MANDIANT®

A FireEye® Company

THE BLACKBOX OF DPAPI

The Gift That Keeps on Giving

SECURITY CONSULTING

About the Speaker

- Bart Inglot (@BartInglot)
- Senior Consultant, Mandiant

- Incident Responder
- Rock Climber
- Globetrotter
  - 1 year in Brazil
  - 8 years in the UK
  - recently married and relocated to Singapore

Presentation Overview

- Introduction to DPAPI.
- 3 case studies:
  1) Interesting example of using DPAPI by Mandiant.
  2) Not-so-effective DPAPI use by an APT group.
  3) Replaying RDP session.
- Q&A.

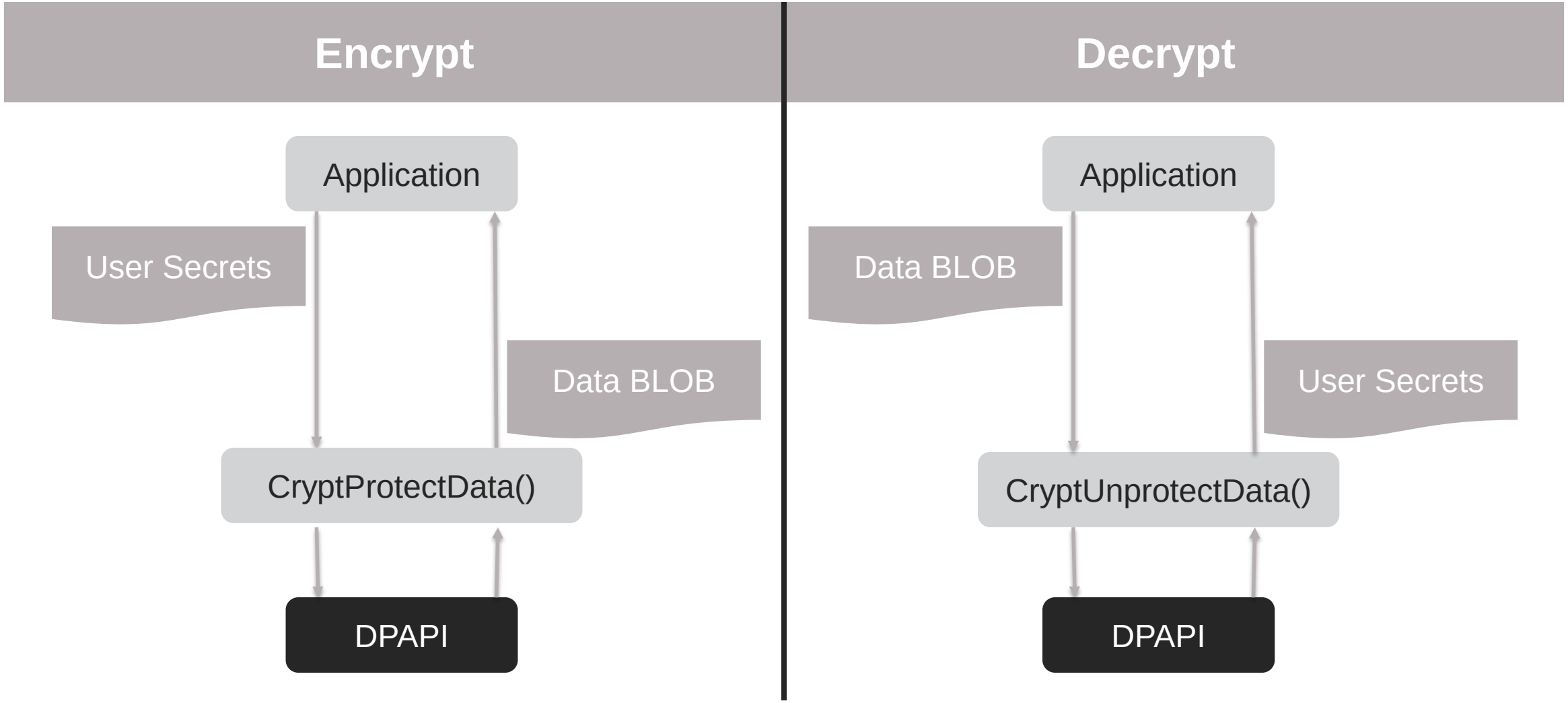
INTRODUCTION TO DPAPI

Peeking into the Blackbox

What is Data Protection API (DPAPI)?

- A special data protection interface, introduced in Windows 2000, aims to protect secrets on disk.
- Goals
  - Tie encryption key to a particular user or system, with an optional salt.
  - Use strong and standardised cryptographic algorithms (PBKDF2, AES, SHA, etc.).
  - To be transparent to the user.
- Key features
  - Simple to use: CryptProtectData() & CryptUnprotectData().
  - Technical implementation is complicated and not well documented by Microsoft.
  - Widely used as a cryptographic blackbox: Internet Explorer, Outlook, Skype, EFS, KeePass, etc.
- First analysed in 2003, a commercial recovery application available since 2005.
- Python library (DPAPIck) available since 2010, last commit March 2017.

How it works?



.NET Wrapper – ProtectedData Class

Inheritance Hierarchy

System.Object

System.Security.Cryptography.ProtectedData

Syntax

C# | C++ | F# | VB

```
public static byte[] Protect(
        byte[] userData,
        byte[] optionalEntropy,
        DataProtectionScope scope
)
```

Syntax

C# | C++ | F# | VB

```
public static byte[] Unprotect(
        byte[] encryptedData,
        byte[] optionalEntropy,
        DataProtectionScope scope
)
```

DataProtectionScope.CurrentUser = 0
The protected data is associated with the current user. Only threads running under the current user context can unprotect the data.

DataProtectionScope.LocalMachine = 1
The protected data is associated with the machine context. Any process running on the computer can unprotect data. This enumeration value is usually used in server-specific applications that run on a server where untrusted users are not allowed access.

.NET Wrapper – Example

```
using System;
using System.IO;
using System.Text;
using System.Security.Cryptography;

public class DataProtectionSample
{
        static void Main(string[] args)
        {
            string secret = "This is a super secret text";
            byte[] secret_array = Encoding.ASCII.GetBytes(secret);
            byte[] encrypted = ProtectedData.Protect(secret_array, null, DataProtectionScope.LocalMachine);

            BinaryWriter bw = new BinaryWriter(new FileStream("mydata.bin", FileMode.Create));
            bw.Write(encrypted);
            bw.Close();

            Console.WriteLine("Done.");
            Console.ReadKey();
    }
}
```

WinApi – CryptProtectData()

```
BOOL WINAPI CryptProtectData(
  _In_       DATA_BLOB  *pDataIn,                  ⟵ User data to be encrypted
  _In_opt_   LPCWSTR  szDataDescr,                 ⟵ Optional description
  _In_opt_   DATA_BLOB  *pOptionalEntropy,         ⟵ Optional entropy (salt)
  _Reserved_ PVOID  pvReserved,
  _In_opt_   CRYPTPROTECT_PROMPTSTRUCT  *pPromptStruct,  ⟵ Optional password prompt
  _In_       DWORD  dwFlags,                       ⟵ Flags (user/system/local machine scope)
  _Out_      DATA_BLOB  *pDataOut                  ⟵ Encrypted BLOB
);
```

https://msdn.microsoft.com/en-us/library/aa380261(vs.85).aspx

WinApi – CryptUnprotectData()

```
BOOL WINAPI CryptUnprotectData(
  _In_        DATA_BLOB  *pDataIn,                    ⟵ BLOB to be decrypted
  _Out_opt_   LPWSTR  *ppszDataDescr,                 ⟵ Optional description
  _In_opt_    DATA_BLOB  *pOptionalEntropy,           ⟵ Optional entropy (salt)
  _Reserved_  PVOID  pvReserved,
  _In_opt_    CRYPTPROTECT_PROMPTSTRUCT  *pPromptStruct,  ⟵ Optional password prompt
  _In_        DWORD  dwFlags,                         ⟵ Flags (user/system/local machine scope)
  _Out_       DATA_BLOB  *pDataOut                    ⟵ Plaintext secret
);
```

https://msdn.microsoft.com/en-us/library/aa380882(v=vs.85).aspx

DPAPI Blob: Full view of encrypted text file

Cipher text (154 bytes):

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   01 00 00 00 D0 8C 9D DF 01 15 D1 11 8C 7A 00 C0   ....ÐŒ.ß..Ñ.Œz.À
00000010   4F C2 97 EB 01 00 00 00 2D 54 43 AD A6 3E 23 4A   OÂ—ë....-TC.¦>#J
00000020   92 3F D4 33 6E 84 17 54 04 00 00 00 02 00 00 00   ’?Ô3n„.T........
00000030   00 00 03 66 00 00 C0 00 00 00 10 00 00 00 99 A1   ...f..À.......™¡
00000040   86 47 00 42 25 6C 94 5F 1B 2E 92 9A A9 46 00 00   †G.B%l”_..’š©F..
00000050   00 00 04 80 00 00 A0 00 00 00 10 00 00 00 4B F6   ...€.. ........Kö
00000060   E3 0B 7F 45 83 55 DB 67 3F A3 B3 7F 28 D6 10 00   ã..EƒUÛg?£³.(Ö..
00000070   00 00 74 71 99 5D 3E EC DE E1 FA F3 8B 66 85 AC   ..tq™]>ìÞáúó‹f…¬
00000080   5A 8D 14 00 00 00 2E FF A0 7F AA 92 D1 4A 31 9D   Z......ÿ .ª’ÑJ1.
00000090   FD 59 33 4A 70 AC 34 BE 03 05                     ýY3Jp¬4¾..
```

Plain text (11 bytes):

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   4C 69 74 74 6C 65 20 74 65 78 74                 Little text
```

DPAPI Blob: Partial view of encrypted “cmd.exe”

Cipher text (345,234 bytes):

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   01 00 00 00 D0 8C 9D DF 01 15 D1 11 8C 7A 00 C0   ....ÐŒ.ß..Ñ.Œz.À
00000010   4F C2 97 EB 01 00 00 00 2D 54 43 AD A6 3E 23 4A   OÂ—ë....-TC.¦>#J
00000020   92 3F D4 33 6E 84 17 54 04 00 00 00 02 00 00 00   ’?Ô3n„.T........
00000030   00 00 03 66 00 00 C0 00 00 00 10 00 00 00 E1 21   ...f..À.......á!
00000040   36 94 25 35 62 99 91 67 09 9A 40 E0 80 A8 00 00   6”%5b™‘g.š@à€¨..
00000050   00 00 04 80 00 00 A0 00 00 00 10 00 00 00 02 19   ...€.. .........
00000060   4F 13 C8 06 9F CB 52 12 27 B2 D9 E4 6E 3F 08 44   O.È.ŸËR.'²Ùän?.D
00000070   05 00 D4 A9 3B 8A DD CF 04 B8 B8 5A BC 46 8A 62   ..Ô©;ŠÝÏ.¸¸Z¼FŠb
00000080   18 1B 1E 38 00 7B 1C 7A BC B1 42 1F 29 96 A9 57   ...8.{.z¼±B.)–©W
00000090   78 5D 59 FE 33 15 CD AE 04 03 F8 27 10 A0 FF 7D   x]Yþ3.Í®..ø'. ÿ}
```

Plain text (345,088 bytes):

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00   MZ..........ÿÿ..
```

CORRUPTING EXFIL

Example of Legitimate Use of DPAPI

Background

- The client was notified they are compromised.
- Mandiant was engaged to perform Incident Response.
- Within first few days we discovered a legacy system (Windows 2000) with GBs of staged exfil.
- The files were encrypted RAR files.
- Action plan:
  - Buy time by corrupting the staged data and allow them to steal that.
  - Do not tip off the attackers.
- Key features:
  - Efficiently corrupt the files but make sure they're not recoverable.
  - Cover the tracks by faking the file timestamps (aka "timestomping").

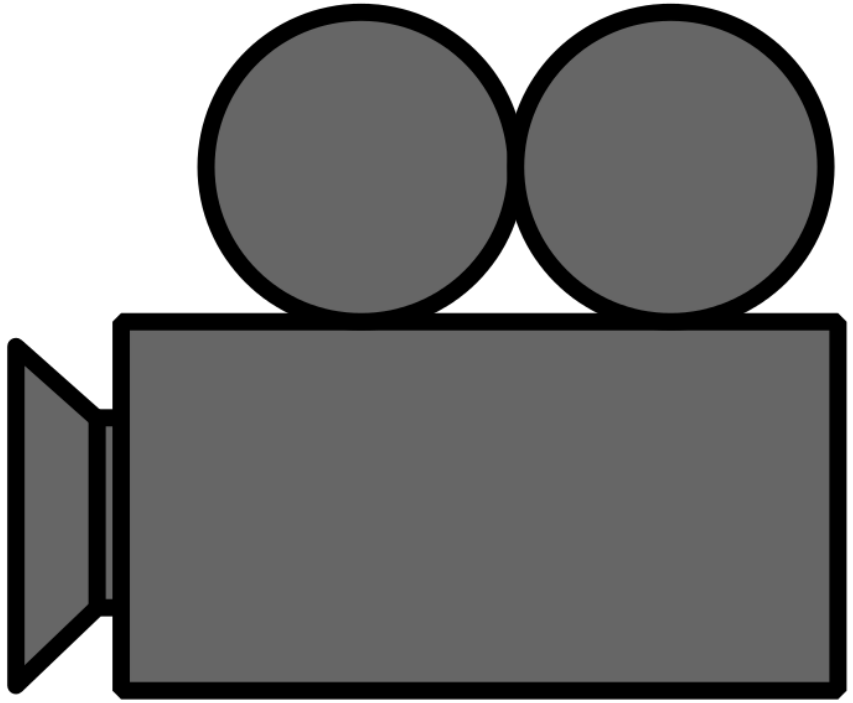
Challenges

- Efficiently corrupt the files but make sure they're not recoverable.
  - Encrypted RAR file with and without header.
  - Corner cases: very small / large files.
  - Double-XOR == plaintext.
- Cover the tracks by faking the file timestamps (aka "timestomping")
  - All 4 timestamps (MACB)?
- Bonus points: different file formats, reversible corruption, monitoring staging dirs (e.g. WMI trigger), decent logging (can be your enemy).
- Other archive formats: CAB, 7zip, Zip (inc. Office files).
- Treat differently? file creation VS file rename VS file copy
- The file has a lock? still writing or already stealing

Recipe

- Keep the secret.
  - Split parts of the encryption key between the application and the config.
  - DPAPI encrypt the config on 1st execution.
  - Initiate PRNG with the key.
  - Use PRNG for byte positions and XOR values.
- Monitor pre-defined list of folders for changes.
  - __InstanceCreationEvent (WMI).
  - System.IO.FileSystemEventHandler (.NET).
- Corrupt the file.
  - Keep trying to open the file (sleep).
  - XOR bytes: (buffer[pos] ^ rnd.Next()) + 1
  - Update Modification timestamp.

DEMO



PROTECTING MALWARE PAYLOAD

A Little Less Legitimate Use of DPAPI

Background

- Mandiant was engaged to perform Incident Response (APJ region).
- State-sponsored attacker (most likely APT17).
- We identified a legit Windows service (“FastUserSwitchingCompatibility”) that was re-purposed.
  - Service DLL: C:\windows\system32\adb.dll
  - Mysterious: C:\Windows\System32\adb.nls
- Submitted for analysis to the FLARE team and the report was surprisingly short:
  - Malware is a loader, likely for a HIGHNOON family.
  - It expects to load as a service and has a non-standard ServiceMain function.
  - It uses DPAPI to protect data and tie it to the system (and maybe user) that runs the malware.
  - “Attempting to decrypt this will fail on systems other than the one it was installed on, unless some research is done to try to obtain the necessary crypto material from the compromised system.”

Challenge

- Modular backdoor used by some APT groups with a rootkit capability.
- Installers – x86 & x64, require password as the argument.
- Drops a launcher (.DLL) and payload (usually .DAT), attempts to hijack an existing “netsvc” service.
- The payload is stored on disk in a DPAPI encrypted format.
- Capabilities:
  - Proxying network connections.
  - Concealing network connections.
  - Loading memory-resident DLL modules.

Action Plan

- Determine DPAPI encryption scope (user VS system) used to protect malware.
- Decode the encrypted payload offline.
- Create IOCs where possible:
  - Loader
    - Hijacked services.
    - Non-standard ServiceMain().
    - Static file analysis, etc.
  - Encrypted payload
    - DPAPI encrypted BLOB in unusual locations.
    - DPAPI optional description.
  - Plaintext payload

Recipe: DPAPI Encryption Scope

- "Programs running under the built-in system accounts, such as Windows services running as LocalSystem, cannot use DPAPI with user-specific keys."
- Load the BLOB with DPAPIck and inspect the flags.

```
[Dbg]>>> print(data_blob)
DPAPI BLOB
    version       = 1
    provider      = df9d8cd0-1501-11d1-8c7a-00c04fc297eb
    mkey          = e59699d3-2207-49aa-ac5d-cfa6b6138392
    flags         = 0x4
    descr         =
[Dbg]>>>
```

```
if (data_blob.flags & CRYPTPROTECT_LOCAL_MACHINE) >> 2 == False:
    print('[!] Data blobs that were encrypted in the User scope are not supported')
    sys.exit(1)
```

Recipe: Signaturing Encrypted BLOBs

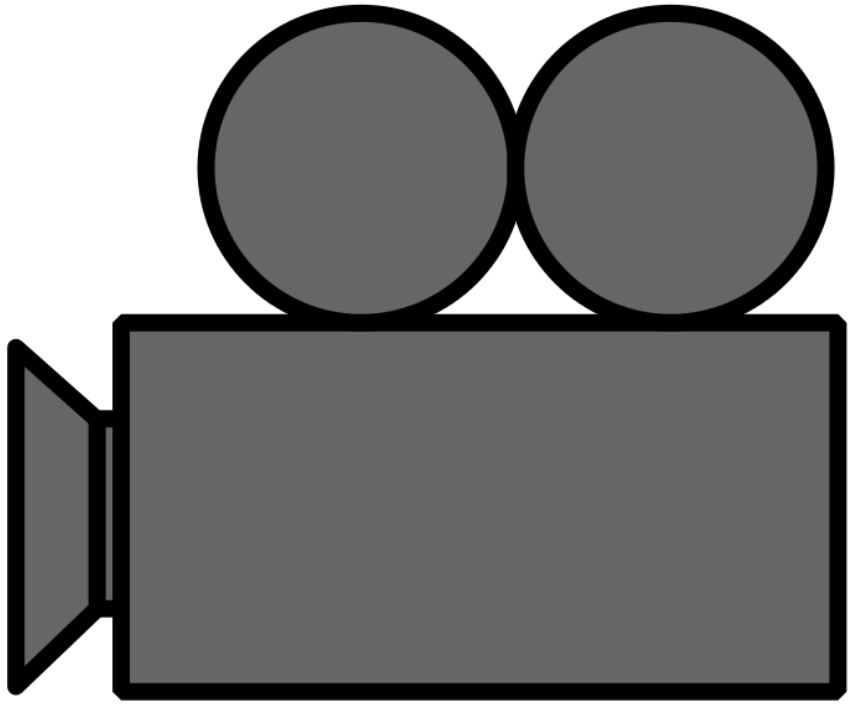
- DPAPI encrypted BLOB.
  - Trial & Error: Compared several payloads, used 24 first bytes.
- DPAPI Optional Description.

```
rule DPAPI_Desc_is_ABC {
        meta:
            created = "<REDACTED>"
            md5 = "<REDACTED>"
        strings:
            $desc = {65 00 66 00 67 00 00 00}      // hex values for “ABC”
        condition:
            uint32(0x2c) == 0x00000008 and $desc at 48
    }
```

Recipe: Offline DPAPI Decryption

- Necessary to decrypt system secrets:
  - Registry Hives – SYSTEM & SECURITY
  - System's Master Keys ("C:\Windows\System32\Microsoft\Protect*")
- Necessary to decrypt user secrets:
  - User SID
  - User's Master Keys ("%APPDATA%\Microsoft\Protect\{SID}*")
  - Logon password used to unlock the master key – "SHA1(UTF16LE(user_password))"
    - Alternatively SYSKEY, SECURITY
    - Or brute-force the password of the CREDHIST file
    - Or extract the SHA1 from a memory dump ("lsass.exe"), see "getcredentialsha1.py" from DPAPIck
- Optional for both: Entropy data.

DEMO



RDP KEYS EXTRACTION

Not So Secret Anymore

Background

- The attackers accessed a domain controller via Remote Desktop (RDP).
- There's little evidence on the host but the client has a full packet capture (PCAP).
- If we could peek into the RDP session, we'd find out what happened.
- Information available:
  - Hostname, keyboard layout, clipboard, file transfer, screen rendering, etc.
- The concept of RDP replay isn't new but until recently it wasn't widely available.
  - Thank you Context IS for "RDP-Replay", an open-source tool based on FreeRDP.
  - Great introduction into RPD replay at https://contextis.com/resources/blog/rdp-replay/.
- Two variants of encryption:
  - RC4 (All OS)
  - SSL (Vista+, preference)

Challenges

- Identifying locations of encryption keys.
- Extracting them offline (think “enterprise forensics”).
- Convert them to a format that we can use.
- Supporting both RC4 and SSL.
- Replay RDP traffic with RDP-Replay.
- Tackle the problem of Diffie-Hellman Key Exchange.

Recipe: Extracting RDP Keys on Active Systems (Vista+)

- Jailbreak by iSEC Partners
  - Launches “mmc.exe” with Certificates snap-in and hooks two Crypto API libraries.
- Mimikatz
  - Example with v2.1, built 6 May 2016.
  - Context IS suggested successful use with PsExec.
  - Break the private key out of the PFX file:
    - $ openssl pkcs12 -in file.pfx -nodes -out x509.pem
    - Use password: mimikatz
    - Get out the x509 private key.
  - If you want to view a x509 PEM private key:
    - $ openssl rsa -noout -in x509.pem –text
- Both require local admin privileges.

```
mimikatz # crypto::capi
 Local CryptoAPI patched

mimikatz # privilege::debug
 Privilege '20' OK

mimikatz # crypto::cng
 "KeyIso" service patched

mimikatz # crypto::certificates
 /systemstore:local_machine /store:"Remote
 Desktop" /export
  * System Store  : 'local_machine' (0x00020000)
  * Store:  […]
```

Recipe: Locations of RDP Keys

Pre-Vista

- RC4
  - LSA Secret named “L$HYDRAENCKEY_28ada6da-d622-11d1-9cb9-00c04fb16e75” (380 bytes, plaintext)

Vista+

- RC4
  - SYSTEM\CurrentControlSet\Control\Terminal Server\RCM\Secrets (DPAPI encrypted)
    - L$HYDRAENCKEY_52d1ad03-4565-44f3-8bfd-bbb0591f4b9d (380 bytes)
    - L$HYDRAENCKEY_28ada6da-d622-11d1-9cb9-00c04fb16e75 (1,340 bytes)
- SSL
  - One key in Windows certificates store and marked as non-exportable.
  - %ALLUSERSPROFILE%\Application Data\Microsoft\Crypto\RSA\MachineKeys
    - Files have embedded name, search for “TSSecKeySet1”

Recipe: Extracting RDP Keys Offline

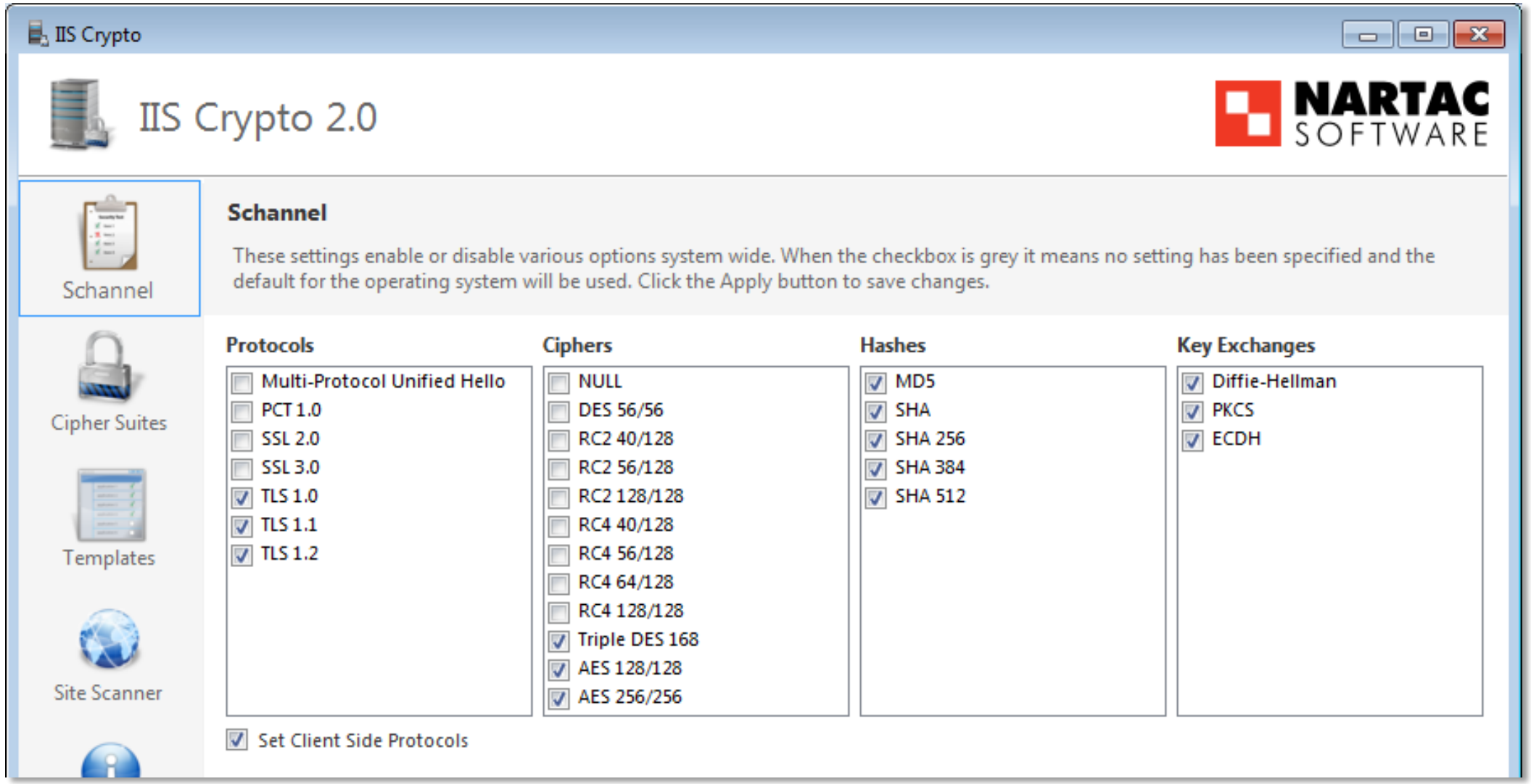
- Use-case: a forensic image or host agent.
- Pre-Vista
  - SYSTEM and SOFTWARE hives.
- Vista+
  - (registry hives as above)
  - System's Master Keys for DPAPIck.
    - C:\Windows\System32\Microsoft\Protect*
  - SSL key
    - %ALLUSERSPROFILE%\Application Data\Microsoft\Crypto\RSA\MachineKeys*
    - Extract the private key – find the first DPAPI header and read until the next header.
      - Magic bytes: \x01\x00\x00\x00\xD0\x8C\x9D\xDF …

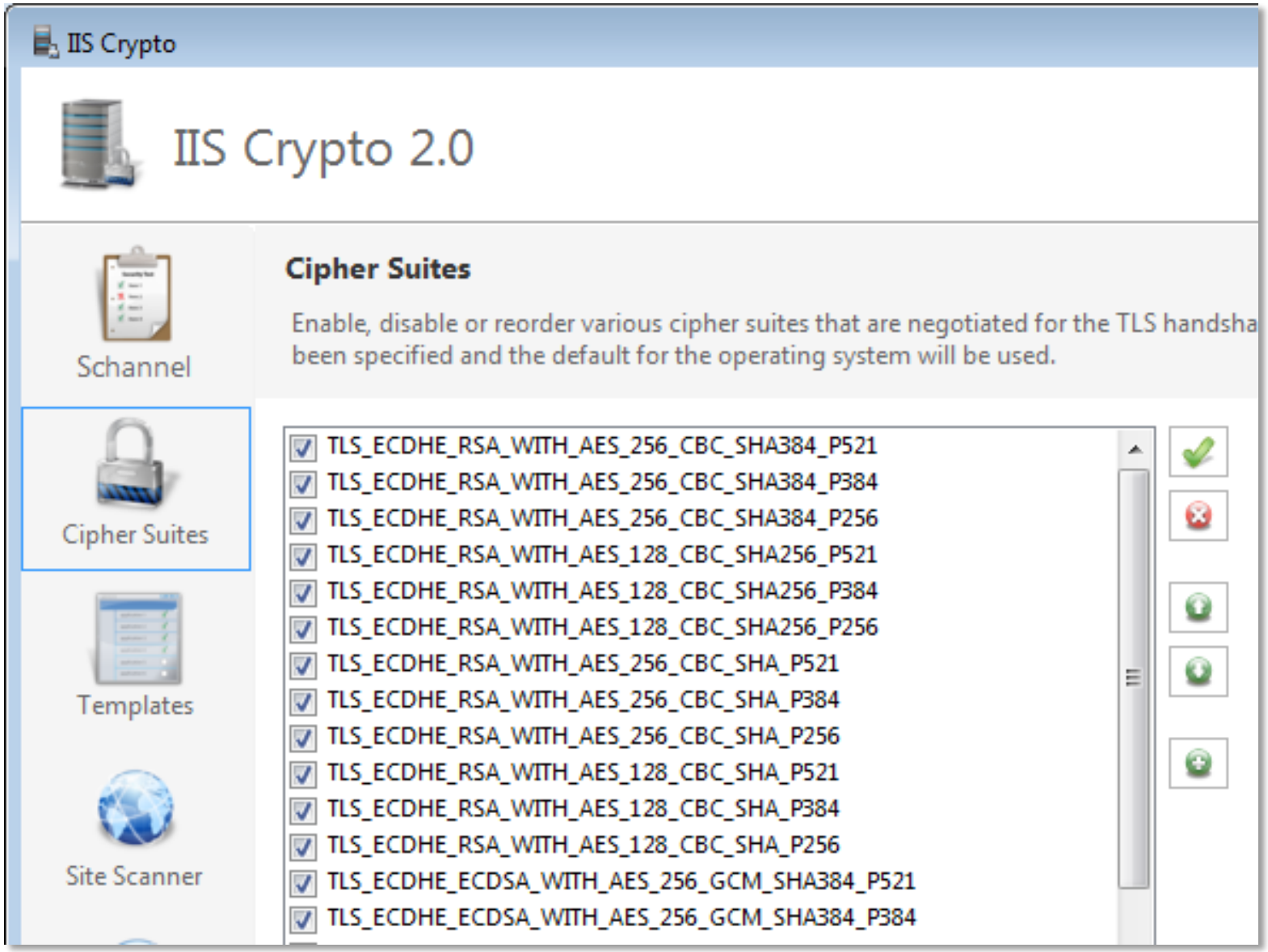
Recipe: Interpreting Exported RDP Keys

- The files appear to be in PRIVATEKEYBLOB format (aka PVK), however…
- OpenSSL and Dr Stephen Henson's "PVK to PEM conversion tool" failed
- The reason:
  - No PUBLICKEYSTRUC.
  - Values are zero-padded.
  - Extraneous zero-padded footer.
- After fixing the PVK structure, borrow CryptImportKey() from CryptoUnLocker.py by Kyrus.
  - Bug? The bytes_to_long() needs to be reversed.

Recipe: Getting Around DH Key Exchange

- The SSL protocol can use the Diffie-Hellman key exchange, which can provide Perfect Forward Secrecy.
  - "PFS is a way for two nodes to cryptographically agree upon a key that can not later be calculated, even if you get the private keys of both nodes."
- To check in WireShark, search for "Server Hello" message in the SSL handshake
  - Cipher Suite: TLS_**ECDHE**_RSA_WITH_AES_256_CBC_SHA (Default, Win7 SP1)
  - Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (FIPS compliant, Win7 SP1)
- Bad solution
  - Change the supported crypto – https://www.nartac.com/Products/IISCrypto
    - Disable ECDHE key exchange.
    - Reorder cipher suites to make RSA default.
- Terrible solution
  - Configure the RD Session Host server to use FIPS as the encryption level.

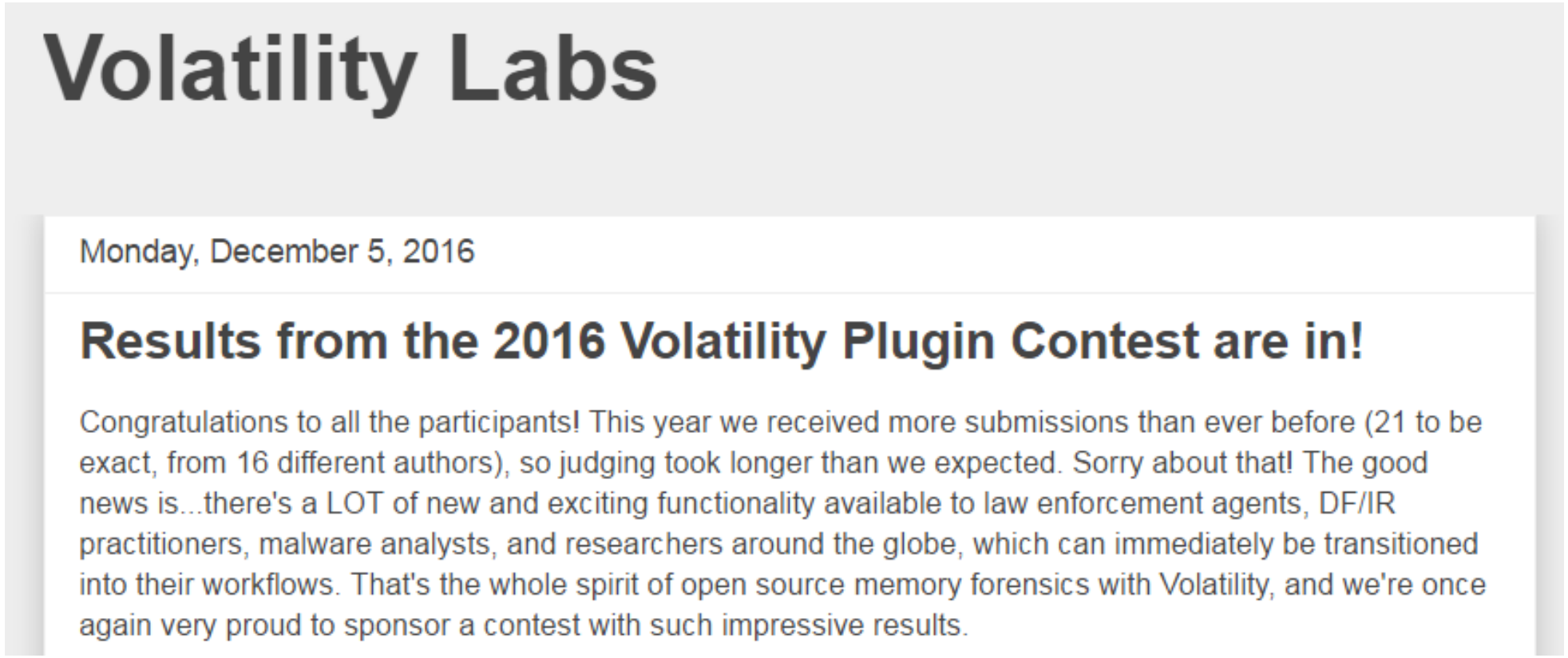




Recipe: Decrypting RDP Traffic

- Open the PCAP in Wireshark 2.0
- Right click a packet of the RDP session, and select “Decode As...”
  - TCP port: “3389”.
  - Current decoding: “SSL”.
- Right click a packet of the RDP session, and select “Protocol Preferences -> RSA keys list…”
- Add a key with the following properties:
  - IP Address: <RDP server's IP>
  - Port: 3389
  - Protocol: tpkt
  - Password: <empty>
- Right click a packet of the RDP session, and select “Follow -> SSL Stream”.

Solution: rdpkeys.py



Volatility Labs

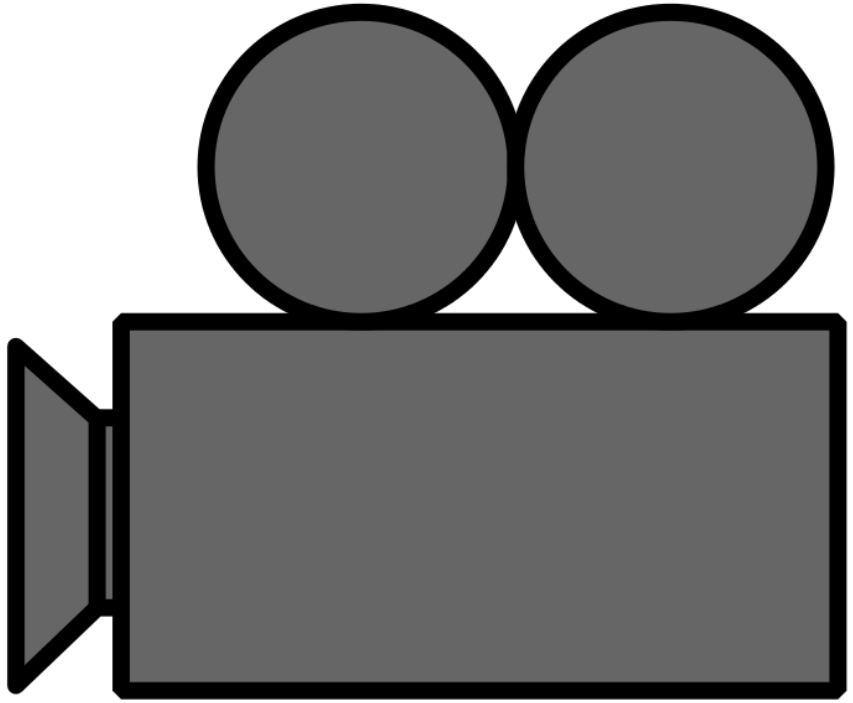
Monday, December 5, 2016

Results from the 2016 Volatility Plugin Contest are in!

Congratulations to all the participants! This year we received more submissions than ever before (21 to be exact, from 16 different authors), so judging took longer than we expected. Sorry about that! The good news is...there's a LOT of new and exciting functionality available to law enforcement agents, DF/IR practitioners, malware analysts, and researchers around the globe, which can immediately be transitioned into their workflows. That's the whole spirit of open source memory forensics with Volatility, and we're once again very proud to sponsor a contest with such impressive results.

- https://volatility-labs.blogspot.sg/2016/12/results-from-2016-volatility-plugin.html

DEMO



QUESTIONS?

bartosz.inglot@mandiant.com

THANK YOU