Secret Underground Club 1

Category: Cryptography
Created: Mar 1, 2021 9:24 PM
Solved: Yes
Subjective Difficulty: 🔥🔥

WriteUp:

Author: @Tibotix

This was a challenge in the CSCG2021 Competition.

📜 Challenge Description:

Can you be part of the secret underground club?

🔎 Research:

When looking at the source code we can see that we somehow need to provide a message and a signature so that this equation is true:

`message == pow(signature, e, n)` | where e and n are the numbers from the server-side generated RSA public key that will be sent to us

This will basically just verify the given message based on the given public key.

📝 Vulnerability Description:

The program does not ask the signature for a *specific* message. Instead we can provide our own message with its own signature. This allowing to provide a valid signature without knowing the private key of the key pair generated at the server.
A signature generation would go as follows:

`sig = message**d mod(n)` | where d is the private key

A signature verification on the other hand would go:

`message == sig**e mod(n)` , where the verification would success if this equation is true.

So when we are given a message to sign, we calculate the signature based on the message.

But when we have the freedom to choose our own message, we can simply set a random signature and calculate the related message to it. In this way we don't even need to know the private key `d` .

🧠 Exploit Development:

Lets set the signature to `0xdeadbeef`,
and calculate the related message with

`mes = sig**e mod(n)`.

🔐 Exploit Program:

```python
from pwn import *

p = remote("7b000000033d6bef33a6ffd0-secretundergroundclub1.challenge.broker.cscg.live", 31337, ssl=True)

p.recvuntil("e=")
e = int(p.recvuntil('\n').strip(), 16)
p.recvuntil("n=")
n = int(p.recvuntil('\n').strip(), 16)

log.info("e: {0}".format(hex(e)))
log.info("n: {0}".format(hex(n)))

signature = 0xdeadbeef
message = pow(signature, e, n)
log.info("signature: {0}".format(hex(signature)))
log.info("message: {0}".format(hex(message)))

p.recvuntil("Message:")
p.sendline(hex(message))

p.recvuntil("Signature:")
p.sendline(hex(signature))

p.interactive()
```

💥 Run Exploit:

```
tizian@tizian-vm1:~/CTF/CSCG2021/pwn/crypto/secret_underground_club1$ python3 exploit.py
[+] Opening connection to 7b000000033d6bef33a6ffd0-secretundergroundclub1.challenge.broker.cscg.live on port 31337: Done
[*] e: 0x10001
[*] n: 0xb39309769a47161f93c55e01d69cac1a245fd8e09971963986a3ce93dbca5872b1fcc77268f528492810d36f3900580cc4559540eb9e57ca694a51af4a374d75b83c1270011f3b6ca6203e72602de88c7f8af415a6e94e71c1c880865d5fd06fba75155ec4826fb74dc78b718bc0c1d07fb412700d1c446cc8907a728cf409a1
[*] signature: 0xdeadbeef
[*] message: 0xabe8c44467370fd2621b780ff4d27fb18df7b586e296a42a20c95139b29f67b05c32bb61ffd14da2ad7ba0aae7161091c9f9650bd9a7843aa5083221c75042b582a133771d9e5b3336317a201905f80ce90959e391cb17ade03bd4b291608312d09c3d615acf370cce4fba82bb972b6a4527ca54df614bdcd38bb4de450d8216
[*] Switching to interactive mode
Welcome
CSCG{rsa_seems_easy_but_apparently_it_is_not}
$
```

FLAG: CSCG{rsa_seems_easy_but_apparently_it_is_not}

🛡 Possible Prevention:

The usage of the RSA algorithm to verify the signature in this case is misused. The signature should only be generated by the one holding the private key.

🗄 Summary / Difficulties:

Always pay attention to how you use cryptographically functions.

📚 Further References:

[RSA (cryptosystem) - Wikipedia]

🔨 Used Tools:

- python
- pwntools