NoteHack++
Securing an Editor

Students names	Neria Tzidkani , Gilad Weiss
Students IDs	209038876 , 206121527
Year	2018
Semester	A
Faculty	Software Engineering
Institution	JCT
Instructor	Arie Haenel

Contents

Abstract

Every running program has a process. If one can inject some code into a running process, he would have the power to “convince” that process to run the code for him. This power could be used for good or for evil. The injected code could be anything; it can be added security, or a malicious backdoor. It can be new functionality, or it can cause the program to crash. In our project, we use injection to add some crypto functionality to the well-known text editor “*notepad++*”.

Introduction

Every day the world becomes a more connected place. From a productivity standpoint, this is extremely convenient; your files are accessible from anywhere. This convenience, however, has an inherent security problem. If you have access, theoretically so does everyone else. Even the most mundane task, such as taking a note, can be a potential security risk.

Notepad++ is a very popular text editor. Many people choose it as their main editor because of its friendly, feature rich environment. Its glaring fault is its total lack security. This gaping hole can be filled with ease by adding cryptographic functionality. This is where **NoteHack++** comes in.

NoteHack++ is a simple program that adds crypto functionality to notepad++.

We use DLL injection and API hooking to add the functionality to the editor without making any permanent changes to it.

The method is simple: files are encrypted when saving, and decrypted when loading. All the information needed for the decryption is stored in a KSF file that is created in the program folder.

The files are secured using a *multi-tiered security system*, allowing for secure password changing (and potentially, secure collaboration as well).

The method making the changes to notepad++ is relatively simple; We create an additional thread in the editor, load our DLL into it, and then hook (hijack) some API's that are used by it. The placement of the hooks will ensure that the save and load functions will serve us instead of the editor, giving us control of the streams of information entering and leaving the it. Once we have control over these streams, we can add crypto functionality while the editor remains none the wiser.

Methods used

DLL Injection

The act of “adding” code to an existing program by forcing it to load a DLL containing the code. (the steps are explained in “*DLL Injection*” of the “*Procedure*” section)

API Hooking

The act of subtly (or not so subtly) redirecting the flow of the code to add arbitrary code. the added code can be for monitoring, added functionality, debugging etc. (the steps are explained in "*API Hooking*" of the "*Procedure*" section)

In our project, we used hooking to add crypto functionality.

Key Derivation Function

In this context we are interested specifically in key stretching and strengthening: in cases where a password will be used as a key, the password itself usually doesn't have the requirements to be considered a secure (e.g. commonly used), or even useable key (e.g. in the case where a key must be a certain size). In these cases, key derivation will "stretch" the key so it will be the correct size. Key derivation functions are designed to be slow, in order to raise the difficulty of a brute force attack and use a salt to take care of dictionary attackers.

We used PBKDF2 for key derivation. We chose it because it is time-tested and has been officially approved by NIST (SP 800-132).

CTR & GCM

Block cipher - reminder

An algorithm to encrypt a fixed-size group of bits, called a block. It can be considered secure if only a single block is encrypted per key, but there are plenty of designs that allow repeated use of the key in a secure way. Each block cipher has an encryption algorithm and an inverse algorithm for decryption.

Stream cipher - reminder

A symmetric key cipher where instead of using the key to directly encrypt the plaintext, the key is used to derive a pseudorandom string. Each section of the derived string (usually bits), is used to encrypt a corresponding section in the plaintext (usually using a XOR operation). The same key must never be used twice.

Block cipher mode of operation - reminder

An algorithm that uses a block cipher but adds functionality such as confidentiality or authenticity.

In our project, we use modes that are essentially A block cipher acting like a stream cipher. these are algorithms that use a block cipher per block, but add a unique Initialization Vector (IV), so that even though the same key is used for all the blocks, the ciphertext will be different, regardless of the plaintext. For this to work, the IV must be non-repeating.

CTR

A type of "block cipher mode of operation". In this mode, the IV is chosen (unique, of course) and for every block i, the effective IV is IV+i. meaning there is no dependency between blocks. Because of this, given an IV and a key, it is possible to choose which blocks to decipher when (to decipher block i, just use IV+i and decrypt away). Parallel deciphering is also possible for this reason.

We use this mode for encrypting and decrypting the "Master Key" (defined in "*Key Management*"), because the amount of information in the Master Key is larger than a block, and there is no need for error checking in this case (explained in "*Key Management*").

GCM

Uses the CTR mode of operation, and combines it with "Galois Message Authentication", which combines each ciphertext block with authentication data to produce an authentication tag. This tag can be used to authenticate the message, making sure no tampering or errors have occurred.

We use this mode for encrypting and decrypting the files themselves (see "*Key Management*"), because the files are (usually) larger than a block and we need error checking that won't reveal anything about the plaintext. Also, the error checking is faster than just doing CTR+HMAC because the authentication occurs during the decryption process (theoretically).

Scheme creation & Parsing

To properly decrypt a cipher using GCM, there is a need for more than just a key. An IV is also necessary, as well as the tag. Both of these aren't secret and therefore can be saved plainly in the file that contains the cipher.

It is also necessary to add the encryption method and mode to the scheme, if more than one combination is possible.

Parsing (in this context) is the process of extracting the meaning of the scheme from its respective string. For example, given this example string -> algorithm always starts from the 11th character, and ends with a semicolon. The characters in-between will be saved, as they are, and used to define the method of decryption.

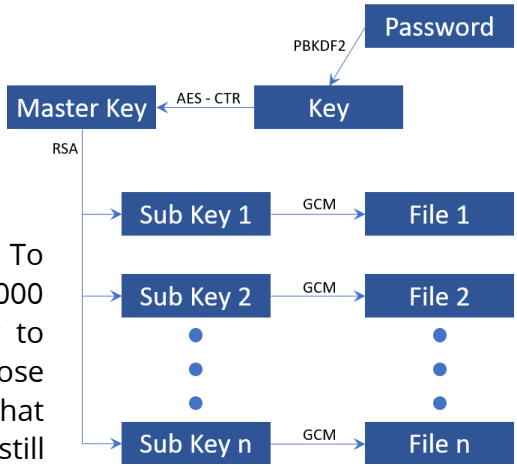
EXAMPLE OF A SCHEME

```
algorithm:AES256;
mode:GCM;
iv:IV_EXAMPLE_1234;
flid:FLID_EXAMPLE_45;
tag:TAG_EXAMPLE_123;
```

In our project, the encryption method is always GCM in AES256 mode, so the first two lines are cosmetic, but have the added bonus that decryption of the file doesn't have to be exclusive to our tool.

Key Management

The system which will securely generate, store and share cryptographic keys.



Our key management system is multi-tiered, and all the information is saved in a KSF file.

1. *The Password* – most likely not very secure. To mitigate this issue, we use PBKDF2 with 100,000 iterations and 16-byte random salt in order to make dictionary attacks less easy. We chose 100,000 because a typical computer can do that many iterations without taking too long, while still being over 10,000 which is the recommended minimum. For the hashing, we chose

SHA-256 over SHA-512 for its higher memory usage, making it potentially harder to create a quick dictionary. The key derived from the password is used to decrypt the next tier:

2. *The Master Key* – 2048-bit RSA. The minimum required for security, but enough for this project. Both the public key and private are stored together, but the private key (and other secret stuff) is encrypted using AES-CTR, with the aforementioned derived key and another random 16 bytes for the IV. There is no need for authentication, as any errors will become obvious if the decrypted key does nothing. To check if the password is correct, the first 4 bytes in the plaintext will be “GOOD”. If that is messed with so that a password will be incorrectly identified as correct, the key simply won’t decrypt properly, and the files will remain secure. It is still possible however, to manually destroy the keys or files using a text editor, so keep the files safe and backed-up. The decrypted private key will be used to decrypt each of the keys present in the next tier.

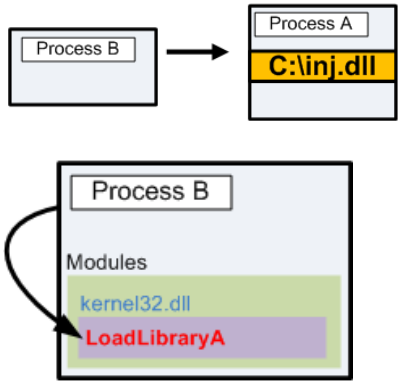
3. *The File Keys* – each of them is the key for decrypting their respective file. Each key is unique and randomly generated. The files are encrypted using GCM to allow error checking with a TAG that won’t give away information about the plaintext. The keys are encrypted using the Master Key and stored in the KSF file, along with the file ID and TAG. Both the ID and TAG will be stored in the file as well, and will be used to verify which key belongs to which file.

Procedure

DLL Injection

This is done by:

1. Finding the target process' PID by taking a snapshot of the current running processes and finding the one who's name fits the name of the program we want.
2. Allocating memory in the target process that will contain the path of the DLL to be injected
3. Finding the address of the *LoadLibrary* function (which definitely exists in the process – it is in kernel32.dll). This function can allocate memory for, and load DLLs into the program.
4. Creating a new thread in the target process that starts at the *LoadLibrary* function's address, and has the allocated DLL path address as an argument. This will initiate the loading of the DLL.



5. The main function in the DLL will be invoked by the operating system, and from there we can force the target process to do anything we want.

API Hooking

The method we used to create a hook uses the following steps:

1. Using IDA, (with the help of the program's source code) we find the functions that are optimal hooking candidates (In our case these are the load and save functions).
2. Since hooking involves adding a jmp command (and we don't want to mess with the offsets of the entire program), our jmp needs to overwrite some command. Our victim command (or group of commands) must have a size of at least 5 bytes (jmp is 5 bytes), and must be in a location where we can easily extract and/or implant the information we want.
3. We take note of the relative addresses of these commands, as well as their respective size in bytes.
4. In the injected DLL (who is now allowed to edit the memory), we find the base address of the EXE that called it – our target program
5. for each of the addresses we noted above, we find the absolute address by adding them with the base address. we also find the address we will return to when we're done – which is just adding the size of the overwritten commands to the absolute
6. we create a function for each of the hook spots, but we must also replace the commands we will overwrite, as well as make sure no registers are changed due to our actions. Failing to do so will probably cause the target program to crash, if not malfunction. For this we use naked inline assembly, so the compiler won't add potentially destructive "fixes". At the end of the function we put a jmp that will go back to the original hooked function, restoring the flow of the program.
7. We calculate the relative address of the function we created compared to the hook spot (while taking into account that jmp is 5 bytes).

8. For each candidate, we overwrite the commands, first with E9 (jmp), and then with the relative address we just calculated. This will force the program to go to our assembly function, which will in turn call our real function written in c++. This redirection will be applied for the time that the program is running, so every time the hooked function is called, it will redirect to our assembly and then go back to running as usual.

How to use the project

Compilation

Make sure the project is in x86 mode. Tested with Visual Studio 2017.

Running

1. Run our program: *NoteHack++.exe*
2. On first run, a ksf file must be created- enter a password in the prompt
3. On runs that aren't the first, when opening a file that was created with NoteHack++, a password must be entered.
4. Opening any other file will raise no such prompt on load, but saving will require a password.
5. Every time a password is requested, there is an option to change the password, by clicking "yes sir" in the prompt.
6. The KSF file is personal and will only work for one person. It must be kept in the program folder.
7. The included DLL must be in the program folder as well.
8. Enjoy your super-secure editor.

Future Development

Collaboration

We have not implemented this due to time constraints, but we did have the time to create the system in a way that will allow for easy implementation of this feature down the line.

We defined profiles that can be personalized per-person, each having a unique asymmetric key pair. If a file key were to be encrypted using a chosen user's public key, the file key will be available to that user, without disclosing the sharer's password. There is no need for the collaborator to change his password either, because his existing private key is all that is needed.

Features

- One password to rule them all – *no need to remember how to unlock every one of your files. Just one password and your files are secure.*
- Easy password change – *changing your password is as easy as clicking a button. Conveniently found every time you enter a password*
- Simple, friendly interface – *take your notes. simple as that. No hassle*
- Easy activation – *just click on **launch** and you're good to go*

- Full multi-tab support – *open any number of tabs you want. All of them are secure*

Known Bugs

- When a password prompt is open, using the "*ctrl+shift+esc*" shortcut to open Task Manager (or anything replacing it) will cause a crash. Opening Task Manager any other way has no such effect on the program.
- If a version of notepad++ other than 7.5.1 is installed, there may be unknown side effects or failure.

Division of Labor

Gilad

1. Hooking, injecting, correcting stack.
2. Interfacing the two parts of the project
3. Debugging & refactoring

Neria

1. Integration of libraries with project.
2. Programming crypto functions (RSA, AES, GCM, PBKDF2).
3. Writing unit-tests.

Both

1. Designing crypto scheme (JSON like structure, including this fields: algorithm, mode, iv, file-id and tag)
2. Designing windows (main window, enter password, about etc.)

Bibliography

Apriorit *https://www.apriorit.com/dev-blog/160-apihooks*

AxCrypt *http://www.axcrypt.net/forums/topic/how-to-change-main-password-on-axcrypt-2-x*

Code Guru *http://forums.codeguru.com/showthread.php?118308-How-do-you-make-a-Dialog-Box-from-a-Console-application*

https://www.codeguru.com/cpp/misc/samples/article.php/c4687/PasswordSpy--Retrieving-lost-passwords-using-Windows-hooks.htm

https://www.codeguru.com/cpp/i-n/ieprogram/security/article.php/c4387/Peeking-into-Password-Edit--Internet-Explorer--Super-Password-Spy.htm

Code Project *https://www.codeproject.com/Articles/227831/A-dialog-based-Win-C-program*

http://www.codeproject.com/KB/edit/SecEditEx.aspx?

http://www.codeproject.com/KB/edit/secureedit.aspx

https://www.codeproject.com/Articles/10137/CSecureEditEx-A-More-Secure-Edit-Control

Crypto Sense *https://cryptosense.com/parameter-choice-for-pbkdf2/*

CS OpenCW *https://ocw.cs.pub.ro/courses/_media/so/laboratoare/resurse/home/hotpatch.png?cache=*

Drazzy *http://drazzy.com/e/espruino/PicoRam/espruino__pico_bigram.bin/libs/crypto/jswrap_crypto.c*

MbedTLS Forum *https://tls.mbed.org/kb/how-to/reduce-mbedtls-memory-and-storage-footprint*

Msdn *https://msdn.microsoft.com/en-us/library/ms645489(VS.85).aspx*

https://msdn.microsoft.com/en-us/library/w0kywcw3.aspx

https://blogs.msdn.microsoft.com/oldnewthing/20050330-00/?p=36023

http://msdn.microsoft.com/en-us/library/6e36b89f(VS.80).aspx

http://msdn.microsoft.com/en-us/library/ms908376.aspx

http://blogs.msdn.com/oldnewthing/archive/2008/10/10/8969403.aspx

MQL4 *https://docs.mql4.com/common/messagebox*

Open Security Research *http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html*

Rohitab *http://www.rohitab.com/discuss/topic/40060-extendedhook-functions-c/*

http://www.rohitab.com/discuss/topic/40192-header-file-for-api-hooking/#entry10106168

Stack Exchange *http://security.blogoverflow.com/2013/09/about-secure-password-hashing/*

https://security.stackexchange.com/questions/71766/can-the-iv-salt-be-the-same

https://crypto.stackexchange.com/questions/47517/is-pbkdf2-rfc-2898-broken-because-sha1-is-broken

https://crypto.stackexchange.com/questions/48407/should-i-be-using-pkcs1-v1-5-or-pss-for-rsa-signatures

https://crypto.stackexchange.com/questions/35214/security-of-aes-ctr-with-multiple-messages-containing-the-same-known-plain-text

Stack Overflow *https://stackoverflow.com/questions/21718027/getmodulehandlenull-vs-hinstance*

https://stackoverflow.com/questions/33225800/disable-control-character-input-from-keyboard-shortcuts-in-notepad

https://stackoverflow.com/questions/1074362/embedded-resource-in-c

https://stackoverflow.com/questions/27816110/usage-of-getdlgitemtext-in-win32-apps

https://stackoverflow.com/questions/61634/windows-api-dialogs-without-using-resource-files

Wikipedia *https://en.wikipedia.org/wiki/Galois/Counter_Mode*

https://en.wikipedia.org/wiki/Block_cipher

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

https://en.wikipedia.org/wiki/Cryptographic_hash_function

https://en.wikipedia.org/wiki/DLL_injection

https://en.wikipedia.org/wiki/Hash_function

https://en.wikipedia.org/wiki/Key_derivation_function

https://en.wikipedia.org/wiki/Key_management

https://en.wikipedia.org/wiki/PBKDF2

https://en.wikipedia.org/wiki/Salt_(cryptography)

https://en.wikipedia.org/wiki/Stream_cipher

YouTube *https://www.youtube.com/watch?v=jTl3MFVKSUM*

https://www.youtube.com/watch?v=b1ahj347pDc

https://www.youtube.com/watch?v=knPMS01QbxU