FEATURE 1

NEW PERSISTENCE THREATS

Eric Filiol
ESAT, France

Recent research has shown that, contrary to popular belief, the content of computer memories (RAM) is not erased when a computer is shut down. Different kinds of data can survive even after events that should normally result in the RAM being erased and reset to zero: program termination, shutdown or switching off the computer. The survival of data in RAM may not only affect the security of cryptographic applications but may also be used efficiently to design new, powerful malware threats.

This article first presents an in-depth analysis of computer memory weaknesses that could enable the theft of sensitive data via malware attacks. Most of these attacks are made possible due to the persistence properties of modern computer memory modules. In the second part of the article, we present different attack methods that have been identified and tested and which could be used maliciously. In the final part we present some tools and security policy enhancements that should greatly contribute to preventing or limiting those attacks.

THE PROBLEM: PERSISTENCE OF COMPUTER MEMORY MODULES (RAM)

State of the art: memory remanence

For a long time it was widely believed that computer memory modules (aka Random Access Memory or RAM) were erased (reset to zero) immediately after a program terminates or a computer is shut down, thus causing their content to disappear from the computer. However, a number of studies have shown this assumption to be partially wrong. A number of studies [1–3] have identified risks attached to what is known as memory remanence:

> '...Ordinary DRAMs typically lose their contents gradually over a period of seconds, even at standard operating temperatures and even if the chips are removed from the motherboard, and data will persist for minutes or even hours if the chips are kept at low temperatures. Residual data can be recovered using simple, non destructive techniques that require only momentary physical access to the machine…' [3]

The authors of [3] observed a data remanence effect at normal operating temperatures (between 25.5 °C and 41.1 °C) after 2.5 to 35 seconds (depending on the computer) with a binary error rate ranging from 41% to 50%. They managed to increase this remanence time to 60 seconds with a very negligible error rate, simply by cooling the RAM at a temperature of -50 °C.

From those results, the researchers identified a number of security risks with respect to data remanence. In particular, they explained how secret cryptographic keys could illegitimately be retrieved by exploiting the RAM remanence property. Despite the undisputed interest of this study, its operational scope is rather limited: the attacker must have physical access *and* must cool the RAM immediately after a sensitive application has been executed (e.g. encryption/decryption). Except in the case of investigation by police forces, this attack remains of theoretical interest only.

At the time of publication of [3], another team was working on the same subject but with a broader, more operational approach and at normal operating temperatures by considering the concept of RAM persistence [4].

Memory data persistence

RAM data remanence considers only the physical, electronic effects that enable data to survive temporarily in RAM. But data disappearing from memory does not necessarily mean that the data has disappeared from the computer, and in many cases, memory contents remain available inside the computer for a very long time: we call this *memory data persistence*. Let us adopt the following definition:

Memory data persistence [4]: the set of both physical (remanence) and operating system effects/mechanisms that cause data to survive in RAM and/or in a computer after a program terminates or a computer is shut down.

Without entering into too much detail, besides the single remanence effects that have been confirmed and developed further, we have identified a number of other mechanisms which preserve the content of memory modules. The main ones can be summarized as follows:

- Swap files (the pagefile.sys file under *Windows*, and the swap partition under *Linux*) generally contain whole or part of the memory.
- Hibernation files (the hiberfil.sys file under *Windows*) contain a lot of memory data.

Data persistence mechanisms	Security risk
RAM remanence	1
Swap file	3 – 4
Hibernation file	2
Memory dump file	3 – 4

Table 1: Security risk with respect to data persistence (lowest = 0 highest = 4).

- The *Windows* memory dump file (MEMORY.dmp) contains the whole RAM content.

Table 1 summarizes the level of computer security risk attached to those mechanisms.

The different experimental results (see [4] for details) clearly demonstrate that the operating system saves the RAM content very frequently (wholly or partly) into dedicated files, thus causing critical data to survive far longer than expected, even after the computer has been rebooted. All of those mechanisms, besides the RAM remanence effect, constitute a critical risk against data confidentiality.

MALWARE-BASED ATTACKS AGAINST CONFIDENTIALITY

Let us first consider how a dedicated piece of malware could exploit the RAM persistence in a computer. In other words, we consider the different actions that a piece of malware could take to steal critical data with respect to memory persistence. We will present only a few of the most illustrative examples included in [4]. We will consider a piece of malware that is undetected by anti-virus products as a general framework.

Eavesdropping confidential data

In this case, the malware will look for sensitive data that survive either in memory (remanence) or in memory dump files. It is worth mentioning that the malware itself may induce the creation of such files.

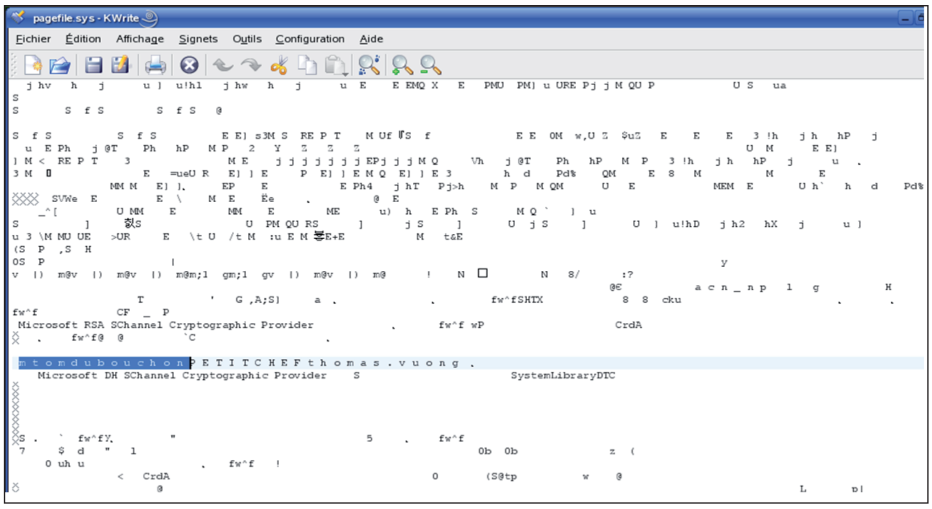
- Let us suppose that a secret (inert or not) file is processed (scanned by an anti-virus engine or processed by a dedicated application) on a computer that is infected with a piece of malware. A %SystemRoot%\MEMORY.DMP file is created. In most cases, this file will contain at least a significant part of the secret data. In some cases, it is possible to steal plain-text data during the decryption of an encrypted document.
- A piece of malware can explore the computer's RAM content directly in order to find secret data. Even after a few hours, in some cases, the information remains in memory. As an example, we plugged in a USB key containing a secret file and then unplugged it. It was still possible after the USB key has been removed to find a lot of data with respect to the file (the experiment can be reproduced by using the *WinHex* software which embeds a forensics function called '*RAM editing*').
- Secret data is also saved by *Windows XP* in the hibernation file HIBERFIL.SYS. Any piece of malware could very easily access this file and retrieve a lot of data that is contained in RAM when the computer goes into sleep mode. If sleep mode is not activated by default, the malware is able to activate it.

There are also many more ways for malware to collect sensitive data – even when it is protected by encryption – by exploiting the data persistence.

Theft of password or encryption keys

Now let us see how a piece of malware could collect critical data with respect to the security of the computer itself: password and encryption keys.

Analysis of the *Windows* swap file (PAGEFILE.SYS) or of the hibernation file may reveal such critical data, as well as



```
  j hv   h   j     u ]  u!hl   j hw   h   j      u  E    E EMQ X   E   PMU   PM] u URE Pj j M QU P          U S   ua
S
S      S  f S      S  f S   @

S  f S         S  f S        E E] s3M S  RE P T    M Uf ΓS  f            E E  OM  w,U Z  $uZ   E    E    E   3 !h   j h   hP   j
 u  E Ph   j @T   Ph   hP   M P   2   Y    Z   Z   Z                                                 U  M      E E]
] M <  RE P T    3            M E    j j j j j j j EPj j j M Q     Vh   j @T    Ph    hP    M  P    3 !h   j h   hP   j      u   .
3 M  ▯           E   =ueU R   E] ] E     P  E] ] E M Q  E] ] E 3       h  d    Pd%    QM     E  8   M        M       E
           MM M    E] ].     EP    E          E Ph4    j hT   Pj>h    M  P   M QM     U   E          MEM  E      U h`   h   d    Pd%
 SVWe  E           E  \    M  E    Ëe     .          @  E
  _^[        U MM    E        MM     E        ME       u)   h    E Ph   S      M Q `    ]  u
S          ]     S                 U  PM QU RS       ]        j S     ]      U  j S     ]       U  ]  u!hD    j h2   hX   j     u ]
u 3 \M MU UE   >UR      E   \t U  /t M   :u E M E+E          M    t&E
(S  P  ,S  H
OS  P                 |                                                                                   y
v  |)  m@v  |)  m@v  |)  m@m;l  gm;l  gv  |)  m@v  |)  m@       !    N  □        N   8/       :?
                                                                                       @€        a c n _ n p   l   g          H
               T           '   G ,A;S]      a  .              .          fw^fSHTX          8   8  cku                      .     .
fw^f           CF  _  P
 Microsoft RSA SChannel Cryptographic Provider             .          fw^f wP                  CrdA
   .    fw^f@  @          `C                .

 m t o m d u b o u c h o n P E T I T C H E F t h o m a s . v u o n g  .
   Microsoft DH SChannel Cryptographic Provider    S                      SystemLibraryDTC






 S .   `  fw^fY.        "                        5     .     fw^f
 7    $  d   "   l                                                 0b  0b            z  (
   0 uh u                     .     fw^f    !
             <   CrdA                                  0       (S@tp         w    @
                 @                                                                                L       p|
```

Figure 1: Session login password inside a PAGEFILE.SYS file.

it appearing in the %SystemRoot%\MEMORY.DMP file. As an example, let us consider the PAGEFILE.SYS file. The session password can survive totally or partially in that file, even after a reboot. The most interesting thing is that we can recover the passwords of different users (in multi-user mode). Figure 1 shows the presence of a session login password inside the PAGEFILE.SYS file (nine characters out of a total of 11 are recovered).

As for encryption keys, the data persistence (including data remanence) will depend partly on the security enforced at the application level. Tests have been conducted on different pieces of encryption software. For some of them, it is possible to retrieve wholly or partly the password used to protect private keys in public key encryption applications, and even the private key itself can be retrieved either in the HIBERFIL.SYS file or in the MEMORY.DMP file. In some cases, the private key may also be present inside the PAFEGILE.SYS file. Figure 2 shows the presence of the encryption password inside a *Windows* hibernation file after decryption with the open-source *Cryptonit* [5] software. (The same applies with various other encryption software packages.)

It is therefore essential to keep in mind that the security provided by the operating systems (some of the same results have been obtained under *Linux*) and/or the security software (e.g. encryption application) is not watertight and critical data such as passwords and encryption keys may be leaked. Even if such data is only partly recovered by a malicious attacker (most of the time the recovery rate is higher than 80%) it will be easy to guess the remaining part (e.g. using a reduced brute force approach).

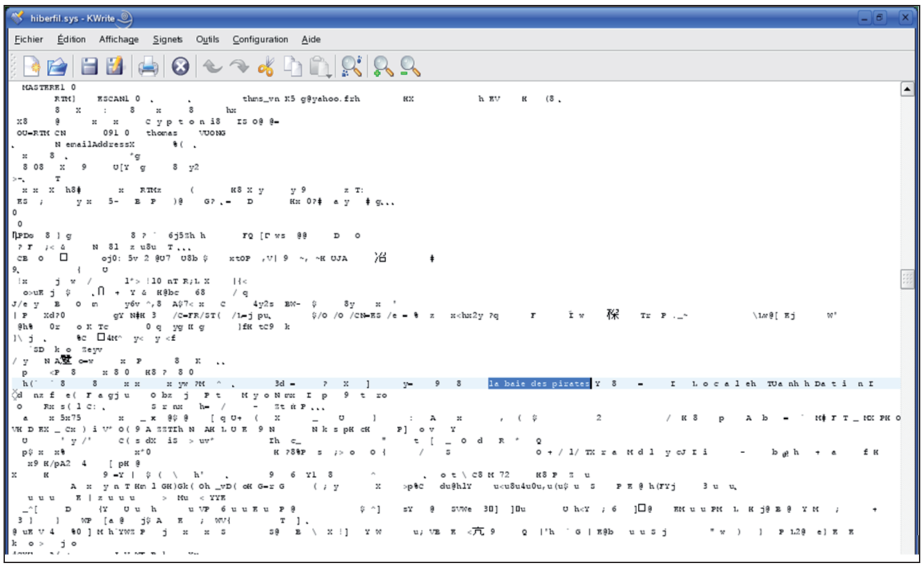


Figure 2: Cryptonit encryption password in a HIBERFIL.SYS file.

The other essential point lies in the fact that most of the system files we have considered (swap file, hibernation file, memory dump file) can be created by any malware itself. It only has to manipulate the appropriate system configuration files and access the appropriate system description tables (e.g. ACPI tables).

NEW MALWARE CONCEPTS EXPLOITING MEMORY PERSISTENCE

In this section we will explain how data persistence can be exploited by a piece of malware to replicate (self-reproducing codes) or just to operate (installation of simple malware such as trojans, logic bombs etc.). The payload will not be taken into account here. We will present a very simple, yet powerful proof of concept.

Before revealing the general mechanisms operated by our proof of concept, it is essential to make one very important point clear. For the attacker, the main problems with data persistence (especially the remanence part) lie in the fact that the data can only partly be recovered and in the fact that he *a priori* does not know what that data is. For example, in the case of a obtaining secret key, the exact location of the key and the amount of remaining information that needs to be guessed following data recovery may make the attack more complex than expected [3]. Moreover, the attacker cannot initially operate on the data that is supposed to be persistent in memory (using error correcting techniques for example).

But in the context of a piece of malware that is going to exploit data persistence to operate, the initial preparation of data used for that purpose is possible. In other words, the malware will always know what it is looking for and where to find it. We just have to use error correcting techniques to prevent data loss due to the natural and random limitation of data persistence (including data remanence).

The general design of the proof of concept combines the data persistence effects with the most sophisticated malware techniques that have recently been identified:

- *K*-ary codes [7, 8]. Instead of having a single file containing all the malicious information, *k*-ary malware are composed of *k* different files, each of which looks innocuous. A suitable combination – either serially or in parallel – of (at least) a subset of those *k* parts results in the malware operation. In [8], it was shown that detecting *k*-ary malware is an intractable problem. One very interesting approach for malware is both to split the malicious information and to introduce a time delay between them. In this respect, data persistence can provide a very powerful set of techniques to realize such codes. A significant subset of those *k* parts can simply be persistent data either in memory (remanence) or in some system files.

- Strongly armoured codes [6, 7]. Such malware is encrypted with strong algorithms (e.g. AES, RC6, and Serpent), but unlike most encrypted malware the secret key is not stored inside the code. In this setting, the key is only available as a quantity taken from the environment and is basically under the attacker's control. In our context, this key may be taken from data that is known to be persistent in the computer at a given time or after a given event, under the attacker's control.
- Cryptography-based obfuscation techniques [7, 9]. This approach is quite similar to the previous one, however in this case it is not the key which is the information taken from data known to be persistent in the computer at a given time/after a given event, but the obfuscation algorithms themselves.

All these techniques have been tested and have proven to be very efficient. This shows that data persistence represents extraordinary potential for developing existing malicious techniques further, in a very sophisticated way.

PREVENTION

The essential question is: how can we prevent or limit the exploitation of data persistence by malware?, since detecting such sophisticated code is bound to be a very complex challenge.

The following methods should greatly contribute to preventing such attacks:

- Anti-virus software should scan the entire memory systematically and not only the memory actually used. Critical system files (hibernation file, swap file or area, memory dump file) should also be checked systematically.
- Critical configuration files managing the creation of those files should also be protected by a suitable security policy. Anti-virus software should warn against any unsuitable configuration for those files with respect to data persistence.
- Critical software (for example encryption software) should be implemented securely. Before terminating, the physical memory that has been used should be erased securely in order to prevent data remanence. Critical data (such as cryptographic keys) should be locked into memory in order to prevent information becoming available via the swap file or error file (e.g. CORE file under *Linux*). Most high-level programming languages contain suitable primitives that can be used to achieve this.

REFERENCES

[1] Gutmann, P. (1996). Secure deletion of data from magnetic and solid-state memory. Proceedings of the 6th USENIX Security Symposium, pp.77–90.

[2] Gutmann, P. (2001). Data remanence in semiconductor devices. Proceedings of the 10th USENIX Security Symposium, pp.39–54.

[3] Halderman, J.A.; Schoen, S.D.; Heninger, N.; Clarkson, W.; Paul, W.; Calandrino, J.A.; Feldman, A.J.; Appelbaum, J.; Felten, E.W. (2008). Lest we remember: cold boot attacks on encryption keys. Available at http://citp.princeton.edu/memory.

[4] Filiol, E.; Tuccelli, C.; Vuong, T. (2008). Analyse de la mémoire RAM. Récupération de données par le phénomène de rémanence (RAM analysis: data forensics through data persistence). Technical Report ESAT 2008_M2.

[5] http://sourceforge.net/projects/cryptonit/.

[6] Filiol, E. (2004). Strong cryptography armoured computer viruses forbidding code analysis: the BRADLEY virus. Proceedings of the 2005 EICAR Conference. Available at http://vx.netlux.org/lib/aef02.html.

[7] Filiol, E. (2006). Techniques virales avancées. Springer, Collection IRIS, ISBN 978-2-287-33887-8. (An English translation entitled Advanced malware techniques is pending end 2008.)

[8] Filiol, E (2007). Formalisation and implementation aspects of K-ary (malicious) codes. EICAR 2007 Special Issue, Broucek V.; Turner, P. eds. Journal in Computer Virology, 3 (2), pp.75–86.

[9] Beaucamps, P., Filiol, E. (2006). On the possibility of practically obfuscating programs. Towards a unified perspective of code protection. Journal in Computer Virology, 2 (4), WTCV'06 Special Issue. Bonfante G.; Marion J.-Y. eds.