From simple keylogger to advanced logging

Dmitri Gabbasov

December 16, 2014

1 Introduction

The aim of this project is to first create a simple key logger for the Microsoft Windows platform, and then see in which ways can it be made more powerful and less easily detectable. In this report we describe the implementation process and the design decisions of the keylogger. Following is the plan according to which the keylogger is to be developed.

The first step is to write a simple program that just logs all the keypresses along with their timestamps to a file.

Next, encryption of logs should be added to the keylogger. The logs are to be encrypted with a symmetric cipher using a session key, which in turn is encrypted with an asymmetric cipher using a predefined public key.

The third step is to create an installer for the keylogger. The role of the installer (one can also think of it as a builder or a customizer) is to create a customized keylogger executable that has certain parameters (log rotation frequency, public key used for encryption etc) embedded into it.

The next addition is about changing the simple file based logs into something more "stealthy". In particular, we consider using NTFS alternate data streams.

The fifth step is to implement the uploading of logs to some sort of online storage. Important thing to keep in mind during this is to avoid using services that would allow easy identification of the attacker. The solution also needs to gracefully handle periods of time when no internet connection is available.

The next part is to write a decrypter for the logs, to make it easy to read the encrypted logs from other machines.

As the seventh step, it is necessary to implement hiding features such that the running keylogger process wouldn't be as easily detectable on an infected system. This involves, at a minimum, hiding the process from Windows Task Manager.

As a bonus, the BadUSB [1] vulnerability should be exploited to create a USB flash drive that would install the keylogger on the machine that it is plugged in to.

The end result of this work did not, unfortunately, manage to implement the aforementioned plan in full. In particular, the last two steps (hiding and BadUSB) were not done. Also the uploading of logs does not work reliably due to CAPTCHAs and daily limits.

2 Implementation process

2.1 Capturing keys and basic logging

Keypresses are captured by using the Windows hooks API [2] to monitor the system for low-level keyboard input events. It is also possible to get the actual character that is represented by a given keypress taking into account the locale of the process that owns the foreground window (the Q key produces different letters for US and Russian keyboard layouts, and different processes may have a different active locale).

The information about every keypress is saved to a plain text file in the current working directory. Basically, each line in the file contains the timestamp, the type of event (key down or key up) and the key name (for non-character keys) or the actual character produced by the given key.

One may question the decision of saving the information as text as opposed to a binary form (which would be far more space efficient). The author agrees, that a binary form is more suitable, however the format was kept as text for simplicity of implementation. The modifications required to support a binary format are trivial and can be implemented at a later time.

2.2 Encryption of logs

Instead of logging keypress events as plain text, we changed the keylogger to encrypt them. The purpose is to make sure that no one can have access to the logs and identify their contents other than the attacker.

We also introduced log file rotation whereby a new log file is created every so often, while the old one simply gets renamed and put aside. When logging starts to a new file, a random 256-bit AES key is generated. It is encrypted with a predefined RSA public key and saved in the beginning of the log file. The initialization vector (IV) is saved right after the encrypted key. After that, the following stream of log messages is encrypted with the AES key into that file, until the next file rotation.

It is worth pointing out that the main purpose of file rotation is to facilitate AES key renewal. If we wanted, we could also renew keys inside a single log file.

To read the logs, one has to first read the encrypted key and the IV from the start of the log file. Then, one has to decrypt the AES key using a predefined RSA private key. After that, the decryption of log messages is straightforward.

2.3 Installer

As the next step, an installer was implemented that would allow embedding desired properties and values into a compiled keylogger executable. The installer reads input from a JSON file, that can have the following format:

```
{
    "rsa_key_path": "key.pem",
    "log_container_file": "C:\\Windows\\system32\\ikg.pdb",
    "log_rotation_period": 3600
```

```
}
```

The meaning of the properties is as follows:

- `rsa_key_path` – the path to the RSA key file, the public key will be embedded in the keylogger
- `log_container_file` – the file on the infected system where the logs are going to be stored
- `log_rotation_period` – number of seconds between log file rotation

Once the properties have been embedded in the keylogger, it will be able to read them at runtime and change its behaviour according to the provided values.

2.4 Stealthy log files

For the next part, we started saving logs to alternate data streams [3] of an existing file, the path of the file is configurable through the installer, but should probably be one that is guaranteed to exist on any Windows system. Alternate data streams are supported only on the NTFS file system, which has been the default filesystem of the Windows NT family since Windows NT 3.1.

By default, the contents of a file are stored in the so called unnamed data stream, but a file can also have any number of named data streams. Using the native Windows file APIs to read files, it is possible to access the named streams by specifying a filename of the form `file.txt:streamname`.

Log rotation works very well with streams too. Every time a new log is created, it is saved to a new stream in the container file. The name of the stream represents the timestamp when the log was rotated.

2.5 Uploading logs to the internet

The general idea of this part is to upload existing logs to the internet whenever an internet connection is available. The key problem here is to avoid using any services, that would allow easy identification of the attacker who deployed the keylogger and is monitoring the logs. This basically rules out any services where one has to register an account.

One approach is to upload the files somewhere where all uploaded content is publicly visible, for example `http://pastebin.com/`. The problem with pastebin.com (and many other similar services) is that every uploaded paste ends up having its own unique URL, which the attacker does not know in advance. However, even if we did have full control over the resulting URL of the uploaded content (which is possible for some services) it is still non-trivial to come up with a URL scheme that would allow to gather all logs of every user.

Instead, we use the service `http://justpaste.it/`, which is similar to pastebin.com, but allows to specify the desired URL (the path component) and, more importantly, allows to edit existing pastes, assuming you know the special "editing" URL. We use one "master paste" that is modified by a keylogger each time it uploads a new log – the keylogger simply appends the URL of the newly uploaded log to the master paste. The logs themselves are uploaded to pastebin.com, however it is also possible to extend the keylogger to use any other file or text upload service for the storage of logs.

The logs on pastebin.com are stored in Base64 encoded format, and thus need to be decoded before they can be processed by the log viewer.

It is worth mentioning that log uploading is plagued by things like CAPTCHAs (in case of justpaste.it) and daily quotas (10 pastes per day on pastebin.com). Upon an unsuccessful attempt to upload a log, the keylogger will simply wait and try again later. This seems to mostly work.

2.6 Log viewer

A simple log viewer is implemented, that takes an encrypted log file and outputs its plain-text contents.

2.7 Hiding the process and infecting through USB

The last two parts – hiding the keylogger process from the list of processes and exploiting the BadUSB vulnerability as an infiltration technique – were not implemented.

3 Conclusion

We started with a simplistic keylogger and empowered it by adding features such as log encryption, customization through embeddable properties, usage of alternate data streams of system files for storage and uploading of logs to online services. Unfortunately, due to insufficient time, we did not manage to implement the hiding of the process, neither did we get to exploit the BadUSB vulnerability.

References

[1] Turning USB peripherals into BadUSB. *Security Research Labs*. `https://srlabs.de/badusb/`

[2] Hooks. *Windows Dev Center, MSDN*. `http://msdn.microsoft.com/en-us/library/windows/desktop/ms632589.aspx`

[3] John Marlin. Alternate Data Streams in NTFS. *Ask the Core Team, TechNet Blogs*. `http://blogs.technet.com/b/askcore/archive/2013/03/24/alternate-data-streams-in-ntfs.aspx`