגה.

בעת בדיקת תקינות CERT בודקים אם התאריך הנוכחי הוא לפני התאריך תפוגה ובמידה ולא הCERT נחשב לא חוקי (וכנ"ל CERT ש"ירשו" ממנו.

נעשה תוך שימוש בספרייה datetime

Revocation List

קיימת פונקציונליות המאפשרת לשרת להוסיף CERT לRevocation List .

בזמן בדיקת תקינות של CERT בודקים את כל השרשרת עד השורש- אם אחד מהם ב Revocation List אז הCERT לא חוקי.

קבצים נוספים:

GLOBALS.py

מכיל קבועים כגון כתובות ופורטים של שרתים מסוימים (בפרט של השורש ושל המערכת שעוזרת לנהל)

CERT.py

מכיל Class עם כל השדות הרלוונטים לcert

Server.py

קובץ המכיל פונקציונליות שחשופה לשרתים.

מכיל את השדה

```
self.CA_FLAG
```

ששווה ל True במידה והשרת מתפקד כשרת שהוא CA (מנפיק CA)

ושווה לFalse במידה ואינו CA.

Performance Benchmarks:

התייחסות לזמנים- במקרה הבא הזמן שלוקח למשתמש קצה לאמת את הcert ש ynet הגיש לו. (זמן בדיקת השרשרת)

```
CA_ROOT
└── CA1
    └── ynet
```

time to validate cert :0.5468144416809082

The Server has valid cert