

task5

December 8, 2022

1 Task 5 - Core Dumped - (Reverse Engineering, Cryptography)Points: 500

1.1 Problem statement

The FBI knew who that was, and got a warrant to seize their laptop. It looks like they had an encrypted file, which may be of use to your investigation.

We believe that the attacker may have been clever and used the same RSA key that they use for SSH to encrypt the file. We asked the FBI to take a core dump of `ssh-agent` that was running on the attacker's computer.

Extract the attacker's private key from the core dump, and use it to decrypt the file.

Hint: if you have the private key in PEM format, you should be able to decrypt the file with the command `openssl pkeyutl -decrypt -inkey privatekey.pem -in data.enc`

1.2 Downloads

- [Core dump of ssh-agent from the attacker's computer \(core\)](#)
- [ssh-agent binary from the attacker's computer. The computer was running Ubuntu 20.04. \(ssh-agent\)](#)
- [Encrypted data file from the attacker's computer \(data.enc\)](#)

1.3 What to do

Enter the token value extracted from the decrypted file.

1.4 Write-up

First, check given file types.

```
[1]: %%bash
file data/task5/core
file data/task5/ssh-agent
```

```
data/task5/core: ELF 64-bit LSB core file, x86-64, version 1 (SYSV), SVR4-style,
from 'ssh-agent'
```

```
data/task5/ssh-agent: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
```

BuildID[sha1]=3734cf9330cd22aab10a4b215b8bcf789f0c6aeb, for GNU/Linux 3.2.0, stripped

Now, we know its architecture.

Next, we want to know what information we are going to extract. Since `core` has what was on memory when `ssh-agent` crashes, I want to know what information `ssh-agent` should have in its memory.

From `ssh-agent.c`, we know `struct idtable *idtab` can be used to decrypt our given `data.env`.

```
typedef struct identity {
    TAILQ_ENTRY(identity) next;
    struct sshkey *key;
    char *comment;
    char *provider;
    time_t death;
    u_int confirm;
    char *sk_provider;
    struct dest_constraint *dest_constraints;
    size_t ndest_constraints;
} Identity;

struct idtable {
    int nentries;
    TAILQ_HEAD(idqueue, identity) idlist;
};

/* private key table */
struct idtable *idtab;

int max_fd = 0;

/* pid of shell == parent of agent */
pid_t parent_pid = -1;
time_t parent_alive_interval = 0;

/* pid of process for which cleanup_socket is applicable */
pid_t cleanup_pid = 0;

/* pathname and directory for AUTH_SOCKET */
char socket_name[PATH_MAX];
char socket_dir[PATH_MAX];
```

There is `char socket_name[PATH_MAX]`; and `char socket_dir[PATH_MAX]`; after `idtab`. These variables store socket paths and its directories.

According to [the document by Oracle](#),

A unix-domain socket is created (`/tmp/ssh-XXXXXXXXX/agent.pid`) and the name of this socket is stored in the `SSH_AUTH_SOCK` environment variable. The socket is

made accessible only to the current user. This method is easily abused by root or another instance of the same user.

Thus, find the path and directory from the dumped data.

Run gdb to debug it. We get the following result.

```
pwndbg> search /tmp
Searching for value: '/tmp'
warning: Unable to access 16000 bytes of target memory at 0x55a080704e83, halting search.
<explored>      0x55a0807577e0 '/tmp/ssh-WESfbN143LWa/agent.18'
<explored>      0x55a080758820 '/tmp/ssh-WESfbN143LWa'
```

idtab is above the address 0x55a0807577e0 for char socket_name[PATH_MAX].

Think about the data size.

Just to reconfirm, this is the data we have above socket_name.

```
struct idtable *idtab;           // 8 bytes
int max_fd = 0;                  // 4 bytes
pid_t parent_pid = -1;           // 4 bytes
time_t parent_alive_interval = 0; // 8 bytes
pid_t cleanup_pid = 0;           // 4 bytes
```

What is the size of idtable? Let us calculate it.

This is the implementation of TAILQ_HEAD.

```
/*
 * Tail queue definitions.
 */
#define TAILQ_HEAD(name, type) \
struct name { \
    struct type *tqh_first; /* first element */ \
    struct type **tqh_last; /* addr of last next element */ \
}
```

Then, the size is

```
struct idtable {
    int nentries;           // 4 bytes
    TAILQ_HEAD(idqueue, identity) idlist; // 16 bytes
};
```

Thus, 20 bytes in total.

First look for the address of pointer to idtable. From the below code block, you can have it at 28 bytes before.

```
struct idtable *idtab;           // 8 bytes
int max_fd = 0;                  // 4 bytes
pid_t parent_pid = -1;           // 4 bytes
time_t parent_alive_interval = 0; // 8 bytes
```

```
pid_t cleanup_pid = 0;           // 4 bytes
char socket_name[PATH_MAX];
```

So, show the memory around there.

```
pwndbg> x/48wx 0x55a0807577e0 - 48
0x55a0807577b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a0807577c0: 0x812253c0 0x000055a0 0x00000000 0x00000000
0x55a0807577d0: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a0807577e0: 0x706d742f 0x6873732f 0x5345572d 0x314e6266
0x55a0807577f0: 0x574c3334 0x67612f61 0x2e746e65 0x00003831
0x55a080757800: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a080757810: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a080757820: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a080757830: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a080757840: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a080757850: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a080757860: 0x00000000 0x00000000 0x00000000 0x00000000
```

You can find a part, 0x55a0812253c0, that looks relevant. Let us input it.

```
0x55a0807577c0: 0x812253c0 0x000055a0 0x00000000 0x00000000

pwndbg> x/20wx 0x55a0812253c0
0x55a0812253c0: 0x00000001 0x00000000 0x8122ab90 0x000055a0
0x55a0812253d0: 0x8122ab90 0x000055a0 0x000001e1 0x00000000
0x55a0812253e0: 0x00000000 0x00000000 0x81209010 0x000055a0
0x55a0812253f0: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a081225400: 0x00000000 0x00000000 0x00000000 0x00000000
```

There is only one pointer after `int nentries`, 0x00000001. That is 0x55a08122ab90. Two places for a pointer are filled with that pointer. This is the pointer to `Identity`, so we need to check its definition.

Before revisiting `Identity`, read the definition of `TAILQ_ENTRY`.

```
#define TAILQ_ENTRY(type) \
struct { \
    struct type *tqe_next; /* next element */ \
    struct type **tqe_prev; /* address of previous next element */ \
}
```

OK, so the `TAILQ_ENTRY(identity)` has two pointers. Below is the calculation of data size of `Identity`.

```
typedef struct identity {
    TAILQ_ENTRY(identity) next;           // 16 bytes
    struct sshkey *key;                   // 8 bytes
    char *comment;                        // 8 bytes
    char *provider;                       // 8 bytes
    time_t death;                         // 8 bytes
    u_int confirm;                        // 4 bytes
    char *sk_provider;                    // 8 bytes
}
```

```

    struct dest_constraint *dest_constraints; // 8 bytes
    size_t ndest_constraints;                // 8 bytes
} Identity;

```

From what we know from the memory of `idtable* idtab`, `0x55a00x8122ab90` is the starting address of `Identity`.

```

pwndbg> x/20wx 0x55a08122ab90
0x55a08122ab90: 0x00000000 0x00000000 0x812253c8 0x000055a0
0x55a08122aba0: 0x81228ee0 0x000055a0 0x81226c00 0x000055a0
0x55a08122abb0: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a08122abc0: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a08122abd0: 0x00000000 0x00000000 0x000000a1 0x00000000

```

We can verify that `0x55a081226c00` is the pointer of `char* comment`.

```

pwndbg> x/s 0x55a081226c00
0x55a081226c00: "xWRb0rfx9G7CAkN0Y989cg"

```

Thus, `0x55a081228ee0` is the starting address of `struct sshkey* key`.

`struct sshkey* key` is like below. You can check it from [openssh-portable/sshkey.h](https://cvsweb.openbsd.org/cvsweb/src/lib/libssh/sshkey.h).

```

/* XXX opaquify? */
struct sshkey {
    int type;
    int flags;
    /* KEY_RSA */
    RSA *rsa;
    /* KEY_DSA */
    DSA *dsa;
    /* KEY_ECDSA and KEY_ECDSA_SK */
    int ecdsa_nid; /* NID of curve */
    EC_KEY *ecdsa;
    /* KEY_ED25519 and KEY_ED25519_SK */
    u_char *ed25519_sk;
    u_char *ed25519_pk;
    /* KEY_XMSS */
    char *xmss_name;
    char *xmss_filename; /* for state file updates */
    void *xmss_state; /* depends on xmss_name, opaque */
    u_char *xmss_sk;
    u_char *xmss_pk;
    /* KEY_ECDSA_SK and KEY_ED25519_SK */
    char *sk_application;
    uint8_t sk_flags;
    struct sshbuf *sk_key_handle;
    struct sshbuf *sk_reserved;
    /* Certificates */
    struct sshkey_cert *cert;
    /* Private key shielding */

```

```

    u_char  *shielded_private;
    size_t   shielded_len;
    u_char  *shield_prekey;
    size_t   shield_prekey_len;
};

```

When highlighting their bytes,

```

struct sshkey {                                //          total
    int  type;                                // 4 bytes
    int  flags;                               // 4 bytes 8
    RSA  *rsa;                                // 8 bytes 16
    DSA  *dsa;                                // 8 bytes 24
    int  ecdsa_nid;                           // 4 bytes 28
    EC_KEY *ecdsa;                            // 8 bytes 36
    u_char *ed25519_sk;                       // 8 bytes 44
    u_char *ed25519_pk;                       // 8 bytes 52
    char  *xmss_name;                         // 8 bytes 60
    char  *xmss_filename;                    // 8 bytes 68
    void  *xmss_state;                        // 8 bytes 76
    u_char *xmss_sk;                          // 8 bytes 84
    u_char *xmss_pk;                          // 8 bytes 92
    char  *sk_application;                    // 8 bytes 100
    uint8_t sk_flags;                         // 2 bytes 102
    struct sshbuf *sk_key_handle;             // 8 bytes 110
    struct sshbuf *sk_reserved;               // 8 bytes 118
    struct sshkey_cert *cert;                 // 8 bytes 126
    u_char *shielded_private;                 // 8 bytes 134
    size_t shielded_len;                      // 4 bytes 138
    u_char *shield_prekey;                    // 8 bytes 146
    size_t shield_prekey_len;                 // 4 bytes 150
};

```

What you can see from 0x55a081228ee0(struct sshkey) is like below.

```

pwndbg> x/44wx 0x55a081228ee0
0x55a081228ee0: 0x00000000 0x00000000 0x8122c0e0 0x000055a0
0x55a081228ef0: 0x00000000 0x00000000 0xffffffff 0x00000000
0x55a081228f00: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a081228f10: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a081228f20: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a081228f30: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a081228f40: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a081228f50: 0x00000000 0x00000000 0x00000000 0x00000000
0x55a081228f60: 0x00000000 0x00000000 0x8122bab0 0x000055a0
0x55a081228f70: 0x00000570 0x00000000 0x8122cc00 0x000055a0
0x55a081228f80: 0x00004000 0x00000000 0x00000031 0x00000000

```

The memory correspondence is

```

int  type;                                // 0x00000000

```

```

int  flags;                // 0x00000000
RSA  *rsa;                 // 0x8122c0e0 0x000055a0
DSA  *dsa;                 // 0x00000000 0x00000000
int  ecdsa_nid;            // 0xffffffff
EC_KEY *ecdsa;             // 0x00000000 0x00000000
u_char *ed25519_sk;        // 0x00000000 0x00000000
u_char *ed25519_pk;        // 0x00000000 0x00000000
char  *xmss_name;           // 0x00000000 0x00000000
char  *xmss_filename;      // 0x00000000 0x00000000
void  *xmss_state;         // 0x00000000 0x00000000
u_char *xmss_sk;           // 0x00000000 0x00000000
u_char *xmss_pk;           // 0x00000000 0x00000000
char  *sk_application;     // 0x00000000 0x00000000
uint8_t sk_flags;          // 0x00
struct sshbuf *sk_key_handle; // 0x00000000 0x00000000
struct sshbuf *sk_reserved;  // 0x00000000 0x00000000
struct sshkey_cert *cert;    // 0x00000000 0x00000000
u_char *shielded_private;    // 0x8122bab0 0x000055a0
size_t  shielded_len;        // 0x00000570
u_char *shield_prekey;       // 0x8122cc00 0x000055a0
size_t  shield_prekey_len;   // 0x00004000

```

You can verify `u_char* shielded_private` and `u_char *shield_prekey` by printing.

(348 = 0x00000570 / 4, 4096 = 0x00004000 / 4)

```
pwndbg> x/348wx 0x0055a08122bab0
```

```
pwndbg> x/4096wx 0x55a08122cc00
```

Store those data at `/tmp/shielded_private` and `/tmp/shield_prekey`.

```
pwndbg> dump memory /tmp/shielded_private 0x55a08122bab0 0x55a08122bab0 + 0x00000570
```

```
pwndbg> dump memory /tmp/shield_prekey 0x55a08122cc00 0x55a08122cc00 + 0x00004000
```

Then, get and build `openssh` at some directory. After building it, run `gdb` with the built `ssh-keygen`.

```
wget https://mirror.esc7.net/pub/OpenBSD/OpenSSH/portable/openssh-8.6p1.tar.gz
```

```
tar xvfz openssh-8.6p1.tar.gz
```

```
cd openssh-8.6p1
```

```
./configure --with-audit=debug
```

```
make ssh-keygen
```

```
gdb ./ssh-keygen
```

Copy and paste these commands into the `gdb`.

```
b main
```

```
b sshkey_free
```

```
r
```

```
set $miak = (struct sshkey *)sshkey_new(0)
```

```
set $shielded_private = (unsigned char *)malloc(1392)
```

```
set $shield_prekey = (unsigned char *)malloc(16384)
```

```

set $fd = fopen("/tmp/shielded_private", "r")
call fread($shielded_private, 1, 1392, $fd)
call fclose($fd)
set $fd = fopen("/tmp/shield_prekey", "r")
call fread($shield_prekey, 1, 16384, $fd)
call fclose($fd)
set $miak->shielded_private=$shielded_private
set $miak->shield_prekey=$shield_prekey
set $miak->shielded_len=1392
set $miak->shield_prekey_len=16384
call sshkey_unshield_private($miak)
bt
f 1
x *kp
call sshkey_save_private(*kp, "/tmp/plaintext_private_key", "", "comment", 0, "\x00", 0)
k
q

```

You have the openssh key at /tmp/plaintext_private_key.

```

$ bat -p /tmp/plaintext_private_key
-----BEGIN OPENSSH PRIVATE KEY-----
b3B1bnNzaC1rZXktbjEAAAAABG5vbmUAAAAAEbm9uZQAAAAAAAAABAAAABlAAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEARoHaV+pJ/EH6xn3TGW2WX2Prw4lZ+2Z42Hm5j/bdAsUs/Vdu8pkA
xvRUc5TUFDDilCwx0a+n/6wclARAKVxbHfQFrDpZwvbJw093SAOGKiRXP29pwzPC4Xmtno
W3nfs+meAoa5Q3TvwTin1arWLneP50w84L1jCMIZgd1XBe8vw0ZYISpG041cVUBLwd8odV
UN4zOesqaBOTCr0yLR958qf5690TDRzpePLh8KFHBTghWWuNpXZRx00ZSRM9Qj4LGRZsPD
zi5IU3118P2L76iveacM8T1tSWMURJ8hZoY6FzL9kJdnXp/93yCbUxVnrqh0gJLEQXl9yu
wrjV8Hbpa554YL1IKqV90QBAd4P9koeD6y4Ev+dV32xwewacCv3RkQj2c9yh6WPzQthiwP
GNZICneJRFzZx2Yunm160dwdBc/f224Mpeftvx7rn6gr6o5MjY+CrzRSwo4lfSpWBemad
tXQ4+Q2iG1+GTHxrtY/gcj0WJgU10WBEverDLfZtAAAFgDRtqao0bamqAAAAB3NzaC1yc2
EAAAGBAK6B21fQsfxB+sZ90xlt1l9j680JWftmeNh5uY/23QLFLP1XbvKZAMB0VHOu1BQ3
YpQsMTmvp/+sHJWkQClV2x30Ba3T88L2ycDvd0gDhiokVz9vacMzwuF5rZ6Ft537PpngKG
uUN078E4p9Wq1i53j+dMPOC9YwjCGYHZVwXvL8DmWCEqRjuNXFVAS8HfKHVVDemZnrKmgT
kwq9Mi0fefKn+evdEw0c6Xjy4fChRwU4IVlrjaV2UcdNGUkTPUI+CxkWbDw84uSFN9ZfD9
i++or3mnDPE9bUljFESfIWaG0hcy/ZCXZ16f/d8gm1MVZ66oToCSxEF5fcrsK41fB26Wue
eGC9SCqlfTkAWg+D/ZKHg+suBL/nVd9scHsGnAr90ZEI9nPcoelj80LYYsDxjWSHJ3iURc
2cdmLp5tejncMA23P39tuDKXn7b8e65+oK+qOTI2Pgq80UsK0JX0qVgXpmnbV00PkNohpf
hkx8a7WP4HIzliYFNdFgRL3qwy32bQAAAAAMBAAEAAAGAEJ5yWd/GAEHNJ7PC3RmdTbPw00
jN9HIg52hDtdmpbAfEEByGKJNPMTWixf1CFfMuZUKih7/9fECHxXsxiiC5AtwvrLYLEGcQ
lXwv04vOCV7zG3qz/OPjORx0wFSm1UDbRVRmNW30SP79tsTs1oEquIe7VSXY+ZMhc/x6u
AwEP3eKZWdsxjmooPxjhBFFuUqGpNuCPvPN8CzPJ3Iy9ViD9JBRna2D56tkoTl9KyF05z
5q3ryOmcPwjwiE0PE6Fzv95Fl8juYiw/Mp2VNaGu0/uKAsf67jshy+V7nJAs7b+hMt4LcT
G8JAtGYBsOudUG/0JmJ0vFrz30WhOLX1aq/9rY+hTcOUm7frwXE5x5nhXjzQDMiBbFRIG0
tvvqartedoCJ0dcTZB/luDxERl6XmwMC48nHDgA9QfnEucBbb8Wv4c1lg9mw0xNLMbfiYA
j6XZMdA7LOFw52RQ0aayPdKQuUeHbkqqK5wqeBzz7Zumx2oYw75EoQSxLOBtLZ7r7BAAAA
wAoOBsPJEBIVGc3KvSgP0WamvkuJM698W0+rhW8hZtwNQ6DtvQ4cGINaU/otbUtHMcTYtG
AXJ1CZFTa3z2akNvzALj5IcHaqgkGiFr6CoVUWDXixVdKnN8CMQssyKF0jb7Ea9YRnhdlE
Qs3+H1hf9xIrpntP9sq0pJj7kQaf1+3b4vY/FgZyjiIxgWSLeYgnLfJLkntnDXdKPOePtM

```



```
HmlJhJnB3hGw7PX17N3tyQyohqrb11GgRSqc8r5hTbhqqapQAAAMEA3K0KkgkaM7tQFhDR
YEMHwdRHQoJmxGKWyhp9IXcWez1KvHYe6YjGh/wtWkFNKwzMtEFVn/Bzzr2+SD3Ed2A03Q
QYIkQymJOvN+ZZUSEbF8p0NBVxNTQsxTZuq9HHTTJKOfuo41L67S5pMzrf21zLVZ7QIH4x
gIlhoshEmt1mPvHA21H5XPvtySsVCy2HKB5Y/GYdbyHQA1RLyzgWx7ad9ppcTPA8WQewdz
h0rFtgNn/Q6sLxDScuPL8c3pulrTi9AAAAwQDKc004At1xE9y8zZS7QV6h0FuPVIy0TisD
os8uva6LgiIM9FoHBzTOzvbd734fiobWgPQn7ly8prcPvpaB1Rnqn+jZPdEjPz4hP60A+/
MtDWbGUq4tXFHv76+h2FyObT2aI5WcERXX0aMsv31tNrP0aR5p48zbrFD0WwwhFoEeP9qz
EQTlUESza/YAWGP2WBs6JzdMh1+TnamYpbzcDm6G6Fb50MF1ZN4Anlw/R0JoqleG7S/HLW
15M/ioT5dhx3EAAAAHY29tbWVudAECAwQ=
-----END OPENSSH PRIVATE KEY-----
```

The next thing to do is to convert this key into rsa.

```
$ ssh-keygen -p -N "" -m pem -f /tmp/plaintext_private_key
```

```
$ bat -p /tmp/plaintext_private_key
```

```
-----BEGIN RSA PRIVATE KEY-----
```

```
MIIG4wIBAAKCAYEaU8pIYjqyFbPiRUyQgCCPeKfp/EVC3lN9Q3r5qF+7TqyUFQkv
N5uBIV+cPnfiLa6u3nq9KX0l3DXGRwqZNH1bQaNJfPCJNcj6dNdBe7D1oUkKo98j
Ngl70xTTfCid9/sRLCP4dQLpeeUsexKm5aREEPZgNm5lEvvfBTMvlzgYvVikYymf
Pd0NabXJYAusos8AFJNasqt9KVDrg22mwCj2HvqmMnz1dB4Me12QIWEyN7UrxUH7
XtVEytpsY2WCMKLz/z3oG/M35wznjONChltYXXM84mLPJ1EitnevWuqhCZPT8Tu5
/FqLeWdEqy9pgsbgrh5tNA0oHISxV1e0YjTqt4fUL1zvV8rW5Y19l5eBQpEpd1F
Ht02oVjk5LzkW4eREqg91JI+cpNMykFLsc82TANU+Yri0TYJQqVapCmsndoeBD1U
/A+C7m25s4zFyxkablAVUcVOBI+zes4cs1Ws7xMRd7jn5WUU+MY5oF+itQ/pJgt0
iyJlJfM2GFEq+a1JAGmBAECggGALLQFsSSJ2jK9LYQBbg7S0IW4ZK5UMbqs0hEf
N166pwrcIXnTPKi0/9PTzIQNihoVFvhhevdxGMktSqnt98fwytMxio4b45NWN0Z
IInKnz9VAhePOUDejfyZLz84A51HxDs33Rr5W2Qbx5x0EHYfG26bDro044sD1ygN
LzFSalicoWJ6Hu7tT3nP20ZMLWGbgkplr9y06c3yedALfvkBRfSi9afbgm3YyNFI
CzopCycvpJ0FdFj0jH+ch7jVWiCgDlUXHZa6aFIDKY8yYxJQ5dnwCiXcZ/spAqWp
gua0uiA/jEIrq2Yz5usiQRSqwx4XpMsI56Sr3YrZEU764lwI/rITQqoCF2HnME/D
1MAcFhCNoEqdbYw9HjodI/6Y4w7k4EBB3g6Mht8xKoiCqZ57L+HGgJqueaQROyfo
TCcZwty7EocudNgSg3czbWA1yruk7BK5H5rbRL3Wgc63YYSgtZcgMVLHV3wwLIh
1VmWZbdeLD0NZiMp42G8oMC+xMBAoHBA0hUGFkVav137PxUIhfT17bXAhFq5sut
Zqn93ih2zqdi1Xvbn3fFt6v5RoD7RvC7iF/2gYbHtJsya1GQrvFmHn+5jK6rxar/
N5K5Bi6NfzqqSjG0smgXSG87rkFsgdgbXuQQBN2900HP5XoCkNYa0dapTrewk7C
MVz4eAMX8qU1Mx8aayk90cxv6rEm07Uz6TDdh9rGqxnCJglblqlJYDn9dE1KjXB8
E39YkHokCk4f95dM9uai9ZqMIBkjh7mh4QKBwQD07HqaSfyDxbj+T3ev2Cd/MrQ1
zfkZExZq2b4ttg0yyzaJbpxJ+EMe2f7RVStaWpZatz5Z/OtfeVFCo20ww5Wc0ddH
PTcB0q+RawaE8woIE8IozozIIItjw7WkuVh6myxFqn0Gqt7Q2NrTp0409aiDZSj7H
Ei0vRaMhuxk0iEBldXbYTNsFoTaYrxQ693njmAxx9cis0JQf2PMuKPqxQoPcNMDF
hI6PvQK+ah/DeazszrNKkvdm82nWqqxi8mimxGkCgcBTGSkwS+Hd0VRZmHotTZ9d
jZ/2v0baOSZ8bQpT2aRm6In1aE0sonKkt1+JaNch6eHIeTiXFCM8U5dhMD/yphUI
mESCgXwK71ngR/+3DPMfRDvHEv54EbzeAWbd1PTtGlcIaDs+z1PNuV+Z+Wr2udc
OUbaPK0BYKIPx9IFQT6P5TNYz6k+shblaa5n0LOglg+1Y7dhACHy6UGTnUfR0yaI
92fu8ViPwEF1Did3GIb7FKIJJYwt4z5bLdY1RkIzOT+ECgcAwhb0aSo80N0FSIUBn
s8BPS4yGsw4gOyGenMDD51KmspBvWci8b8MNkQD4BYjM/OEstLWiVQNPuBseDUSl
kreatnpM8kXbu+8/omE8++D2U+vLte09aq6EkKeBfiBh7GyBHCdB6SIRNHLn3lcY
d7KMqaTG+bH0f9fPKP6YGvs+z3S29A7IGy26UBPMX/HtNNJ2RrRv7jSN40njISjo
```

```
ODjDgntR1eu40RJTpH8eT4IFZdMVQbFSKvI+rHcQHNf9g0ECgcEAs+7g6ym5j2Q+
Xy/FokIeKcmJblJvS8nZtq3olaAqMPj pzKe4NJLvnXzXLeds7GXJlfJUJ0k0Q+0b
EOTtsALg1dLkej8yaXCn0jxuVGU6HMBgqUvFxFxGUrCAKlfaCbDnJxaEcq7jhECRgw
LVpe7FcA/iVKgJJHVfGeYoKzH0sYW79IJIiLMzb59Cddbox5UUBR0m53iuu8zEGbI
SliKLk7/sohFd5orqfHJUAqSqSCA06SM8YDWfKlq0REx7Ula+Tm0
-----END RSA PRIVATE KEY-----
```

Let's decrypt the given data.enc to data.dec.

```
openssl pkeyutl -decrypt -inkey privatekey.pem -in data.enc -out data.dec
```

```
$ bat -p data.dec
```

```
# Netscape HTTP Cookie File
```

```
fqdhckntpkovqhu.ransommethis.net FALSE / TRUE 2145916800 tok eyJ0eXAiOiJKV1QiLCJhbG
```

What we want is its token, so input this.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlE2NTUyNjAwMzMsImV4cCI6MTY1Nzg1MjAzMywic2VjIjoim
```

You've got the flag.

