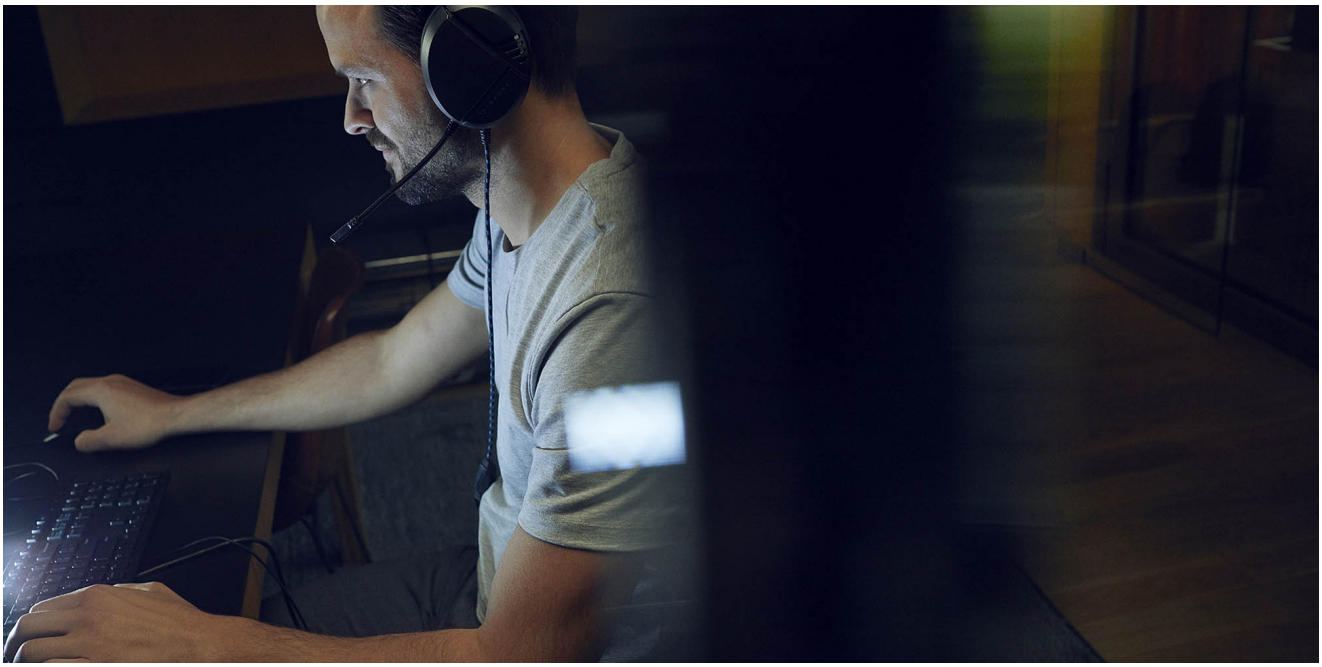
The big difference with Bad Rabbit

labsblog.f-secure.com/2017/10/27/the-big-difference-with-bad-rabbit/

October 27, 2017



Bad Rabbit is the new bunny on the ransomware scene. While the security community has concentrated mainly on the similarities between Bad Rabbit and EternalPetya, there's one notable difference which has not yet gotten too much attention. The difference is that Bad Rabbit's disk encryption **works**.

EternalPetya re-used the custom disk encryption method from the original Petya. Although it didn't implement the actual ECDH key delivery mechanism, it installed the Petya boot loader, and effectively just rendered the machine useless.

Petya's disk encryption had one specific weakness: it only encrypted some parts of the key file system structures, not the whole disk. This design obviously lead to speculations about whether it is possible to recover the disk using a known clear-text attack, and in fact researchers have made significant progress in investigating this recovery technique.

At least on the surface, things look quite different with Bad Rabbit. Instead of using a custom encryption mechanism, it follows the current trend in the ransomware community of leveraging known legitimate encryption tools.

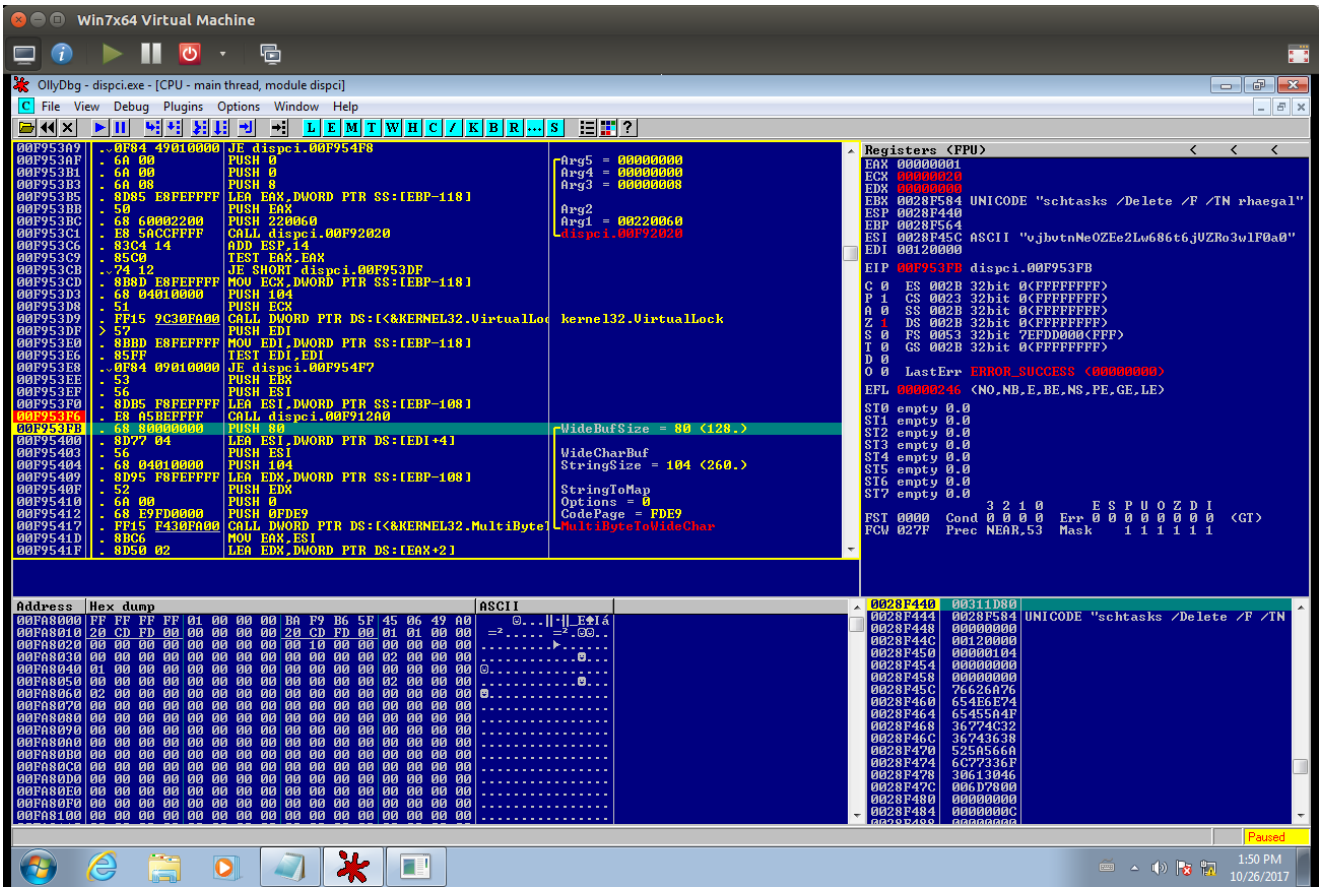
Bad Rabbit uses DiskCryptor, a full disk partition encryption software for locking the user disk. The ransomware ships with an unmodified DiskCryptor driver (borrowed from ReactOS) and implements relevant parts of the DiskCryptor user-mode code for communicating with

the driver. The ReactOS driver is a signed, valid driver, so it gets loaded by the Windows cleanly with the elevated privileges required by Bad Rabbit's fake Flash installer dropper.

Key generation

Bad Rabbit uses Windows' crypto API to generate random key data for the disk encryption. This key data is converted to a human-readable encryption key, which is a 32-bytes long ASCII encoded string, presenting around 165 bits of entropy for the stream cipher AES used by DiskCryptor.

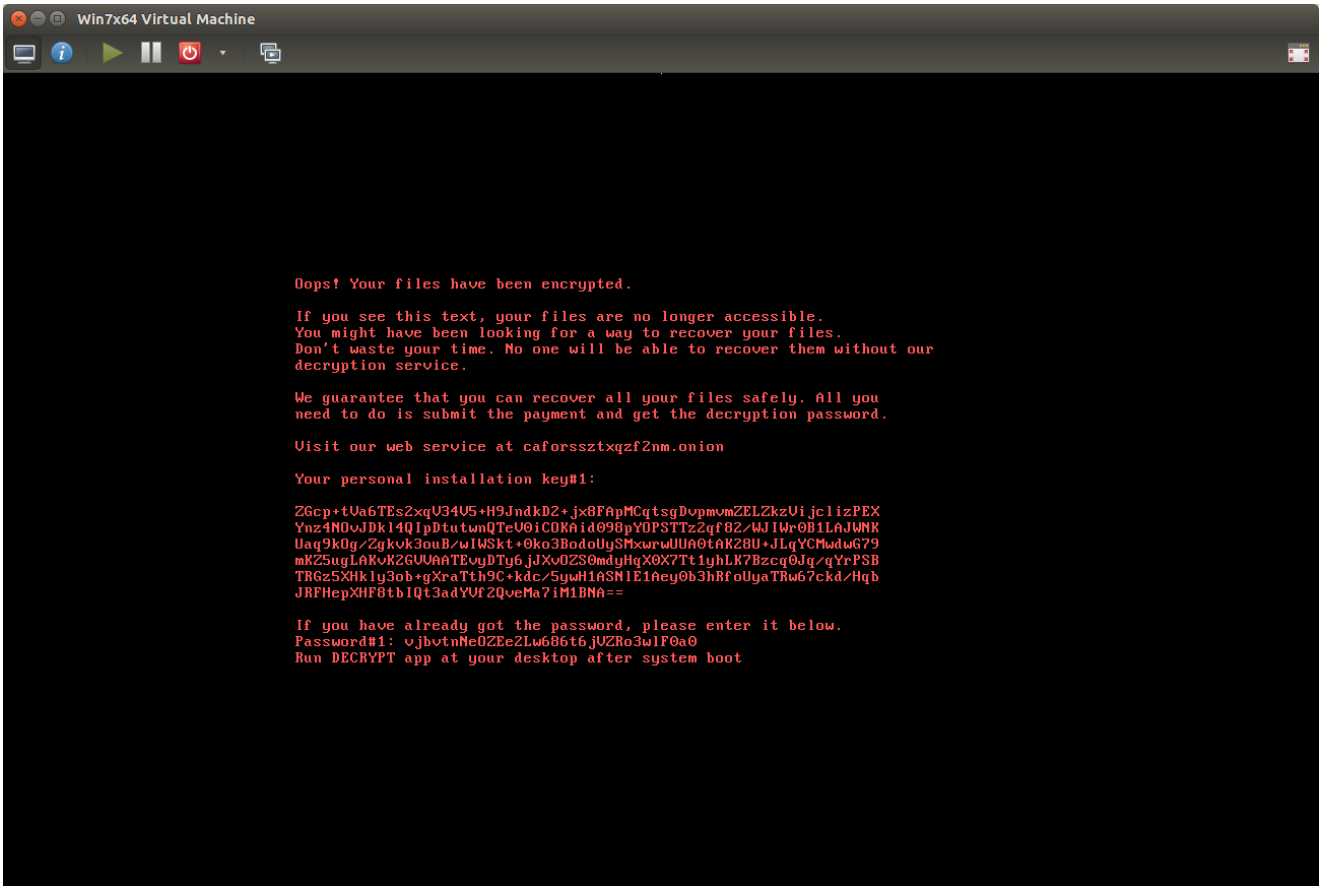
The following screenshot represents the random disk encryption key as it is being generated by the malware:



Running dispci.exe under user-mode debugger

Along with the random key, Bad Rabbit packages some other information like the victim computer name and domain, then forms the so-called installation key that will be presented after the reboot. In Petya, this code was relatively short code that was protected by the public EC key. In Bad Rabbit, it is a much longer blob of data that is protected by the RSA public key shipped with the malware installer.

After packaging the installation key, the random key data is just discarded. The installation key is written to the disk so that the Bad Rabbit boot loader can present it on the boot screen:



Bad Rabbit's boot loader

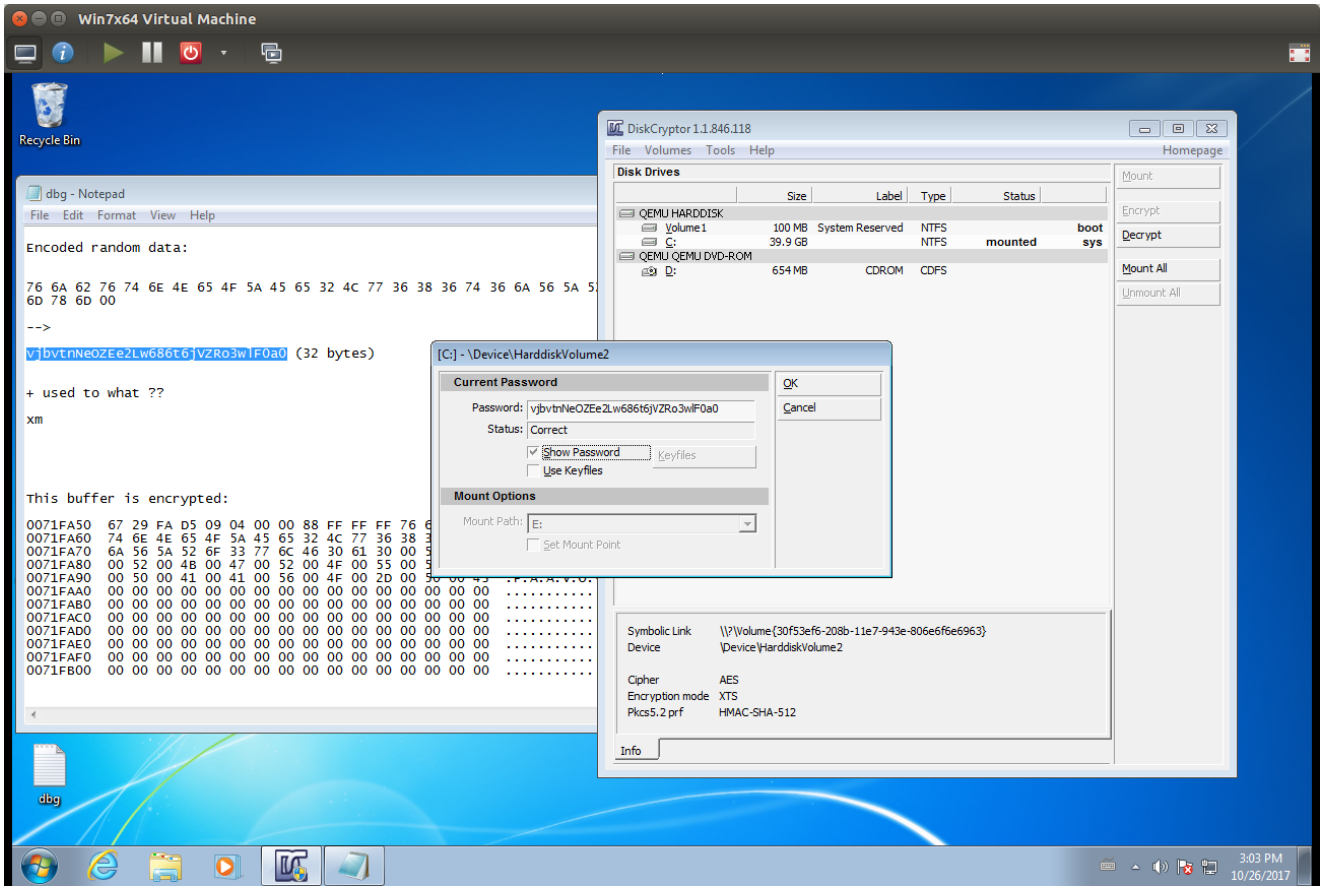
The user is expected to grab the installation key from the boot screen, paste it to the attacker's TOR site, which then will use its own private RSA key to extract the 32-bytes password (typed in the screenshot above).

After the boot screen

Because the disk encryption software used is pretty much similar to any other disk encryption used by businesses around the globe, like TrueCrypt or Microsoft BitLocker, the disk is in fact *still* encrypted, although it has been mounted by the DiskCryptor driver transparently.

So if the user now reboots the machine, the same password prompt will be presented as long as the disk decryption routine is initiated. The DECRYPT tool referenced by the boot loader is actually just a shortcut to the dispci.exe tool dropped by the malware. This tool borrows code from legit DiskCryptor sources for implementing the relevant parts of the code for communicating with the disk encryption driver.

Even though all this is quite apparent by just looking at the code, we wanted to demonstrate the encryption scheme by catching the password inline, at the time it was generated by Bad Rabbit. When this password is used for unlocking the machine, it is possible to install the real DiskCryptor GUI tool and initiate the disk decryption process.



Using DiskCryptor to verify encrypted volume

DiskCryptor identifies the disk presented by the virtual machine (QEMU HARDDISK in the above screenshot) as an AES-encrypted volume, and accepts the random password.

It works now

We have speculated before that the flawed disk encryption in EternalPetya was due to problems in the malware development process. Or it may be that they just didn't care. People will pay anyways, right?

Whatever was the reason, they have now fixed this issue (if they are the same group of malware developers, which seems to be the consensus in the research community).

At least the developers of Bad Rabbit have noted the recent developments in research on Petya's disk encryption weaknesses and decided to use something different.

Recovery considerations

As we have demonstrated in this blog post, Bad Rabbit seems to use a sound principle in its disk encryption, a full disk encryption scheme familiar to all businesses.

We don't yet have the full details of this scheme, so there might be bugs in the implementation. But at least its design enables a strong mechanism for locking the machine until the correct password is really typed to the boot screen.

Disclaimer

For screenshots used in this blog post, we DID NOT go to the attacker TOR site and pay for the recovery key.

The procedures presented in this text DO NOT mean there's an easy way to unlock the disk protected by the Bad Rabbit. We just present them as proof of the encryption scheme. Catching the password inline to the encryption process is not practical in a general sense, because it requires software that is aware of the exact password generation mechanism **prior to the infection**. It is used here just for a relatively easily reproducible proof-of-concept.

Categories

Threats & Research