Black Hat USA 2011

Doing forensics in the cloud age
OWADE: beyond files recovery forensic

Elie Bursztein, Stanford University *elie@cs.stanford.edu*
Ivan Fontarensky Cassidian, *ivan.fontarensky@eads.com*
Matthieu Martin, Stanford University *mamartin@stanford.edu*
Jean-Michel Picod, Cassidian *jean-michel.picod@eads.com*

http://www.owade.org

Table of Contents

1 INTRODUCTION

The field of forensics is currently undergoing drastic changes due to the fact that most user activity is moving to the cloud. It used to be the case that analyzing a users computer was sufficient to reconstruct his activity, but now a large of user data is stored remotely. In the cloud and wireless age, reconstructing a users activity requires knowing where the computer was connected, which online services were accessed, and which online identities were used by the user.

Traditional forensic file based techniques are insufficient to extract the information needed to reconstruct the users online activity because such information is encrypted, obfuscated and scattered across multiples files and registry keys.

Add to this the fact that the encryption / obfuscation schemes used by various pieces of software tend to be different and it becomes clear that this type of advanced analysis is very challenging. For example, in order to extract the logins and passwords stored by Internet Explorer, one needs to crack the Windows user password, acquire the DPAPI (Data Protection API) master key, reconstruct the browsing history, and decrypt the Internet Explorer vault–and these steps describe the simple case, in which the user did not use Internet Explorers private browsing mode.

To reconstruct the users online activity from his hard-drive, we have developed OWADE (Offline Windows Analysis and Data Extraction http://www.owade.org), an open-source tool that is able to perform the advanced analysis required to extract the sensitive data stored by Windows, the browsers, and the instant messaging software.

OWADE decrypts and geolocates the historical WiFi data stored by Windows, providing a list of wifi points the computer has accessed (including the locations of the access points to within 500 feet) and when each point was last accessed. It can also recover all the logins and passwords stored in popular browsers (Internet Explorer, Firefox, Safari, and Chrome) and instant messaging software (Skype, MSN live, Gtalk, etc.). Finally, it can reconstruct the users online activity by reconstructing their browsing history from various sources: browsers, the Windows registry, and the Windows certificate store [1].

In certain cases, OWADE is even able to partially recover the users data even when the user has utilized the browsers private mode. OWADE is written in Python, runs on Linux, and only uses GPL libraries and software (John the ripper, dd_rescue). Its cryptographic engine is the first to fully re-implement the Windows DPAPI without using any windows dll files and is able to decrypt the Windows Credential store. Its registry analyzer is able to reconstruct the computer environment (hardware, network, user) almost perfectly and extract software information that allows OWADE to perform a vulnerability analysis post-mortem and to detect pirated software.

The remainder of this paper is divided into two parts:

In the first part, we discuss how OWADE extracts all of the information it needs from a hard-drive. We discuss how the files are recovered, which information is extracted from the registry, and how Windows passwords are recovered.

In the second part, we cover in-depth portions of the OWADE advanced analysis: we discuss how it is performed, which information it returns, and its limitations and challenges. We discuss the OWADE geo-localization module, the browser data extraction module, the instant messaging extraction module, and the software post-mortem vulnerability analysis module.

Before delving into the details we start with an overview of how OWADE works and a brief presentation of the Windows storage protection mechanisms

2 OVERVIEW

This section provides a brief overview of OWADE architecture and how it performs a forensic analysis.

2.1 OWADE Architecture

Early in the development process, we made the choice to design OWADE as data centric software, as its goal is to reconstruct user activity. Accordingly, the data OWADE collects are files, credentials and visited URLs rather than software information. For example, having an aggregate browsing history instead of having a separate history for each browser simplifies the analysis when the user employs multiples browsers. Similarly, a unified list of login and passwords is more interesting than one list for each piece of software, as users tend to reuse the same login and password on multiples services.

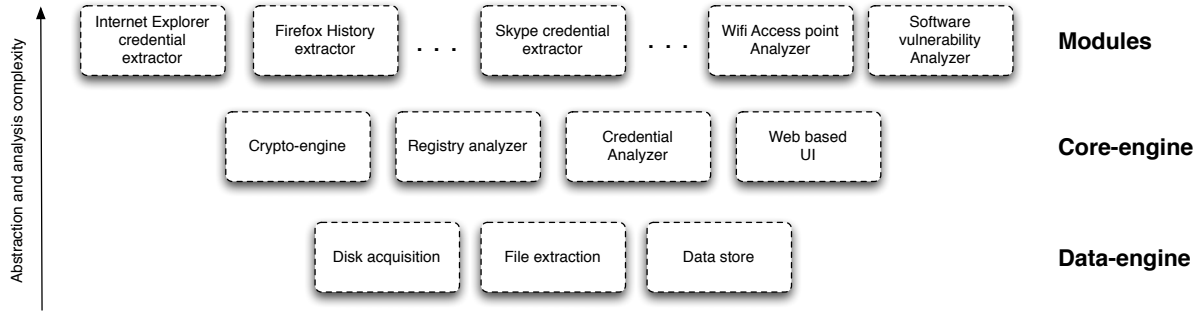

Figure 1: OWADE architecture overview.

As visible in the figure 1 OWADE is composed of three distinct layers of abstraction:

- **Data Engine**: The data engine is specialized in disk acquisition and file extraction.This engine is also responsible for storing OWADE analysis results.
- **Core Engine**: The core engine provides all the core functionalities needed to handle and process Windows information. The registry analyzer is used to extract the useful information from the registry and provide a robust way for modules to query the registry for specific keys. The crypto-engine is used to decrypt DPAPI data, the Credential Store, and the Protected storage. The Windows credential analyzer is used to extract and crack the Windows credentials. Finally the UI engine is used to control the analysis and display its results from a web page. The UI engine uses Django as a backend.

- **Modules**: The modules implement OWADE advanced analysis. Each module aims at gathering a specific type of information from a specific piece of software. OWADE has, for example, modules dedicated to extracting credentials from various software (Internet Explorer, Firefox, Skype) and storing them in the credential table. Having one module for each task makes them easier to maintain, improves OWADE's overall robustness and makes it easy to add a module that targets new software.

2.2 Analysis Overview

Overall a typical analysis is performed as follows:

1. **Disk Acquisition**: OWADE makes an image of the disk.
2. **File Extraction**: The disk files are extracted from the disk image using various techniques (see section 4)
3. **Registry Analysis**: The windows registry is analyzed and the relevant information is extracted. (see section 5).
4. **Windows Credential Recovery**: The Windows credentials are extracted and cracked (see section 6).
5. **Credential Store Recovery**: Using the recovered Windows credentials and the crypto-engine, the credentials saved in the Windows credential store (see section 3.2) are recovered.
6. **WiFi analysis**: The wifi information is extracted, the access point password recovered, and the access point geo-located (if a proper geolocalization API is available) (see section 7)
7. **Software analysis**: OWADE correlates the software version and serial numbers extracted from the registry with known vulnerability databases and pirated serial numbers to provide an analysis of the computer security posture post-mortem. (See section 10)
8. **Online activity recovery**: OWADE's various modules are used to reconstruct the user's online activity from the browser data (see section 8) and any instant messaging clients that the user may use (see section 9).

3 BACKGROUND

In this section we provide a quick overview of the Windows cryptographic mechanisms that are encountered while performing a forensic analysis.

3.1 DPAPI (Data Protection Application Programming Interface)

DPAPI (Data Protection Application Programming Interface) is the cryptographic programming interface offered by Microsoft since Windows 2000 for safe storage of sensitive data on disk. In a nutshell, DPAPI is a cryptographic scheme that provides a transparent way to encrypt data with a key derived in a secure manner from the user password. Because of the complexity of the encryption scheme, the DPAPI internals remained unknown for almost 10 years until we revealed how it works last year [8]. The reader interested in how DPAPI works is invited to look at our analysis. The two key points to know about DPAPI when doing forensics are:

1. The SHA-1 hash of the user's Windows password is needed to decrypt DPAPI protected data. As explained in section 6, the best option to get this SHA-1 hash is to crack the Windows password and hash it back into SHA-1. The other option is to try to find it in a Windows memory dump but this is less reliable.

2. DPAPI encryption offers programmers the ability to add an extra vector of entropy (their own salt). Accordingly, one of the key difficulties when attempting to recover the data from new software is to figure out how this extra entropy is constructed. Sometimes it is a constant string (Safari), and sometimes it is a very complicated algorithm (Google Talk).

Overall the OWADE crypto-engine makes it as easy to decrypt DPAPI data as it is to use the real API, provided that the user's Windows password and the additional entropy is known.

3.2 The Credential Manager

The *Credential Manager* [5] is the Windows framework used to store network based passwords in an encrypted format tied to the Windows user password. The credential manager is meant to provide a convenient and reliable mechanism to store the passwords for network logins so that users don't have to enter their password every time while accessing the network resources. Microsoft provides a standard API to store and retrieve passwords from the Credential store, which makes it easy for other applications to use.
Besides the fact that Windows uses the Credential Manager to store the network and domain passwords, many other applications including Outlook, Windows Live Messenger and Remote Desktop use it to store their own credentials. The main difference between DPAPI and the Credential Manager is that DPAPI deals with encryption while the Credential store deals with both encryption and storage.

The Credential Manager stores the credentials in the credential store which is a binary array of credential structures (see appendix for the details of the structure). The credential store for Windows XP is stored in the directory:

```
<appdata>\Microsoft\Credentials\[user SID\Credentials
```

and encrypted using DPAPI. Depending on the credential type, the Credential Manager either stores the password in the clear or encrypted with DPAPI. Note that on Windows Vista and 7 the credential store is split in two files. One is located in the user local profile and the other one in the roaming profile. The four types of credentials stored by the credential stores are:

1. **Generic Password** Encrypt the password with DPAPI using the following string as DPAPI additional entropy "abe2869f-9b47-4cd9-a358-c22904dba7f7". Note that the string is not directly used but instead each character is cast in a signed short and multiplied by 4 to create the additional entropy. This type of storage is used by Live messenger and Internet Explorer > 6 for the HTTP authentication credentials.

2. **Domain Password** Store the the username and password in clear. Mainly used for Netbios passwords.

3. **Domain Certificate** Store only the hash of the certificates content.

4. **Domain Visible Password** Encrypt the password with DPAPI using the following string as DPAPI additional entropy: "82BD0E67-9FEA-4748-8672-D5EFE5B779B0". This type of storage is used for remote access and the .Net passport credentials.

3.3 Protected Storage

The ancestor of the credential store is the Protected Storage. The only reason why this deprecated API is worth mentioning is because Internet Explorer (version 4 - 6) uses it to store websites' credentials. The protected storage is simply a registry blob located under the key (similar to the the credential store):

```
HKEY_CURRENT_USER\Software\Microsoft\Protected
```

Storage system provider that contains an array of credentials. This blob is encrypted using the 3-DES algorithm. The 3DES key is stored in registry keys that the user usually cant access. However this is not an issue while performing an offline analysis.

4 File acquisition and analysis

The goal of the first OWADE module is to convert the hard-drive into a directory of files and databases of meta-data that will be used by later modules to perform higher-level analysis. File and meta-data extraction is the bread and butter of many forensic tools, so instead of reinventing the wheel, OWADE uses rock-solid GPL projects as a back-end for this module (*ddrescue [2]*, photorec [3]).

The only thing that OWADE does differently from most forensic tools is that it consolidates files extracted during previous analysis. This allows OWADE to deduce which files are unique to certain cases and quickly identify anomalies, such as a set of Windows core files that have been tampered with.

The files and meta-data extraction process is accomplished in three steps: first a faithful image of the disk is made, then the files are extracted from this image, and finally each file is individually analyzed.

To create an image of the disk, OWADE performs a sector by sector copy using the GNU program *ddrescue* [2] . The file extraction method depends whether the disk was fully formatted or not. If the disk is not fully formatted, has not been formatted at all, or has been ”quick” formatted, then a simple MBR restoration is performed, and all of the files are copied. The files stored in the recycle bin are recovered using the *undelete* utility.

The file carving technique [11] is used to recover the files that were fully deleted and are not present in the recycle bin any more. The file carving technique is also used to recover the file from fully formatted hard drives. Under the hood OWADE uses a modified version of Photorec to perform the file carving. Our modified version of Photorec is able to recover the DPAPI related files (DPAPI blobs, master key and Credhist files).

The biggest issue with using this file carving technique is that it does not recover the file paths. This is problematic because sometimes the path contains information that is not stored elsewhere. The most prominent example of this is the user SID (Windows User Security Identifier) needed to decrypt the DPAPI master keys.

5 Analyzing the Windows Registry

To parse the registry, OWADE relies on the hivex project [4] which provides an API to parse directly the hives in their binary form. OWADE extracts the following type of information from the registry:

- **Hardware Information**: By looking at the registry, OWADE is able to infer accurately what computer hardware was used. Besides being able to reconstruct accurately the computer hardware (CPU, graphic card, hard-drive, etc.), OWADE also extracts from the registry the list of all the USB devices, including USB drives, that were ever plugged in to the computer.
- **OS Information**: OWADE extracts the operating system version, its service pack level, which patches were applied, its licence key, and if it is a genuine version.
- **Configuration Information**: the configuration information extracted includes the computer's Netbios name, its time zone, and the path of the user profiles. Note that the computer's Netbios is used by Google to encrypt the GTalk credentials.
- **Software Information**: The list of every piece of software installed on the computer along with their respective versions is extracted from the registry. This list is used in the post mortem software vulnerability assessment as discussed below in section 10
- **Environment Details**: OWADE mainly reconstructs the user(s) environment variables to ensure that later analysis targets the right files.

The OWADE registry parser is also used by higher-level plugins (including those of Skype, Internet Explorer, and Gtalk) to extract specific registry keys needed to perform their tasks.

The next mandatory step is to recover the users' Windows credentials, as the SHA1 of the user password is used to derive the key used to decrypt the DPAPI master keys.

Windows stores the hash of the user passwords in the SAM (security account manager) [15]. Since Windows 2000, the SAM is encrypted with the SYSKEY utility [14] by default in an attempt to make offline attacks harder. However, by default all the information needed to reconstruct the SYSKEY key is stored in the registry hive `%SystemRoot%\System32\Config`. Accordingly, despite the SYSKEY encryption, unless the user has specifically decided to store the SYSKEY key on an external device, the extraction of the password hashes is easy.

The SAM stores two copies of the user password, each of which is hashed with a different hash function: the old LM hash function [12] and the newer NTLM one [13].

The LM hash function is known to exhibit two serious cryptographic weakness: First the passwords are hashed in upper-case which reduces the key space. Secondly a password is hashed into 2 different chunks of 7 characters instead of a single one which drastically reduce the search space from 46^{14} to 2×46^{7}. The LM hash is enabled by default on every Windows version except Windows 2008 and Windows 7.

OWADE uses the creddump project [6] to extract the hashes from the registry and John the Ripper [7] to crack them. When available, OWADE targets the LM Hash, as it is easier to crack, and then uses the NTLM hash to determine the case of the characters in the password.

Note that no salt is used while hashing the user passwords, regardless of the hash function used. Accordingly both hashes are vulnerable to rainbow table attacks that will speedup the cracking process. The LM hashes are the target of choice for rainbow tables–because of the small key space it is possible to have rainbow tables for every hash. We plan to release a module that uses the rainbow tables in the near future.

In the difficult case where the LM hash is not available and the user password cant be brute forced because it is too strong, a last option is to bypass the cracking step altogether and attempt to recover the SHA1 of the user password used to decrypt DPAPI keys from the Windows memory dump. While this approach is not going to recover any Windows user passwords, it allows OWADE to get the necessary piece of information (the SHA1 of the user password) needed to decrypt all the information protected by DPAPI and decrypt the Windows Credential store. This method suffers from the limitation that the computer must have been put in sleep mode at least once so that the hibernate file is present on the disk. The OWADE module that implements this method using Matthieu Suiche Moonsol memory toolkit [9] to decompress and linearize the hibernate file. As the Moonsol memory tookit is not free, this module is not included by default with the OWADE distribution. In theory, the open-source volatility framework [10] could be used to replace Moonsol but the current version of volatility is broken. Once it is fixed we plan to release a module based upon it.

7 WIFI PASSWORD AND USER GEO-LOCATION

The wifi access points information are stored in the registry in Windows XP under the key

```
HKLM\Software\Microsoft\WZCSVC
```

and in the file:

```
C:\ProgramData\Microsoft\Wlansvc\Profiles\Interfaces\
{GUID-IF}\{GUID}.xml
```

for Windows Vista and 7.

7.1 Recovering the access point passwords

Regardless of the Windows version and where the information is stored, the access point passwords are encrypted using DPAPI. What makes this encryption different from all others is that these passwords are encrypted with the system account, so every account on the system can connect to the access point. As the system account does not have a password, a random string is used to decrypt the DPAPI master key. This random string is stored as a LSASS secret and is acquired by OWADE at the same time as the Windows credentials.

7.2 Geo-locating the computer

Another interesting piece of information provided by the access point list is the MAC address of each access point that the computer was connected to and when the last connection occurred. This is useful in a couple of scenarios. For example, it allows OWADE to know if a corporate computer was connected to a rogue access point by comparing the list of MAC addresses stored with the list of company access point MAC addresses. It is also used to know where a laptop was at a given point by using one of the many geo-databases that allow geolocation of MAC address (i.e Google Map or Skyhook). **Note that all the APIs, as far as we know, limit the geo-query to the current region so it is impossible for anyone other than law enforcement agencies to be able to localize arbitrary mac addresses.**

8 Acquiring Browser Data

Browser data are split into two categories: user credentials and user history. The user history data is used to determine which websites the user visited; whereas his credentials allow OWADE to determine which online identity was used to access those sites. As visible in the table below (Table 1), each browser has its own way to store data, with SQLite being the most commonly used container. Encryption-wise every browser except Firefox uses DPAPI to ensure that the user data cant be read unless the user's Windows password is known.

Browser	Version	History location	Credential location	Protection
Chrome	All	SQLite files	SQLite	DPAPI
Firefox	< 3.5	file	file	FIPS-140
Firefox	>= 3.5	SQLite	SQLite	FIPS-140
Internet Explorer	6	file	Protected storage	Protected Storage
Internet Explorer	>= 7	file	Registry key	DPAP + extra entropy
Safari	All	.plist file	.plist file	DPAPI + extra entropy

Table 1: Overview of where the different browsers store their data and how they are protected.

8.1 Internet Explorer

What makes Internet Explorer the most difficult browser to deal with is its clever encryption scheme used to encrypt website credentials. Starting with Internet Explorer 7, each website's credentials are stored in a separate registry key that looks like this:

```
[sha1(url)] => DPAPI(entropy=url)
```

What makes the recovery difficult is that each of these registry keys is encrypted with DPAPI, and the website URL is used as the DPAPI entropy. As a result, we can only recover user credentials for URLs that we know/guess the user has visited.

OWADE tries to maximize the amount of online identities recovered by doing two things: First, OWADE constructs a list of potential URLs by aggregating all the URLs gathered via OWADEs various plugins including the browser history plugins, the registry analyzer, and the file data extractor. Secondly OWADE uses a brute force approach by comparing a list of known login URL SHA1 hashes against the registry keys' names. As explained in a previous section, recovering Internet Explorer 6 data is easier because it use the old protected storage API to store the user credentials 3.3.

Internet Explorer usually stores the user history in its profile. In this directory the index.dat contains a summary of all of the URLs visited. On any Windows version, the History directory path can be found by looking at the key:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\
Explorer\User Shell Folders\Cache
```

8.2 Firefox

Starting with Firefox 3.5, the user history is stored in various SQLite databases. Beside places.sqlite, which contains the list of all the URLs visited by the user, another interesting database is the formhistory.sqlite, which contains the history of every form field that has been entered except password fields. This file is very useful for reconstructing the user's activity, as it includes all the search query terms used and potentially the WiFi key, as these keys are usually not stored as a password field. For example a Linksys router will store its WPA PSK key in the field *wl1_wpa_psk*

Firefox uses a scheme inherited from Netscape Communicator and inspired by the FIPS 140-2 security policy to store its credentials. A file (signon.txt) for Firefox < 3.5 or a SQLite database stores the list of the users and passwords encrypted using a block cipher (3DES). The salt used to derive the key used to decrypt the passwords is stored in the file key3.db, which is a Berkeley DB. As recommended by the FIPS 140-2 specification, the key3.db file includes a test vector that consists of a known password (check-password) stored in clear and in encrypted form, making it easy to test if the derived key is correct. This test vector is used by OWADE to brute-force the user's master password if needed.

8.3 Chrome

Similarly to Firefox, Chrome stores its information in multiple SQLite databases, but unlike Firefox it uses DPAPI to encrypt the user password. Chrome has two interesting quirks: first, its SQLite file doesn't have a filename extension, and secondly its data is stored in the user local profile (`AppData\Local`) rather than in the standard roaming profile (`AppData\Roaming`).

The most interesting table to look at for the purpose of reconstructing the user's activity is the *Archived History* table that contains a consolidated view of all the URLs the user visited in the *url* table. The user cookies are stored in the *Cookies* database under the cookies table. The user login and password data are stored in the *Login Data* database in the table *login*. In the old version of Chrome, the credentials were stored in the *Web Data* database. The passwords are encrypted using DPAPI.

8.4 Safari

Because Safari comes from the Mac OS world, instead of using SQLite to store its data, it uses the famous Apple Property list file format (.plist). The property list format used to be a simple XML file, but in its most recent versions, the plist format is now a binary format so performing a deserialization is needed before processing. In Windows XP, Safari stores its data in the directory:

```
C:\Documents and Settings\%username%\Application Data
\Apple Computer\Preferences
```

In Windows Vista/7 Safari stores its data in the directory:

```
C:\Users\<username>\AppData\Roaming\Apple Computer\Preferences
```

The websites' credentials are stored in the `keychain.plist` file. The password fields are encoded in base64 and encrypted with DPAPI. A fixed string is used as additional DPAPI entropy. The user history is stored in the `History.plist` file and the bookmarks in the `Bookmarks.plist` file. Another interesting file is the `Form Value.plist` which contains the user's inputs that are not his credentials.

9 Acquiring Instant Messaging Data

The OWADE instant messaging plugin(s) are designed to recover the usernames and passwords stored by the most popular instant messaging software programs. In this section we describe how various messaging software programs encrypt their data. Where and how each messaging program stores its data is summarized in the table 2

Software	Version	Storage	Protection	extra-entropy source
Skype	All	File	DPAPI + custom algo	Username
Gtalk	All	Registry	DPAPI + custom algo	Login windows + nom netbios
MSN	5	Registry	"Base64"	
MSN	6	Registry	Protected Storage	Fixed string
MSN	7	Registry	DPAPI	random salt encrypted with DPAPI
MSN	Live	Registry	Credstore	
Pidgin	File	in clear		
Trillian	All	File	XOR	
9talk	All	File	XOR	
AMSN	All	File	DES	
PalTalk	All	File	Custom encryption	VolumeSerialNumber

Table 2: How the various instant messenger software are protected.

As visible on the table, except for the big ones (Skype, Gtalk and Microsoft Messengers), none of the instant messaging software programs rely on DPAPI to store the key used to encrypt their data. This malpractice makes the recovery process easy because the key used in their custom encryption schemes has to be either constructed from information stored on the disk or a fixed string.

9.1 Skype

Skype is the most difficult instant messaging software to deal with because the user credentials are not stored in a reversible form. Attempting to recover the Skype credentials involves performing the following steps:

1. Decrypt using DPAPI the secret token located in the registry under the key

   `HKCU\Software\Skype\ProtectedStorage\0`

2. Expand the secret token to an AES 256bits key using SHA-1 in the incremental counter mode.

3. Use this key to decrypt using AES the credential string stored in the

   `%APPDAPTA%\Skype\<account>\config.xml`

   in the `<Credentials2>....</Credentials2>` element. The first element of this string is the MD5 hash of the string `login + \nskyper\n + password`. Newer versions of Skype store the credentials in the `<Credential3>...</Credential3>` XML element.

4. Brute force the password using the other passwords harvested before using the patched version of John the ripper provided with OWADE.

Note that the Skype login name can be either recovered from the directory structure or recovered from the main.db file which is a SQLite file. The Skype username can be found in the Accounts table.

9.2 Google Gtalk

Google Gtalk uses DPAPI with a salt used as extra entropy. The salt value is derived from the windows username and the computer Netbios domain or Workgroup name. The GTalk DPAPI blob is stored under the registry key

`HKCU\Software\Google\Google Talk\Accounts\<login>\pw`

Google talk is the only messenger with Paltalk that ties the credential encryption to the machine information.

The salt derivation algorithm roughly works as follow:

1. The seed is initialized to 0xBA0DA71D

2. The seed is Xored with the Windows account name

3. The seed is Xored with the Netbios group/domain name

4. The seed is shuffled with the blob data.

9.3 Microsoft MSN/Live Messengers

The way the various versions of Microsoft messenger store the user credentials has changed drastically from one version to another until the first Live Messenger version. Since then Microsoft has settled down and decided to use the Windows credential store.

MSN Messenger 5. The credentials are stored in the registry and simply base-64 encoded.

MSN Messenger 6. The credentials are stored in protected storage and encrypted using DPAPI with a constant extra entropy string

MSN Messenger v7. Windows 7 uses a double layer of DPAPI protection: the random string used as DPAPI extra entropy to encrypt the credentials is itself encrypted with DPAPI using the fixed string `%GKP$^%^&LL(%^$^O&TR$^%^GV6;lxzd`. The DPAPI blob that contains this extra entropy is stored under the registry key:

```
HKCU\\Software\\Microsoft\\IdentityCRL\\Dynamic Salt\\Value
```

The credentials are stored under the registry key:

```
HKCU\\Software\\Microsoft\\IdentityCRL\\OfflineCreds
```

MSN Messenger Live. Every version of Microsoft Messenger Live uses the credential store to store the user credentials. They are automatically recovered by the OWADE credential manager module that extracts the entire contents of the credential store at once.

9.4 Other Instant Messengers

Here is how to recover the user data from some other messengers:

AMSN. stores its encrypted data in the file:

```
%USERPROFILE%\\amsn\\[username]\\config.xml
```

AMSN uses the DES encryption algorithm to encrypt the password. The DES key used to encrypt the password is simply the first eight characters of the user's login suffixed by the fixed string `dummykey (deskey = substr(login . dummykey, 8)`

9talk. stores its data in a file (`%USERPROFILE%\\neuftalk\\user.config`) and encrypts the password with a simple XOR 9.

Trillian. stores its data in the file (`%APPDATA%\\Trillian\\users\\global\\accounts.ini`) which look like this:

```
Account=<login>
Display Name=<display>
Password=ODc0OUY1QUIwOEI0RThBNgA=
```

The password is decrypted by doing a base64 decode followed by a XOR with the fixed key:

```
0x00F3, 0x0026, 0x0081, 0x00C4, 0x0039, 0x0086, 0x00DB, 0x0092,
0x0071, 0x00A3, 0x00B9, 0x00E6, 0x0053, 0x007A, 0x0095, 0x007C,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x00FF, 0x0000,
0x0000, 0x0080, 0x0000, 0x0000, 0x0000, 0x0080, 0x0080, 0x0000,
0x00FF, 0x0000, 0x0000, 0x0000, 0x0080, 0x0000, 0x0080, 0x0000,
0x0080, 0x0080, 0x0000, 0x0000, 0x0000, 0x0080, 0x00FF, 0x0000,
0x0080, 0x0000, 0x00FF, 0x0000, 0x0080, 0x0080, 0x0080, 0x0000,
0x0055, 0x006E, 0x0061, 0x0062, 0x006C, 0x0065, 0x0020, 0x0074,
0x006F, 0x0020, 0x0072, 0x0065, 0x0073, 0x006F, 0x006C, 0x0076,
0x0065, 0x0020, 0x0048, 0x0054, 0x0054, 0x0050, 0x0020, 0x0070,
0x0072, 0x006F, 0x0078, 0x0000
```

Pidgin. stores the credentials in the clear in the file `%APPDATA%\\.purple\\accounts.xml` This XML file looks like this:

```
<protocol></protocol>
<name></name>
<password></password>
```

Paltalk. Paltalk uses basically a simple substitution algorithm with a S-BOX that is created by combining the Paltalk username and the C: partition VolumeSerialNumber. The use of the VolumeSerialNumber is problematic for an offline decryption as it is not recoverable as far as we know. So the only option is to brute-force it.

The Paltalk decryption algorithm works as follows:

- An S-Box is created by interleaving the username and the VolumeSerialNumber. If the username is myusername and the 8 first hex values of the VolumeSerialNumber are 01234567 (in Hex) then the S-Box will be $m0y1u2s3e4r5n6a7me$ repeated three times: $m0y1u2s3e4r5n6a7mem0y1u2s3e4r5n6a7mem0y1u2s3e4r5n6a7me$
- The encrypted password looks like this in the registry $yyyz_1yyyz_2yyyz_3yyyz_4$. Each chunk $yyyz_n$ separated by a space is a password character. The last digit (z) is related to the timestamp and not used in the decryption.
- The password is decrypted by performing a character by character substitution in the following manner:

$$plainChar = yyy_i - asciiCode(S - BOX_n - i)$$

A last quirk to note is that the S-BOX is used starting from the end and not the beginning.

10 SOFTWARE ANALYSIS

OWADE extracts software information from the registry for two purposes:

- **Finding potential vulnerabilities**: OWADE is able to infer the list of potential vulnerability that affected the computer post-mortemBy correlating the list of software installed on the machine and their version with a known list of vulnerabilities (CVE). OWADE also compare the list of installed software with a list of known anti-virus and anti-malware to understand if the computer was protected.
- **Detecting Pirated software** OWADE extract the installed software version serials to compare it with a list of serial number that are known to be pirated.

11 CONCLUSION

In this paper we have presented OWADE (Offline Windows Analysis and Data Extraction www.owade.org), the first tool that is able to perform the advanced analysis required to extract the sensitive data stored by Windows. the browsers, and the instant messaging software. OWADE is written in Python, runs on Linux, and only uses GPL libraries and software (John the ripper, dd_rescue). Its cryptographic engine is the first to fully re-implement the Windows DPAPI without using any windows dll files and is able to decrypt the Windows Credential store.

OWADE decrypts and geolocates the historical WiFi data stored by Windows, providing a list of wifi points the computer has accessed (including the locations of the access points to within 500 feet) and when each point was last accessed. It can also recover all the logins and passwords stored in popular browsers (Internet Explorer, Firefox, Safari, and Chrome) and instant messaging software (Skype, MSN live, Gtalk, etc.). Finally, it can reconstruct the users online activity by reconstructing their browsing history from various sources: browsers, the Windows registry, and the Windows certificate store [1].

You can download OWADE from www.owade.org.

REFERENCES

[1] J. D. Clercq. Windows certificates. TechNet http://technet.microsoft.com/en-us/library/cc700805.aspx. 3, 22

[2] GNU. Ddrescue - data recovery tool. http://www.gnu.org/software/ddrescue/ddrescue.html. 9

[3] C. Grenier. Photorec, digital picture and file recovery. http://www.cgsecurity.org/wiki/PhotoRec. 9

[4] R. W. Jones. hivex - windows registry "hive" extraction library. http://libguestfs.org/hivex.3.html. 10

[5] Microsoft. Credential manager. MSDN http://msdn.microsoft.com/en-us/library/aa923650.aspx, 2010. 7

[6] moo...@gmail.com. creddump extracts credentials from windows registry hives. http://code.google.com/p/creddump. 11

[7] OpenWall. John the ripper password cracker. http://www.openwall.com/john/. 11

[8] J. M. Picod and E. Bursztein. Reversing dpapi and stealing windows secrets offline. Blackhat DC http://ly.tl/t6, 2010. 7

[9] M. Suishe. Moonsol memory toolkit. http://www.moonsols.com/windows-memory-toolkit/. 12

[10] V. System. The volatility framework: Volatile memory artifact extraction utility framework. https://www.volatilesystems.com/default/volatility. 12

[11] Wikipedia. File carving. http://en.wikipedia.org/wiki/File_carving. 9

[12] Wikipedia. The lm hash function. http://en.wikipedia.org/wiki/LM_hash. 11

[13] Wikipedia. The ntlm hash function. http://en.wikipedia.org/wiki/NTLM. 11

[14] Wikipedia. Syskey. http://en.wikipedia.org/wiki/SYSKEY. 11

[15] Wikipedia. System account manager. http://en.wikipedia.org/wiki/System_Account_Manager. 11