Can YOU crack the code?

…then participate in the second NSA Codebreaker Challenge! See details below for instructions on how to participate.

NSA Codebreaker Challenge 2.0

DISCLAIMER: The following is a FICTITIOUS story meant for providing a realistic context for the Codebreaker Challenge and is not tied in any way to actual events.

Background

An international terrorist organization has recently revised the operational security (OPSEC) procedures used to communicate with their members in the field. We have recovered a program that we believe is being used to covertly send encrypted messages. On the surface, the program appears to simply check the weather forecast in a few cities, but we believe there is more to the program than meets the eye. Your mission is to figure out how to execute the covert functionality, reverse-engineer the encryption algorithm, create a decryption program, and lastly figure out a way to decipher a message that was captured from a high-value target.

There are 4 different levels or "tiers" to this challenge problem, with each increasing in difficulty and building off the previous tier(s).

The goals of each tier are as follows:

> **Tier 1:** Determine how to execute the hidden functionality. To solve this challenge you will need to run the program with the correct arguments and execution environment.
>
> **Tier 2:** Accessing the hidden functionality requires passing an authentication check. To solve this challenge you will need to either find a way of bypassing the check or find a correct set of credentials.
>
> **Tier 3:** Once you bypass the authentication check, you will be able to create encrypted messages for any recipient of your choice. However, this program only provides the secret encryption functionality so you will need to figure out how to properly decrypt messages in order to read them. To solve this challenge, you must write a decryption program that takes as input an encrypted message and associated key (provided by the program) and produces plaintext output. You may use any language you prefer.
>
> *** NOTE *:** To receive the encrypted message for Tier 4, you'll need to send an email to the address provided by the program after you complete Tier 3.
>
> **Tier 4:** We have captured an encrypted message sent to a high-value target, but we do not know the secret key used to encrypt the message. Your challenge is to figure out how to decipher this message and recover the original plaintext. Once you solve this problem you will be able to decrypt any message you want without knowing the key! Good luck!

Getting Started

To solve these challenges, you will need to analyze the executable file with low-level tools such as a disassembler, debugger, hex editor, Linux binutils, etc. We recommend downloading the IDA disassembler from https://www.hex-rays.com/products/ida/support/download_demo.shtml. IDA is a very powerful reverse-engineering tool and used by professional security researchers around the world. But it can be intimidating to novice users, so here are a few tips to get you started:

1. The strings present in executable files can provide substantial insight into the program functionality. To view the strings in IDA (after disassembling a file), select View -> Open Subviews -> Strings.
2. IDA creates useful code and data cross-references that make it easier to navigate the executable. For example, by double-clicking on a string, its location in the binary will be revealed. Clicking on the xref link next to the string will take you to the code that references the string. Using strings and xrefs will help you focus your reverse-engineering efforts on the relevant areas of code.
3. The "Hex View" in IDA, which can be accessed from either View -> Open Subviews -> Hex Dump or by clicking the Hex View tab when IDA first loads, is useful for identifying byte sequences that you may want to modify in order to change the behavior of the program. If you select an instruction in the main disassembly window and then switch to the Hex View, the bytes associated with that instruction will be highlighted.
4. To test a patch that you wish to apply, try running the program in debugger and making the modification in memory to see whether your patch works as expected. When you are ready to make the patch permanent, a hex editor can be used to save your modifications to the program binary.
5. Remember that any value xor'd with itself is equal to zero. Examples:
    a. A xor A = 0
    b. if A xor B = C, then C xor A = B and C xor B = A
6. Base64 is a binary-to-text encoding scheme that represents binary data in an ASCII string format by translating it into a radix-64 representation. There are many commandline tools and free open-source libraries available that will base64 encode/decode data for you. Tiers 3 and 4 will require you to base64 decode data.

For more information about reverse-engineering, check out Lectures 1-5 on the following page: https://plus.google.com/113611061190453317839. Also, the Intel reference manuals may be downloaded from here: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html. These documents will help you understand the x86 instruction set and architectural details. Good luck!