security-assessment.com

Defeating encryption through Reverse Engineering

Citrix NetScaler Encrypted EPA Bypass

Will Boucher 07/07/17

Contents

Introduction

"You know my methods. Apply them." *Sherlock Holmes, "The Sign of the Four"*

The Citrix NetScaler[1] Gateway VPN has the ability to check various conditions on a user device when it attempts to connect to a NetScaler Gateway. Based on the results of those conditions the NetScaler Gateway decides if a client is permitted to attempt a login, if the client is blocked or if the client is to be quarantined. Citrix calls this "Pre-Authentication Endpoint Analysis", or EPA. This is a problem when trying to connect to a NetScaler Gateway VPN without knowing the client-side checks required.

The NetScaler does this by running a client on the user's machine. On Windows this client is called ***nsepa.exe***. It connects to the NetScaler, receives a list of conditions, checks those conditions on the client device and then sends the NetScaler Gateway a result of pass or fail.

On the NetScaler this EPA communication can be configured to be in plaintext or be encrypted. When the NetScaler is configured without 'Client Security Encryption' the EPA check is trivial to bypass.

Previously there was no publicly available way to bypass EPA if 'Client Security Encryption' is enabled.

What follows is a walkthrough of what was discovered, through reverse engineering, of the nsepa.exe client, of the EPA encryption process and the resulting code that now enables encrypted Pre-Authentication Endpoint Analysis to be bypassed almost as trivially as plaintext EPA.

This was the author's first attempt at reverse engineering, and it was thoroughly enjoyed. As such, one aim of this paper is to enable a novice to reproduce the steps involved in the discovery of the bypass technique and then take on their own reverse engineering exercises.

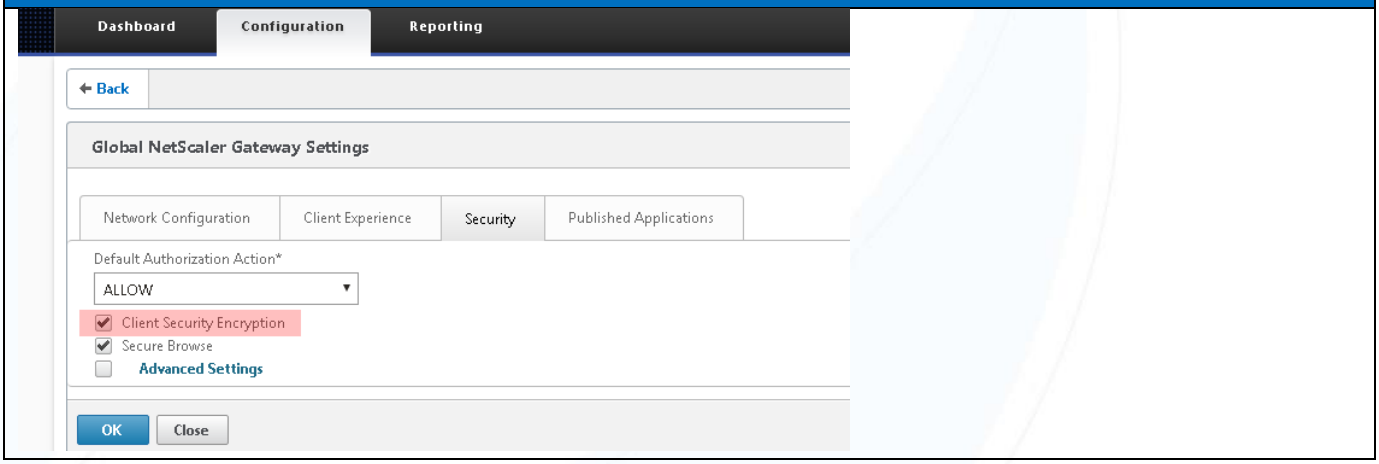
The Setup

"Data! Data! Data!" he cried impatiently. "I can't make bricks without clay." *Sherlock Holmes, "The Adventure of the copper beeches"*

The first step of setting up the environment is to get a Citrix NetScaler VPX installed and running. The version used in this case is NetScaler 10.5 Build 56.22. Next, set up the NetScaler Gateway. The process to set these up is beyond the scope of this document, but there are two key elements worth pointing out.

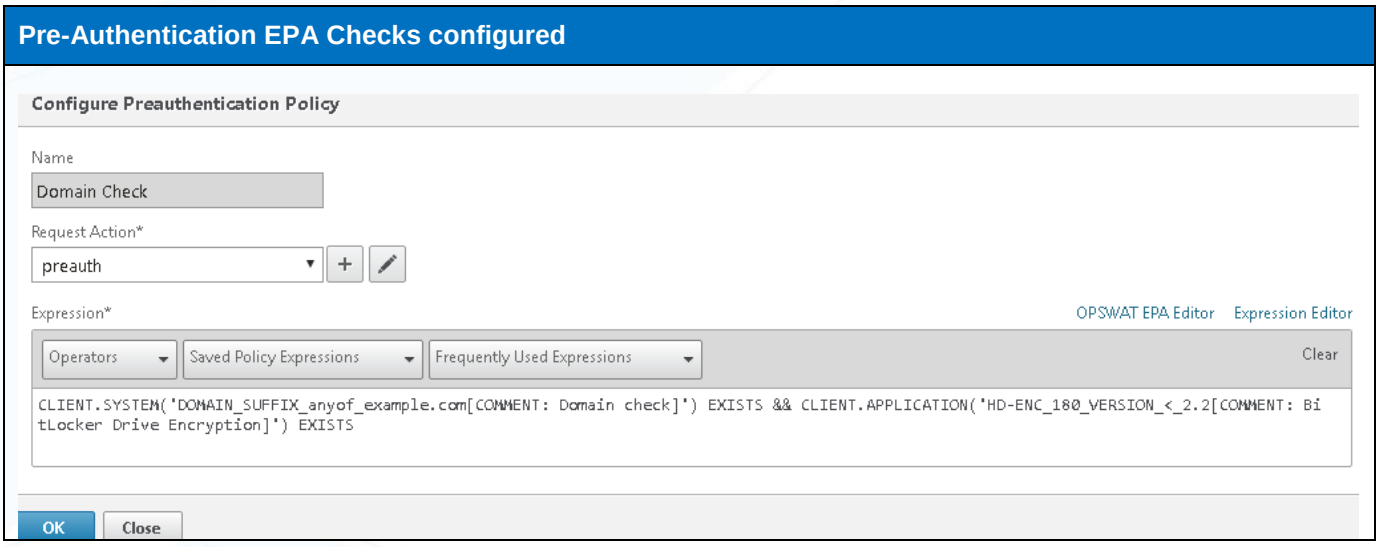
Once the NetScaler Gateway is up and running, to enable or disable the NetScaler Gateway encryption of the EPA traffic, in the configuration GUI, go to Configuration → NetScaler Gateway → Global Settings → Change Global Settings → Security and ensure the "Client Security Encryption" box is checked accordingly.

Client Security Encryption – Check



The other element is the Pre-Authentication profile and checks settings. While the entire process is beyond the scope of this document, it is worth illustrating the two Pre-Authentication EPA checks configured for this testing. These two EPA checks test if the connecting device is a member of the "example.com" domain, and if the connecting device is running full hard disk encryption using a version of BitLocker greater than 2.2

Pre-Authentication EPA Checks configured



A variety of tools were also used for this research, namely, Radare2[2] — The portable reversing framework — for reversing the nsepa.exe binary, WinDGB[3] for examining the execution of nsepa.exe and PortSwigger's Burp Suite[4] for intercepting the HTTP traffic between the web browser, the nsepa.exe client and NetScaler Gateway.

NetScaler Gateway and nsepa.exe have a chat

"It is, of course, a trifle, but there is nothing so important as trifles."

Sherlock Holmes, "The man with the twisted lip"

While watching the unencrypted conversation between the NetScaler and the nsepa.exe process, through an intercepting proxy (Burp Suite in this case), the conversation goes a little something like this:

Web browser requests / from the NetScaler Gateway

```
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: vpn.example.com
Connection: close
```

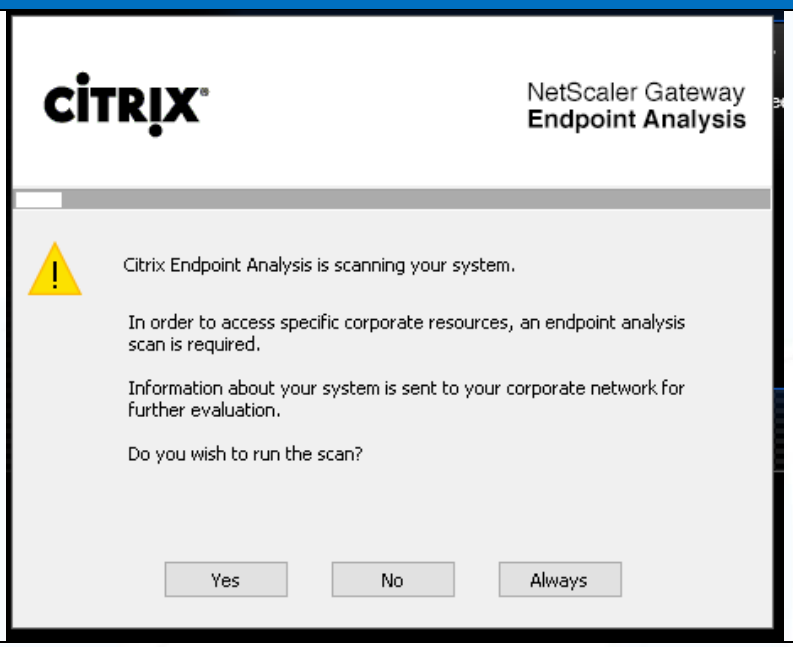
NetScaler Responds

```
HTTP/1.1 302 Object Moved
Location: /epa/epa.html
Set-Cookie: NSC_EPAC=73d7c702c284e7bba037c52e9037344c;Secure;Path=/
Set-Cookie: NSC_AAAC=xyz;Path=/;expires=Wednesday, 09-Nov-1999 23:12:40 GMT;Secure
Connection: close
Content-Length: 532
Cache-control: no-cache, no-store
Pragma: no-cache
Content-Type: text/html

<html><head><META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=UTF-8"><script
type="text/javascript" src="/vpn/resources.js"></script><script type="text/javascript"
language="javascript">var Resources = new ResourceManager("/vpn/resources/{lang}",
"REDIRECTION_BODY");</script></head><body><span id="This object may be found
"></span><a href="/epa/epa.html"><span id="here"></span></a><span id="Trailing phrase
after here"></span><script type="text/javascript"
language="javascript">Resources.Load();</script></body></html>
```

The web browser is issued a cookie named 'NSC_EPAC' and redirected to the /epa/epa.html page. If the NetScaler EPA plugin is installed, epa.html triggers the launch of nsepa.exe. The user is prompted to start the scan.

The nsepa.exe plugin



When the user clicks on the **Yes** button, nsepa.exe begins its conversation with the NetScaler Gateway — first asking for /epatype. The response from the NetScaler tells nsepa.exe if there are EPA checks to be done and if a device certificate is required for authentication:

NetScaler Response to GET /epatype

```
HTTP/1.1 200 OK
Content-Length: 22
Cache-control: no-cache, no-store
Pragma: no-cache
Content-Type: text/html

Epa:on;deviceCert:off;
```

Epa:on tells nsepa.exe to expect instructions for end point analysis while **deviceCert:off** rules out device certificate authentication. Based on Epa:on, nsepa.exe next requests /epaq to find out what it is supposed to check:

Requesting /epaq

```
GET /epaq HTTP/1.1
Cookie: NSC_EPAC=73d7c702c284e7bba037c52e9037344c
Date: 1496892587
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; AGEE 8.0;)
Host: vpn.example.com
Cache-Control: no-cache
Connection: close
```

NetScaler Responds

```
HTTP/1.1 200 OK
TunnelType:nocmp
Set-Cookie: NSC_ERRM=xyz;Path=/;expires=Wednesday, 09-Nov-1999 23:12:40 GMT
CSEC: app_0_HD-ENC_180_VERSION_<_2.2;sys_0_DOMAIN_SUFFIX_anyof_example.com;
Content-Length: 0
Cache-control: no-cache, no-store
Pragma: no-cache
Content-Type: text/html
```

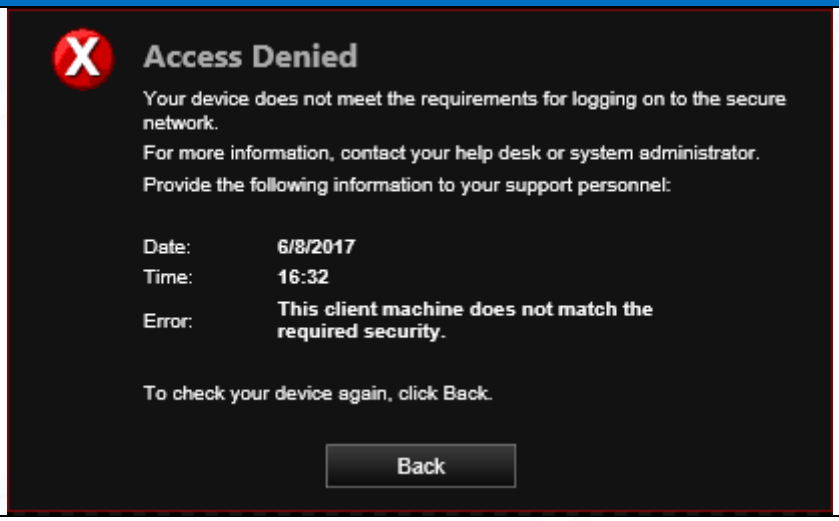
The NetScaler's response indicates, in the `CSEC:` header, that nsepa.exe should check for Hard Drive encryption with BitLocker version greater than 2.2 (app_0_HD-ENC_180_VERSION_<_2.2;) and that the device attempting to connect should be a member of the example.com domain (sys_0_DOMAIN_SUFFIX_anyof_example.com;). The nsepa.exe dutifully checks the connecting device to see if it meets the requirements, in this case no, and sends the result to the NetScaler Gateway.

Nsepa.exe sends EPA result

```
GET /epas HTTP/1.1
Cookie: NSC_EPAC=73d7c702c284e7bba037c52e9037344c
CSEC: 33
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; AGEE 8.0;)
Host: vpn.example.com
Cache-Control: no-cache
Connection: close
```

The CSEC header in the nsepa.exe response is a 33. Anything other than a zero is a fail — the device did not meet the requirements. The NetScaler Gateway revels in the failure.

NetScaler rejects the failure:



When 'Client Security Encryption' is disabled, bypassing the EPA check is as trivial as intercepting the response from nsepa.exe and replacing the '33' message with a '00' message to indicate that the EPA was a success and then the NetScaler will display the Login Page, no sweat.

Things do, however, get a little more complicated when the NetScaler has 'Client Security Encryption' enabled. The NetScaler Gateway and nsepa.exe decide to be a little more covert in their dealings.

NetScaler Gateway and nsepa.exe have a secret chat

"It is obviously an attempt to convey secret information." *Sherlock Holmes, "The Valley of Fear"*

Although the conversation flows pretty much the same when "Client Security Encryption" is enabled, there are a few exceptions, which are illustrated here. The CSEC headers that contain the EPA checks and the EPA responses are now base64 encoded blobs of encrypted information. Here are some relevant examples:

When nsepa.exe requests /epaq, the NetScaler Gateway response has a base64 encoded value for CSEC

Requesting /epaq

```
GET /epaq HTTP/1.1
Cookie: NSC_EPAC=a3f5bf5117387 6cf6bc789c8e16allda
Date: 1496812014
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; AGEE 8.0;)
Host: vpn.example.com
Cache-Control: no-cache
Connection: close
```

NetScaler Responds

```
HTTP/1.1 200 OK
TunnelType:nocmp
Set-Cookie: NSC_EPRM=xyz;Path=/;expires=Wednesday, 09-Nov-1999 23:12:40 GMT
Encode:Yes
CSEC: SXw0QeSnD7Gyr30YRzMPXCphDq9TLGhFkai6bWQuVHBkie0ZYp6AfGhtHniZQR8F
Content-Length: 0
Cache-control: no-cache, no-store
Pragma: no-cache
Content-Type: text/html
```

And again when nsepa.exe responds to the NetScaler Gateway the CSEC values are base64 encoded.

Requesting /epas

```
GET /epas HTTP/1.1
Cookie: NSC_EPAC=a3f5bf5117387 6cf6bc789c8e16allda
CSEC: x8qAtQG1+GWr4Q25E4Uc0w==
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; AGEE 8.0;)
Host: vpn.example.com
Cache-Control: no-cache
Connection: close
```

NetScaler Responds

```
HTTP/1.1 200 OK
Set-Cookie: NSC_EPAC=xyz;Path=/;expires=Wednesday, 09-Nov-1999 23:12:40 GMT;Secure
Content-Length: 24
Cache-control: no-cache, no-store
Pragma: no-cache
Content-Type: text/html

vVj77KIu0u3N0uYD0+MK0Q==
```

Decoding the Base64, as expected, is no help at all, it results in nothing apparently useful.

Base64 Decoded

```
0   49 7c 34 41 e4 a7 0f b1   b2 af 7d 18 47 33 0f 5c  |4Aä§□±²¯}□G3□\
1   2a 61 0e af 53 2c 68 45   91 a8 ba 6d 64 2e 54 70  *a□¯S,hE□¨°md.Tp
2   64 89 ed 19 62 9e 80 7c   68 6d 1e 78 99 41 1f 05  d□í□b□□|hm□x□A□□
```

Looking through the conversation between nsexp.exe and the NetScaler Gateway there is no obvious key exchange. Two working hypotheses formed at this point:

The first possibility is the key is hardcoded. If it is — it should be discoverable in the nsepa.exe client and the communication can be decrypted. Wouldn't that be too easy?

The second, more interesting, hypothesis stems from a curious header passed during the exchange between the client and gateway. There is a non-standard header sent during the request for /epaq by nsepa.exe. A "Date" header that contains, what appears to be, the current epoch time of the client system.

Requesting /epaq

```
GET /epaq HTTP/1.1
Cookie: NSC_EPAC=ec4826dea596be9b0cd540b8e1aed44d
Date: 1496885892
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; AGEE 8.0;)
Host: vpn.example.com
Cache-Control: no-cache
Connection: close
```

If this is used, in part or in whole, to generate a key known by both nsepa.exe and the NetScaler Gateway, reverse engineering the nsepa.exe process should reveal how the key is generated and allow for reproduction of the key — using the same method. This would enable the decryption of the CSEC information and allow the requirements for the Endpoint Analysis to be seen. And enable responding with a false success indicator to bypass the EPA checks.

What goes on under the hood

"In solving a problem of this sort, the grand thing is to be able to reason backward." *Sherlock Holmes, "A Study in Scarlet"*

The first step in figuring out which hypothesis is correct (if any) is to get into the actual code. This requires reversing from binary. Radare2 is the tool chosen for this. The approach: find where the decryption occurs and work backward from there to see if the key creation can be understood.

First, find where the decryption happens. Search the binary for a function with 'decrypt' in the name using radare2's `rabin.exe`:

```
.\rabin2.exe -s .\nsepa.exe | select-string -pattern decrypt
```

Which results in:

```
vaddr=0x00445024 paddr=0x00043e24 ord=010 fwd= sz=0 bind=NONE type=FUNC
name=imp.ADVAPI32.dll_CryptDecrypt
```

This reveals two important pieces of information, the first being that there is only one decrypt function used in the nsepa.exe program, and the second is that it's relying on ADVAPI32.dll for encryption/decryption. The next step is to find out where CryptDecrypt[5] from ADVAPI is being called.

Load nsepa.exe into radare2 using:

```
.\radare2.exe .\nsepa.exe
```

Once loaded, radare2 needs to be instructed to do a full analysis of the binary, to do this execute:

```
aaaaa
```

When it's completed, find where CryptDecrypt is being called by using the `axt` command, which finds cross references to functions:

```
axt: 0x00445024
```

Which returns:

```
call 0x401272 call dword sym.imp.ADVAPI32.dll_CryptDecrypt in
sub.ADVAPI32.dll_CryptImportKey_160
```

That tells us that CryptDecrypt is being called once from address `0x401272`. The next step is to have a look at the reversed code around that address:

Code calling CryptDecrypt

```
0x00401260      8d45dc          lea eax, dword [ebp - local_24h]
0x00401263      50              push eax
0x00401264      8b4de0          mov ecx, dword [ebp - local_20h]
0x00401267      51              push ecx
0x00401268      6a00            push 0
0x0040126a      6a01            push 1
0x0040126c      6a00            push 0
0x0040126e      8b55d8          mov edx, dword [ebp - local_28h]
0x00401271      52              push edx
0x00401272      ff1524504400    call dword [sym.imp.ADVAPI32.dll_CryptDecrypt] ; sym.imp.ADVAPI32.dll_CryptDecrypt
0x00401278      85c0            test eax, eax
0x0040127a      74da            je 0x401256
0x0040127c      57              push edi
0x0040127d      e87e6f0100      call fcn.00418200
0x00401282      83c404          add esp, 4
0x00401285      8b45e0          mov eax, dword [ebp - local_20h]
0x00401288      8b4ddc          mov ecx, dword [ebp - local_24h]
0x0040128b      c6040100        mov byte [ecx + eax], 0
0x0040128f      8b55e0          mov edx, dword [ebp - local_20h]
0x00401292      8917            mov dword [edi], edx
0x00401294      8b45dc          mov eax, dword [ebp - local_24h]
0x00401297      8903            mov dword [ebx], eax
0x00401299      eb08            jmp 0x4012a3
```

The second important piece of information obtained from the `axt` command above now comes into play. Knowing that the CryptDecrypt function is part of ADVAPI32.dll — documentation for that function can be found from Microsoft.

The MSDN entry for CryptDecrypt explains what the various values being set are, and more importantly that before CryptDecrypt is called CryptImportKey[6] needs to be called to setup the key, the handle obtained needs to be passed to CryptDecrypt. Repeat the `rabin.exe` command to find CryptImportKey:

```
.\rabin2.exe -s .\nsepa.exe | select-string -pattern importkey
```

```
vaddr=0x0044501c paddr=0x00043e1c ord=008 fwd= sz=0 bind=NONE type=FUNC
name=imp.ADVAPI32.dll_CryptImportKey
```

Then in radare2

`axt: 0x0044501c` shows several locations for CryptImportKey, this one looks most promising

```
call 0x401215 call dword sym.imp.ADVAPI32.dll_CryptImportKey in
sub.ADVAPI32.dll_CryptImportKey_160
```

The code around the CryptImportKey call shows:

Code calling sym.imp.ADVAPI32.dll_ImportKey

```
0x004011fc      51              push ecx
0x004011fd      6a00            push 0
0x004011ff      6a00            push 0
0x00401201      8b1594b24500    mov edx, dword [0x45b294]   ; [0x45b294:4]=0
0x00401207      52              push edx
0x00401208      a1a0b24500      mov eax, dword [0x45b2a0]   ; [0x45b2a0:4]=0
0x0040120d      50              push eax
0x0040120e      8b0d98b24500    mov ecx, dword [0x45b298]   ; [0x45b298:4]=0
0x00401214      51              push ecx
0x00401215      ff151c504400    call dword [sym.imp.ADVAPI32.dll_CryptImportKey] ; "*v." @ 0x44501c
0x0040121b      85c0            test eax, eax
0x0040121d      751b            jne 0x40123a
0x0040121f      ff15c8534400    call dword [sym.imp.KERNEL32.dll_GetLastError] ; sym.imp.KERNEL32.dll_GetLastError
0x00401225      8bf0            mov esi, eax
0x00401227      8975e4          mov dword [ebp - local_1ch], esi
0x0040122a      56              push esi
0x0040122b      68ec574400      push str.The_cryptimportkey_failed__d._n ; str.The_cryptimportkey_failed__d._n ; "The cryptimpo
0x00401230      e8bb710100      call fcn.004183f0
0x00401235      83c408          add esp, 8
0x00401238      eb69            jmp 0x4012a3
```

Browsing the code around these two calls reveals there is also a function called CryptSetKeyParam[7] that is called between CryptImportKey and CryptDecrypt:

Code calling sym.imp.ADVAPI32.dll_CryptSetKeyParam

```
0x0040123a      a1a4b24500      mov eax, dword [0x45b2a4]   ; [0x45b2a4:4]=0
0x0040123f      85c0            test eax, eax
0x00401241      741d            je 0x401260
0x00401243      6a00            push 0
0x00401245      50              push eax
0x00401246      6a01            push 1
0x00401248      8b55d8          mov edx, dword [ebp - local_28h]
0x0040124b      52              push edx
0x0040124c      ff1520504400    call dword [sym.imp.ADVAPI32.dll_CryptSetKeyParam] ; sym.imp.ADVAPI32.dll_CryptSetKeyParam
0x00401252      85c0            test eax, eax
0x00401254      750a            jne 0x401260
; JMP XREF from 0x0040127a (sub.ADVAPI32.dll_CryptImportKey_160)
0x00401256      ff15c8534400    call dword [sym.imp.KERNEL32.dll_GetLastError] ; sym.imp.KERNEL32.dll_GetLastError
0x0040125c      8bf0            mov esi, eax
0x0040125e      eb40            jmp 0x4012a0
```

MSDN states that before any of this can happen a context has to be set up with CryptAcquireContext[8], which, using the same methods as about with `rabin.exe` and `axt` is found at `0x00401586`:

Code calling sym.imp.ADVAPI32.dll_CryptAcquireContext

```
0x00401530      55              push ebp
0x00401531      8bec            mov ebp, esp
0x00401533      6afe            push -2
0x00401535      6838274500      push 0x452738
0x0040153a      6850cd4200      push 0x42cd50
0x0040153f      64a100000000    mov eax, dword fs:[0]
0x00401545      50              push eax
0x00401546      83ec28          sub esp, 0x28                ; '('
0x00401549      a1389e4500      mov eax, dword [0x459e38]    ; [0x459e38:4]=0xbb40e64e
0x0040154e      3145f8          xor dword [ebp - 8], eax
0x00401551      33c5            xor eax, ebp
0x00401553      8945e4          mov dword [ebp - local_1ch], eax
0x00401556      53              push ebx
0x00401557      56              push esi
0x00401558      57              push edi
0x00401559      50              push eax
0x0040155a      8d45f0          lea eax, dword [ebp - local_10h]
0x0040155d      64a300000000    mov dword fs:[0], eax
0x00401563      8965e8          mov dword [ebp - local_18h], esp
0x00401566      8b4508          mov eax, dword [ebp + arg_8h] ; [0x8:4]=4
0x00401569      8945c8          mov dword [ebp - local_38h], eax
0x0040156c      8b5d10          mov ebx, dword [ebp + arg_10h] ; [0x10:4]=184
0x0040156f      c745fc000000.   mov dword [ebp - local_4h], 0
0x00401576      68000000f0      push 0xf0000000
0x0040157b      6a18            push 0x18
0x0040157d      6a00            push 0
0x0040157f      6a00            push 0
0x00401581      6898b24500      push 0x45b298
0x00401586      8b3560504400    mov esi, dword [sym.imp.ADVAPI32.dll_CryptAcquireContextA] ; [0x445060:4]=0x5769c
                                reloc.ADVAPI32.dll_CryptAcquireContextA_156 LEA sym.imp.ADVAPI32.dll_
                                CryptAcquireContextA ; sym.imp.ADVAPI32.dll_CryptAcquireContextA
0x0040158c      ffd6            call esi
0x0040158e      85c0            test eax, eax
0x00401590      7569            jne 0x4015fb
```

Reading the documentation about CryptAquireContext there is an invaluable gem of information at address `0x0040157b`. This is where dwProvType for this function is set. MSDN documentation, and wincrypt.h[9], tell us that the cryptographic provider being set up is PROV_RSA_AES, defined as 0x18 or 24. The flavor of crypto being used is now known — AES. And because it's AES, three things are needed A key, an initialization vector and data to decrypt.

Knowing that the cryptography being used is AES, looking again at CryptSetKeyParam, it becomes clear that this is setting the Initialization Vector for the decryption process. At line `0x00401246` we can see dwParam being set to 1, which the MSDN documentation, and wincrypt.h shows is KP_IV, or 0x00000001. This is the function where the Initialization vector will be defined.

It looks like that's the complete key import process and decryption. What isn't seen, so far, is where the key comes from. Since nsepa.exe uses ADVAPI32 for encryption and key management, maybe there are other interesting functions that need examination. Using `rabin2.exe` searching for ADVAPI32 function calls is simple.

Searching for ADVAPI32 functions

```
.\rabin2.exe -s .\nsepa.exe | select-string -pattern ADVAPI32
vaddr=0x00445000 paddr=0x00043e00 ord=001 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptCreateHash
vaddr=0x00445004 paddr=0x00043e04 ord=002 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptHashData
vaddr=0x00445008 paddr=0x00043e08 ord=003 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegQueryValueA
vaddr=0x0044500c paddr=0x00043e0c ord=004 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegEnumKeyA
vaddr=0x00445010 paddr=0x00043e10 ord=005 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptGetHashParam
vaddr=0x00445014 paddr=0x00043e14 ord=006 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptReleaseContext
vaddr=0x00445018 paddr=0x00043e18 ord=007 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptDestroyHash
vaddr=0x0044501c paddr=0x00043e1c ord=008 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptImportKey
vaddr=0x00445020 paddr=0x00043e20 ord=009 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptSetKeyParam
vaddr=0x00445024 paddr=0x00043e24 ord=010 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptDecrypt
vaddr=0x00445028 paddr=0x00043e28 ord=011 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptDestroyKey
vaddr=0x0044502c paddr=0x00043e2c ord=012 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptExportKey
vaddr=0x00445030 paddr=0x00043e30 ord=013 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptEncrypt
vaddr=0x00445034 paddr=0x00043e34 ord=014 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegCloseKey
vaddr=0x00445038 paddr=0x00043e38 ord=015 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegEnumKeyExA
vaddr=0x0044503c paddr=0x00043e3c ord=016 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegOpenKeyExA
vaddr=0x00445040 paddr=0x00043e40 ord=017 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CloseServiceHandle
vaddr=0x00445044 paddr=0x00043e44 ord=018 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_QueryServiceStatus
vaddr=0x00445048 paddr=0x00043e48 ord=019 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_OpenServiceA
vaddr=0x0044504c paddr=0x00043e4c ord=020 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_OpenSCManagerA
vaddr=0x00445050 paddr=0x00043e50 ord=021 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegQueryValueExA
vaddr=0x00445054 paddr=0x00043e54 ord=022 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegCreateKeyExA
vaddr=0x00445058 paddr=0x00043e58 ord=023 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_IsWellKnownSid
vaddr=0x0044505c paddr=0x00043e5c ord=024 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_GetTokenInformation
vaddr=0x00445060 paddr=0x00043e60 ord=025 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_CryptAcquireContextA
vaddr=0x00445064 paddr=0x00043e64 ord=026 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegOpenKeyA
vaddr=0x00445068 paddr=0x00043e68 ord=027 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegQueryInfoKeyA
vaddr=0x0044506c paddr=0x00043e6c ord=028 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegSetValueExA
vaddr=0x00445070 paddr=0x00043e70 ord=029 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegDeleteValueA
vaddr=0x00445074 paddr=0x00043e74 ord=030 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_RegDeleteKeyA
vaddr=0x00445078 paddr=0x00043e78 ord=031 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_GetUserNameA
vaddr=0x0044507c paddr=0x00043e7c ord=032 fwd= sz=0 bind=NONE type=FUNC name=imp.ADVAPI32.dll_OpenProcessToken
```

CryptCreateHash[10], CryptHashData[11] and CryptGetHashParam[12] look like they have potential.

Digging into the MSDN documentation the CryptCreateHash function puts data in place in memory for hashing and specifies the parameters of the hash.

Code calling sym.imp.ADVAPI32.dll_CryptCreateHash

```
0x004015fb      6874594400      push str.A_cryptographic_context_key_container_has_been_acquired._n ;
                                str.A_cryptographic_context_key_container_has_been_acquired._n ; "A cryptographic
                                context key container has been acquired.." @ 0x445974
0x00401600      e8eb6d0100      call fcn.004183f0
0x00401605      8b3dc8534400    mov edi, dword [sym.imp.KERNEL32.dll_GetLastError] ; [0x4453c8:4]=0x561a2
                                reloc.KERNEL32.dll_GetLastError_162 LEA sym.imp.KERNEL32.dll_GetLastError ;
                                sym.imp.KERNEL32.dll_GetLastError
; JMP XREF from 0x004015f9 (sub.ADVAPI32.dll_CryptAcquireContextA_530)
0x0040160b      83c404          add esp, 4
; JMP XREF from 0x004015e0 (sub.ADVAPI32.dll_CryptAcquireContextA_530)
0x0040160e      689cb24500      push 0x45b29c
0x00401613      6a00            push 0
0x00401615      6a00            push 0
0x00401617      6804800000      push 0x8004
0x0040161c      8b0d98b24500    mov ecx, dword [0x45b298]   ; [0x45b298:4]=0
0x00401622      51              push ecx
0x00401623      ff1500504400    call dword [sym.imp.ADVAPI32.dll_CryptCreateHash] ;
                                sym.imp.ADVAPI32.dll_CryptCreateHash
0x00401629      85c0            test eax, eax
0x0040162b      750d            jne 0x40163a
0x0040162d      ffd7            call edi
0x0040162f      50              push eax
0x00401630      6850594400      push str.Error_during_CryptBeginHash___x_n ;
                                str.Error_during_CryptBeginHash___x_n ; "Error during CryptBeginHash!
                                %x." @ 0x445950
0x00401635      e9d3000000      jmp 0x40170d
```

Address `0x00401617` is particularly interesting. Looking through the MSDN documentation for CryptCreateHash reveals `0x00008004` = `CALC_SHA` or `CALG_SHA1`. This function is creating a block of, at this time unknown, data to be SHA1 hashed.

The CryptHashData function is where the data in memory is hashed, in place, overwriting the data being hashed.

Code calling sym.imp.ADVAPI32.dll_CryptHashData

```
0x0040163a      6824594400      push str.An_empty_hash_object_has_been_created.__n ;
                                str.An_empty_hash_object_has_been_created.__n ; "An empty hash object has been
                                created. ." @ 0x445924
0x0040163f      e8ac6d0100      call fcn.004183f0
0x00401644      83c404          add esp, 4
0x00401647      6a00            push 0
0x00401649      8b550c          mov edx, dword [ebp + arg_ch] ; [0xc:4]=0xffff
0x0040164c      52              push edx
0x0040164d      8b45c8          mov eax, dword [ebp - local_38h]
0x00401650      50              push eax
0x00401651      8b0d9cb24500    mov ecx, dword [0x45b29c]   ; [0x45b29c:4]=0
0x00401657      51              push ecx
0x00401658      ff1504504400    call dword [sym.imp.ADVAPI32.dll_CryptHashData] ; "zv." @ 0x445004
0x0040165e      85c0            test eax, eax
0x00401660      750d            jne 0x40166f
0x00401662      ffd7            call edi
0x00401664      50              push eax
0x00401665      6804594400      push str.Error_during_CryptHashData__x_n ; str.Error_during_CryptHashData__x_n ;
                                "Error during CryptHashData %x." @ 0x445904
0x0040166a      e99e000000      jmp 0x40170d
```

Finally, the CryptGetHashParam function retrieves that hash from the same address in memory where the hashing occurs.

Code calling sym.imp.ADVAPI32.dll_CryptGetHashParam

```
0x0040166f      c745cc140000.   mov dword [ebp - local_34h], 0x14
0x00401676      6a00            push 0
0x00401678      8d55cc          lea edx, dword [ebp - local_34h]
0x0040167b      52              push edx
0x0040167c      8d45d0          lea eax, dword [ebp - local_30h]
0x0040167f      50              push eax
0x00401680      6a02            push 2
0x00401682      8b0d9cb24500    mov ecx, dword [0x45b29c]   ; [0x45b29c:4]=0
0x00401688      51              push ecx
0x00401689      ff1510504400    call dword [sym.imp.ADVAPI32.dll_CryptGetHashParam] ; "fv." @ 0x445010
0x0040168f      8b55cc          mov edx, dword [ebp - local_34h]
0x00401692      52              push edx
0x00401693      8d45d0          lea eax, dword [ebp - local_30h]
0x00401696      50              push eax
0x00401697      6a14            push 0x14
0x00401699      8b0b            mov ecx, dword [ebx]
0x0040169b      51              push ecx
0x0040169c      e83bb80200      call fcn.0042cedc
0x004016a1      83c410          add esp, 0x10
0x004016a4      a198b24500      mov eax, dword [0x45b298]   ; [0x45b298:4]=0
0x004016a9      85c0            test eax, eax
0x004016ab      741d            je 0x4016ca
0x004016ad      6a00            push 0
0x004016af      50              push eax
```

Debuggering to unravel the key buggery

"What one man can invent another can discover." "

Sherlock Holmes, "The adventure of the dancing men"

Now that all the elements are in place it's time to fire up a debugger and see if the bits of code suspected are actually the chunks that insert, hash, retrieve, import and use our key.

Based on the examination of the code, breakpoints can be set in the debugger and the code stepped through, as it executes, to watch the (presumed) process of key generation, importation and decryption unfold.

The breakpoints selected are:

- 0x0040163a – before CryptHashData
- 0x0040166f – before GetHashParam
- 0x004011f9 – before ImportKey
- 0x0040123a – before CryptSetKeyParam
- 0x00401260 – before CryptDecrypt

With the intercepting proxy up and running, getting things underway starts with pointing the web browser at the NetScaler Gateway. When nsepa.exe launches (before clicking '**Yes'** to start the EPA scan) open a debugger (in this case winDBG), attach to the nsepa.exe process, and set the breakpoints. For winDBG this is the `bp` command.

Example: `bp 0x0040163a`

Once the breakpoints are entered, hit `g` in the debugger to let the program continue and then click **'Yes'** in the nsepa.exe dialogue box to set things running.

When nsepa.exe gets to the point where it requests /epaq, watch what information is passed in the request headers: Cookie, Date and Host.

Requesting /epaq

```
GET /epaq HTTP/1.1
Cookie: NSC_EPAC=ec4826dea596be9b0cd540b8e1aed44d
Date: 1496885892
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; AGEE 8.0;)
Host: vpn.example.com
Cache-Control: no-cache
Connection: close
```

NetScaler Responds

```
HTTP/1.1 200 OK
TunnelType:nocmp
Set-Cookie: NSC_ERRM=xyz;Path=/;expires=Wednesday, 09-Nov-1999 23:12:40 GMT
Encode:Yes
CSEC:
8BRwfJqYM5ackW/JQ+rmvydcZC3WRixGM2coHq8t/6KAM9TLfWUCQ3fMoNVBbvXfrX/4ol6u5CC3vdNG0OTWWLG
5vanu7wuLAnvX9wTXp0A=
Content-Length: 0
Cache-control: no-cache, no-store
Pragma: no-cache
Content-Type: text/html
```

The response from the NetScaler contains the expected Base64 CSEC header. Shortly after the response from the NetScaler, the first break point is hit: `0x0040163a`

Breakpoint 0x0040163a

```
0040163a 6824594400      push    offset nsepa+0x45924 (00445924)
0:010:x86> p
nsepa+0x163f:
0040163f e8ac6d0100      call    nsepa+0x183f0 (004183f0)
0:010:x86> p
nsepa+0x1644:
00401644 83c404          add     esp,4
0:010:x86> p
nsepa+0x1647:
00401647 6a00            push    0
0:010:x86> p
nsepa+0x1649:
00401649 8b550c          mov     edx,dword ptr [ebp+0Ch] ss:002b:048efc24=00000058
0:010:x86> p
nsepa+0x164c:
0040164c 52              push    edx
0:010:x86> p
nsepa+0x164d:
0040164d 8b45c8          mov     eax,dword ptr [ebp-38h] ss:002b:048efbe0=00b86c38
0:010:x86> p
nsepa+0x1650:
00401650 50              push    eax
```

Stepping through the code till `eax` is pushed, we can see exactly what data has been placed in memory for hashing by looking at the memory location pointed to by `eax`, which is `b86c38`.

eax and memory

```
ecx     431314
eax     b86c38
ebp     48efc18
00000000`00b86c38 4e 53 43 5f 45 50 41 43 3d 65 63 34 38 32 36 64  NSC_EPAC=ec4826d
00000000`00b86c48 65 61 35 39 36 62 65 39 62 30 63 64 35 34 30 62  ea596be9b0cd540b
00000000`00b86c58 38 65 31 61 65 64 34 34 64 0d 0a 31 34 39 36 38  8e1aed44d..14968
00000000`00b86c68 38 35 38 39 32 0d 0a 76 70 6e 2e 65 78 61 6d 70  85892..vpn.examp
00000000`00b86c78 6c 65 2e 63 6f 6d 0d 0a ec 48 26 de a5 96 be 9b  le.com...H&.....
00000000`00b86c88 0c d5 40 b8 e1 ae d4 4d 00 28 63 6f 6d 70 61 74  ..@....M.(compat
00000000`00b86c98 74 11 1d 7a 97 7d 00 0c 50 88 c7 04 60 16 b8 00  t..z.}..P...`...
00000000`00b86ca8 57 69 6e 64 6f 77 73 20 4e 54 20 36 2e 30 3b 20  Windows NT 6.0;
00000000`00b86cb8 41 47 45 45 20 38 2e 30 3b 29 0d 0a 48 6f 73 74  AGEE 8.0;)..Host
00000000`00b86cc8 3a 20 76 70 6e 2e 65 78 61 6d 70 6c 65 2e 63 6f  : vpn.example.co
00000000`00b86cd8 6d 0d 0a 43 61 63 68 65 2d 43 6f 6e 74 72 6f 6c  m..Cache-Control
00000000`00b86ce8 3a 20 6e 6f 2d 63 61 63 68 65 0d 0a 0d 0a 00 00  : no-cache......
00000000`00b86cf8 5d 00 00 5d 86 7d 00 00 50 88 c7 04 60 16 b8 00  ]..].}..P...`...
00000000`00b86d08 70 6c 65 2e 63 6f 6d 00 7f 11 1d 71 95 7d 00 0c  ple.com....q.}..
00000000`00b86d18 50 88 c7 04 60 16 b8 00 38 65 31 61 65 64 34 34  P...`...8e1aed44
00000000`00b86d28 64 00 00 00 00 00 00 00 2d 11 1c 22 9e 7d 00 00  d.......-..".}..
00000000`00b86d38 50 88 c7 04 60 16 b8 00 00 00 00 00 00 00 00 00  P...`...........
00000000`00b86d48 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000`00b86d58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Looking at `b86c38` we see: The cookie passed to the web browser in ASCII (blue), followed by Carriage Return (`0x0d`) and Line Feed (`0x0a`), then the epoch Date from the Date header (red), followed by `0x0d` and `0x0a`, the value of the Host: header (yellow), followed by `0x0d` and `0x0a`, and the value of the cookie as HEX (green). If the second hypothesis is correct, then these are the elements that get hashed to make up our key.

Letting the program continue, the next breakpoint is reached at `0x0040166f`, which is near the CryptGetHashParam call. Stepping through until `eax` is pushed:

Breakpoint 0x0040166f

```
0040163a 6824594400      push    offset nsepa+0x45924 (00445924)
0:010:x86> p
nsepa+0x163f:
0040163f e8ac6d0100      call    nsepa+0x183f0 (004183f0)
0:010:x86> p
nsepa+0x1644:
00401644 83c404          add     esp,4
0:010:x86> p
nsepa+0x1647:
00401647 6a00            push    0
0:010:x86> p
nsepa+0x1649:
00401649 8b550c          mov     edx,dword ptr [ebp+0Ch] ss:002b:048efc24=00000058
0:010:x86> p
nsepa+0x164c:
0040164c 52              push    edx
0:010:x86> p
nsepa+0x164d:
0040164d 8b45c8          mov     eax,dword ptr [ebp-38h] ss:002b:048efbe0=00b86c38
0:010:x86> p
nsepa+0x1650:
00401650 50              push    eax
```

eax and memory

```
ecx     d34b8cc9
eax     48efbe8
ebp     48efc18

00000000`048efbe8 2c a1 45 dd 3a 1a 3a 18 0d 4c 02 07 3a db e3 d5  ,.E.:.:..L..:...
00000000`048efbf8 78 52 d1 3d 32 32 a2 f9 d0 fb 8e 04 38 6c b8 00  xR.=22......8l..
00000000`048efc08 7c fc 8e 04 50 cd 42 00 12 e9 69 fd 00 00 00 00  |...P.B...i.....
00000000`048efc18 88 fc 8e 04 1b 18 40 00 38 6c b8 00 58 00 00 00        @.8l..X...
```

The length of a SHA hash is 20 bytes, as shown here in blue at memory location `48efbe8`, pointed to by `eax`.

Continuing execution of the program, it becomes apparent what is being done with that hash: it is being passed to CryptImportKey:

Breakpoint 0x004011f9

```
004011f9 8d4dd8          lea     ecx,[ebp-28h]
0:010:x86> p
nsepa+0x11fc:
004011fc 51              push    ecx
0:010:x86> p
nsepa+0x11fd:
004011fd 6a00            push    0
0:010:x86> p
nsepa+0x11ff:
004011ff 6a00            push    0
0:010:x86> p
nsepa+0x1201:
00401201 8b1594b24500    mov     edx,dword ptr [nsepa+0x5b294 (0045b294)] ds:002b:0045b294=0000001c
0:010:x86> p
nsepa+0x1207:
00401207 52              push    edx
0:010:x86> p
nsepa+0x1208:
00401208 a1a0b24500      mov     eax,dword ptr [nsepa+0x5b2a0 (0045b2a0)] ds:002b:0045b2a0=00b86c20
0:010:x86> p
nsepa+0x120d:
0040120d 50              push    eax
0:010:x86> p
nsepa+0x120e:
0040120e 8b0d98b24500    mov     ecx,dword ptr [nsepa+0x5b298 (0045b298)] ds:002b:0045b298=0067d288
0:010:x86> p
nsepa+0x1214:
00401214 51              push    ecx
```

eax and memory

```
edx     1c
ecx     67d288
eax     b86c20
ebp     48efc88

00000000`00b86c20 08 02 00 00 0e 66 00 00 10 00 00 00 2c a1 45 dd  .....f......,.E.
00000000`00b86c30 3a 1a 3a 18 0d 4c 02 07 3a db e3 d5 60 16 b8 00  :.:..L..:...`...
00000000`00b86c40 74 11 1d 7a 9f 7d 00 0b f0 14 70 7c 9a 98 33 96  t..z.}....p|..3.
00000000`00b86c50 9c 91 6f c9 43 ea e6 bf 27 5c 64 2d d6 46 2c 46  ..o.C...'\d-.F,F
00000000`00b86c60 33 67 28 1e af 2d ff a2 80 33 d4 cb 7d 65 02 43  3g(..-...3..}e.C
```

From looking at the reversed code, the cryptography type is already known, AES. An AES-128 key is 16 bytes long. In the memory capture above, highlighted in blue, are the first 16 bytes of the SHA hash that was created by CryptHashData. The bytes highlighted in red were much harder to decipher, as the documentation on MSDN didn't

readily explain what this was. Searching for implementations of AES using ADVAPI32 did eventually yield up the secrets.

`08`	The key blob type: in this case PLAINTEXTKEYBLOB
`02`	The BLOB version, in this case CUR_BLOB_VERSION
`00 00`	RESERVED (not currently used for anything)
`0e 66 00 00`	Defines the key type as an AES 128 key (ALG_ID = CALG_AES_128 = 0x0000660e)
`10 00 00 00`	A DWORD defining the length of the key (16 Bytes)

At this point, the information used to create the hash is known, and the imported key is an AES-128,16 byte key, the only piece of information that is missing to successfully decrypt the base64 string passed from the NetScaler Gateway to the nsepa.exe program is the initialization vector.

Continue execution of the nsepa.exe program until the breakpoint at `0x0040123a` – CryptSetKeyParam

Breakpoint 0x0040123a - CryptSetKeyParam

```
0040123a a1a4b24500      mov     eax,dword ptr [nsepa+0x5b2a4 (0045b2a4)] ds:002b:0045b2a4=00b86c08
0:010:x86> p
nsepa+0x123f:
0040123f 85c0            test    eax,eax
0:010:x86> p
nsepa+0x1241:
00401241 741d            je      nsepa+0x1260 (00401260)                  [br=0]
0:010:x86> p
nsepa+0x1243:
00401243 6a00            push    0
0:010:x86> p
nsepa+0x1245:
00401245 50              push    eax
```

An initialization vector for AES-128 needs to be 16 bytes long, looking at the memory location when `eax` is pushed there is a segment of data, 16 bytes long. And conveniently, it's the value of the cookie passed from the NetScaler Gateway to the nsepa.exe program.

eax and memory

```
edx     48efc60
ecx     d34b8cc9
eax     b86c08
ebp     48efc88

00000000`00b86c08 ec 48 26 de a5 96 be 9b 0c d5 40 b8 e1 ae d4 4d  .H&.....@....M
00000000`00b86c18 7e 11 1d 70 99 7d 00 0c 08 02 00 00 0e 66 00 00  ~..p.}.......f..
00000000`00b86c28 10 00 00 00 2c a1 45 dd 3a 1a 3a 18 0d 4c 02 07  ....,.E.:.:..L..
```

Now onto the actual decryption of the data

When breakpoint `0x00401260` is hit, step through till `ecx` is pushed, to determine the address of the encrypted data in memory:

Breakpoint 0x0040123a - CryptSetKeyParam

```
00401260 8d45dc          lea     eax,[ebp-24h]
0:010:x86> p
nsepa+0x1263:
00401263 50              push    eax
0:010:x86> p
nsepa+0x1264:
00401264 8b4de0          mov     ecx,dword ptr [ebp-20h] ss:002b:048efc68=00b86c48
0:010:x86> p
nsepa+0x1267:
00401267 51              push    ecx
```

ecx and memory

```
edx     48efcc0
edx     0
ecx     b86c48
eax     48efc64
ebp     48efc88
00000000`00b86c48 f0 14 70 7c 9a 98 33 96 9c 91 6f c9 43 ea e6 bf  ..p|..3...o.C...
00000000`00b86c58 27 5c 64 2d d6 46 2c 46 33 67 28 1e af 2d ff a2  '\d-.F,F3g(..-..
00000000`00b86c68 80 33 d4 cb 7d 65 02 43 77 cc a0 d5 41 6e f5 df  .3..}e.Cw...An..
00000000`00b86c78 ad 7f f8 a2 5e ae e4 20 b7 bd d3 46 3b 44 d6 58  ....^.. ...F;D.X
00000000`00b86c88 b1 b9 bd a9 ee ef 0b 8b 02 7b d7 f7 04 d7 a4 e0  .........{......
00000000`00b86c98 69 00 00 69 97 7d 00 00 50 88 c7 04 60 16 b8 00  i..i.}..P...`...
00000000`00b86ca8 57 69 6e 64 6f 77 73 20 4e 54 20 36 2e 30 3b 20  Windows NT 6.0;
00000000`00b86cb8 1e 11 1c 11 95 7d 00 00 50 88 c7 04 60 16 b8 00  .....}..P...`...
00000000`00b86cc8 3a 20 76 70 6e 2e 65 78 61 6d 70 6c 65 2e 63 6f  : vpn.example.co
```

Continue steeping through the program until after CryptDecrypt is called at `0x00401272`

Breakpoint 0x0040123a - CryptSetKeyParam

```
0040126e 8b55d8          mov     edx,dword ptr [ebp-28h] ss:002b:048efc60=0069ce50
0:010:x86> p
nsepa+0x1271:
00401271 52              push    edx
0:010:x86> p
nsepa+0x1272:
00401272 ff1524504400    call    dword ptr [nsepa+0x45024 (00445024)] ds:002b:00445024={ADVAPI32!CryptDe
0:010:x86> p
nsepa+0x1278:
00401278 85c0            test    eax,eax
0:010:x86> p
nsepa+0x127a:
```

And now the data, decrypted in place, becomes clear.

Memory Address b86c48 after CryptDecrypt

```
00000000`00b86c48 61 70 70 5f 30 5f 48 44 2d 45 4e 43 5f 31 38 30  app_0_HD-ENC_180
00000000`00b86c58 5f 56 45 52 53 49 4f 4e 5f 3c 5f 32 2e 32 3b 73  _VERSION_<_2.2;s
00000000`00b86c68 79 73 5f 30 5f 44 4f 4d 41 49 4e 5f 53 55 46 46  ys_0_DOMAIN_SUFF
00000000`00b86c78 49 58 5f 61 6e 79 6f 66 5f 65 78 61 6d 70 6c 65  IX_anyof_example
00000000`00b86c88 2e 63 6f 6d 3b ef 0b 8b 02 7b d7 f7 04 d7 a4 e0  .com;....{......
00000000`00b86c98 69 00 00 69 97 7d 00 00 50 88 c7 04 60 16 b8 00  i..i.}..P...`...
00000000`00b86ca8 57 69 6e 64 6f 77 73 20 4e 54 20 36 2e 30 3b 20  Windows NT 6.0;
00000000`00b86cb8 1e 11 1c 11 95 7d 00 00 50 88 c7 04 60 16 b8 00  .....}..P...`...
00000000`00b86cc8 3a 20 76 70 6e 2e 65 78 61 6d 70 6c 65 2e 63 6f  : vpn.example.co
```

That which remains must be the truth

"Education never ends Watson. It is a series of lessons with the greatest for the last." *Sherlock Holmes, "The Red Circle"*

There it is, done.

Reversing the nsepa.exe program revealed, what elements go into making the key, how the key is made, how the key is used and the all the other information needed to reproduce the decryption process.

Repeating the same procedure as above for the encryption process shows the same hash, key import and initialization vector are used to encrypt the response from nsepa.exe to the NetScaler Gateway.

Encryption defeated. The NetScaler End Point Analysis can now be bypassed.

This paper makes it seem like a really straight forward process, but make no mistake: There were plenty of false-starts, missteps and sysiphisian learning curves. Sticking with it, through all the pitfalls, is what makes reverse engineering work.

Appendix A: The Game is Afoot.

"As a rule, the more bizarre a thing is the less mysterious it proves to be." *Sherlock Holmes, "The Red Headed League"*

Speaking of bizarre...

When trying to decide what route to take on this task, whether to reverse the nsepa.exe client or head straight to the server and start there, I did some preliminary investigation of the server side and one of the things I did was execute the UNIX `strings` command against a server binary located at /netscaler/nsppe.

To my surprise, the entire contents of Sir Arthur Conan Doyle's THE CASE-BOOK OF SHERLOCK HOLMES is contained within. But why? Next project perhaps…

strings /netscaler/nsppe

```
THE CASE-BOOK OF SHERLOCK HOLMESArthur Conan DoyleTable of contentsPrefaceThe Illustrious ClientThe Blanched SoldierThe Adventure Of The Maz
arin StoneThe Adventure of the Three GablesThe Adventure of the Sussex VampireThe Adventure of the Three GarridebsThe Problem of Thor Bridge
The Adventure of the Creeping ManThe Adventure of the Lion s ManeThe Adventure of the Veiled LodgerThe Adventure of Shoscombe Old PlaceThe A
dventure of the Retired ColourmanPREFACEI fear that Mr. Sherlock Holmes may become like one of those popularteno rs who, having outlived thei
r time, are still tempted to makerepeated farewell bows to their indulgent audiences. This must ceaseand he must go the way of all flesh, ma
terial or imaginary. One likesto think that there is some fantastic limbo for the children ofimagination, some strange, impossible place whe
re the beaux ofFielding may still make love to the belles of Richardson, whereScott s heroes still may strut, Dickens s delightful Cockneys
stillraise a laugh, and Thackeray s worldlings continue to carry on theirreprehensible careers. Perhaps in some humble corner of such aValha
lla, Sherlock and his Watson may for a time find a place, whilesome more astute sleuth with some even less astute comrade may fillthe stage
which they have vacated.His career has been a long one--though it is possible to exaggerateit; decrepit gentlemen who approach me and declar
e that hisadventures formed the reading of their boyhood do not meet theresponse from me which they seem to expect. One is not anxious tohav
e one s personal dates handled so unkindly. As a matter of coldfact, Holmes made his debut in A Study in Scarlet and in The Sign ofFour, two
 small booklets which appeared between 1887 and 1889. It wasin 1891 that  A Scandal in Bohemia,  the first of the long series ofshort storie
s, appeared in The Strand Magazine. The public seemedappreciative and desirous of more, so that from that date,                  thirty-nine
 years ago, they have been produced in a broken series                  which now contains no fewer than fifty-six stories, republished in
The Adventures, The Memoirs, The Return, and His Last Bow. And there                  remain these twelve published during the last few y
ears which are                  here produced under the title of The Case Book of Sherlock Holmes. He                  began his adventure
s in the very heart of the later Victorian era,                carried it through the all-too-short reign of Edward, and has managed
to hold his own little niche even in these feverish days. Thus it                  would be true to say that those who first read of h
im, as young men,                  have lived to see their own grown-up children following the same                  adventures
in the same magazine. It is a striking example of the                  patience and loyalty of the British public.                  I h
ad fully determined at the conclusion of The Memoirs to bring                  Holmes to an end, as I felt that my literary energies shoul
d not be                  directed too much into one channel. That pale, clear-cut face and                  loose-limbed figure
 were taking up an undue share of my imagination.                  I did the deed, but fortunately no coroner had pronounced upon the
remains, and so, after a long interval, it was not difficult for me                  to respond to the flattering demand and to explain
my rash act away.                  I have never regretted it, for I have not in actual practice found                  that these
lighter sketches have prevented me from exploring and                  finding my limitations in such varied branches of literature as
history, poetry, historical novels, psychic research, and the drama.                  Had Holmes never existed I could not have done more
, though he may                  perhaps have stood a little in the way of the recognition of my more                  serious literary wo
rk.                And so, reader, farewell to Sherlock Holmes! I thank you for your                past constancy, and can but
 hope that some return has been made in                the shape of that distraction from the worries of life and                  sti
mulating change of thought which can only be found in the fairy                  kingdom of romance.                  Arthur Conan Doyle
THE ILLUSTRIOUS CLIENT                  It can t hurt now,  was Mr. Sherlock Holmes s comment when, for the                  tenth time
in as many years, I asked his leave to reveal the                  following narrative. So it was that at last I obtained permission t
o                  put on record what was, in some ways, the supreme moment of my                friend s career.
Both Holmes and I had a weakness for the Turkish bath. It was over a                  smoke in the pleasant lassitude of the drying-room
that I have found                  him less reticent and more human than anywhere else. On the upper                  floor of th
e Northumberland Avenue establishment there is an isolated                  corner where two couches lie side by side, and it was on th
ese that                  we lay upon September 3, 1902, the day when my narrative begins. I                  had asked him wheth
er anything was stirring, and for answer he had                shot his long, thin, nervous arm out of the sheets which enveloped
him and had drawn an envelope from the inside pocket of the coat                  which hung beside him.                  It may be s
ome fussy, self-important fool; it may be a matter of                  life or death,  said he as he handed me the note.  I know no more
than this message tells me.                It was from the Carlton Club and dated the evening before. This is                  wha
t I read:                  Sir James Damery presents his compliments to Mr. Sherlock Holmes and                  will call upon him
at 4.30 to-morrow. Sir James begs to say that the                  matter upon which he desires to consult Mr. Holmes is very delicate
 and also very important. He trusts, therefore, that Mr. Holmes will                  make every effort to grant this interview, and that
 he will confirm                  it over the telephone to the Carlton Club.                I need not say that I have confirme
d it, Watson,  said Holmes as I                returned the paper.  Do you know anything of this man Damery?                  Only that t
his name is a household word in society.                Well, I can tell you a little more than that. He has rather a
reputation for arranging delicate matters which are to be kept out of                  the papers. You may remember his negotiations with
Sir George Lewis                  over the Hammerford Will case. He is a man of the world with a                  natural turn for di
plomacy. I am bound, therefore, to hope that it is                  not a false scent and that he has some real need for our assistance
.                Our?                Well, if you will be so good, Watson.                I shall be honoured.
Then you have the hour--4.30. Until then we can put the matter out                  of our heads.                  I was living in my
own rooms in Queen Anne Street at the time, but I                  was round at Baker Street before the time named. Sharp to the
half-hour, Colonel Sir James Damery was announced. It is hardly                  necessary to describe him, for many will remember that larg
e, bluff,                  honest personality, that broad, clean-shaven face, and, above all,                  that pleasant, mell
ow voice. Frankness shone from his gray Irish                  eyes, and good humour played round his mobile, smiling lips. His
lucent top-hat, his dark frock-coat, indeed, every detail, from the                  pearl pin in the black satin cravat to the lavender
 spats over the                  varnished shoes, spoke of the meticulous care in dress for which he                  was famous. The big
, masterful aristocrat dominated the little room.                  Of course, I was prepared to find Dr. Watson,  he remarked with a
courteous bow.  His collaboration may be very necessary, for we are                  dealing on this occasion, Mr. Holmes, with a man to
 whom violence is                  familiar and who will, literally, stick at nothing. I should say that                  there is no
 more dangerous man in Europe.                I have had several opponents to whom that flattering term has been                  app
lied,  said Holmes with a smile.  Don t you smoke? Then you will                  excuse me if I light my pipe. If your man is more d
angerous than the                  late Professor Moriarty, or than the living Colonel Sebastian Moran,                  then he is
indeed worth meeting. May I ask his name?                Have you ever heard of Baron Gruner?                  You mean the Austri
```

Appendix C: Python Proof of Concept

I've put together a quick and dirty PoC in python. The code requires that you have pyCrypto installed. To install pyCrypto:

```
Pip install pycrypto
```

The proof of concept code can be found here:

```
https://github.com/Lucky0x0D/NetScalerEPABypass/
```

References

[1] Citrix NetScaler: https://www.citrix.com/products/netscaler-adc/

[2] Radare2: https://rada.re/r/.

[3] WinDBG: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/

[4] Burp Suite: https://portswigger.net/burp/

[5] CryptDecrpyt: https://msdn.microsoft.com/en-us/library/windows/desktop/aa379913(v=vs.85).aspx

[6] CryptImportKey: https://msdn.microsoft.com/en-us/library/windows/desktop/aa380207(v=vs.85).aspx

[7] CryptSetKeyParam: https://msdn.microsoft.com/en-us/library/windows/desktop/aa380272(v=vs.85).aspx

[8] CryptAcquireContext: https://msdn.microsoft.com/en-us/library/windows/desktop/aa379886(v=vs.85).aspx

[9] WinCrypt.h: https://msdn.microsoft.com/en-us/library/windows/desktop/aa380248(v=vs.85).aspx.

[10] CryptCreateHash: https://msdn.microsoft.com/en-us/library/windows/desktop/aa379908(v=vs.85).aspx

[11] CryptHashData: https://msdn.microsoft.com/en-us/library/windows/desktop/aa380202(v=vs.85).aspx

[12] CryptGetHashParam: https://msdn.microsoft.com/en-us/library/windows/desktop/aa379947(v=vs.85).aspx