**Man in the middle**

For this challenge, you will play again the role of the security expert in your company. Your task is to investigate the suspicious traffic generated by one intern workstation (most probably infected with some malware client) sending encrypted data to one server outside the company. Your mission is to understand the behavior of the malware, and to the end finding a way to get clear messages exported to the outside of the company.

After some analysis work, you noticed that the two parties always use a kind of fixed protocol for their communication: the internal application asks for the server’s RSA public key, therefore using the replied key to encrypt transmitted data. Finally, the sender waits for an acknowledgment (ACK) response from server. You observed too that if you cut this ACK response, the client applications resumes the entire protocol for one more time. You tried to act as man in the middle replacing the server’s public key, but in this case, the communication suddenly stopped. For the key exchange, they does not use digital certificates, and you guess they are using instead a somewhat key validation technique, maybe similar with the one used by SSH.

After many tries, you observed something that could help you with the investigation: if you change only 1 bit of the server’s public key exponent, the client application does not react and go on with data transmission. Of course, in this case, the server is not able to decrypt and therefore it does not send the ACK response.

Using this behavior, now you are ready to put in place the next man in the middle type attack: during first iteration of the protocol, you catch the server’s key and change 1 bit of this before being received by the client. The server does not send the ACK and the client will resume the protocol with the second iteration. This time you will let the protocol run, saving again the encrypted data uploaded by client to server.

You sniffed all this traffic:

The server’s original key:

*-----BEGIN PUBLIC KEY-----*

*MIGdMA0GCSqGSIb3DQEBAQUAA4GLADCBhwKBgQCdD6rtwX+8Bb1D+Fs1nMN6caYW*

*lgTBzzXepC8PDKOSMuPFg3C9ZVmG7pjCrBieGY6vNRyMl3kzgnXebVXHHWLUu0VN*

*IIPnZeK17VBic5jffZUPTXubEQoMnlIZO22SiWEw+TeoTZMiZkeOMwDW0HZwJUov*

*tIqz5ky7YdSRKBurxwIBBQ==*

*-----END PUBLIC KEY-----*

The encryption of data in 1st protocol iteration:

*9218EE5ADAD937876522AE9521CAF193CB7687D98B2C21089392959B95AFB7417732E136989B6D5B035717C7600E9498DEED75ED3FC44223AF410C002D685F6E719F734639406680FF69582BDA77E5EB79E5605400B41197B72393809BEA4A6ECD8B35AD88516DF2B5F40B6E6856321D3330F2792F76319B643F8B97236AE31*

The encryption of data in 2nd protocol iteration:

*BADD5B1B31A370E6BB8E2028F82FD41756E21F1C6B43F56FF6EE4B6D2D083DC06AEBA61170F0253F536430B14C883241CC3B7EFD3017D553765A496B098D424B0234CE4A3779037BAB997360169B5E048019D55D4A5D1D2820FFCC877025B71FCB0D43259ADBCC790000D99A370D615CEF030F2941B0374A3A89FFD6BE8BBCE*

The flag is the message text sent by client to the outside server.