CSC 405
Password Security

Adam Gaweda
agaweda@ncsu.edu

Alexandros Kapravelos
akaprav@ncsu.edu

CSC 405
How to NOT Store Passwords

Adam Gaweda
agaweda@ncsu.edu

Alexandros Kapravelos
akaprav@ncsu.edu

The Naive Approach - Just Store Them!

- Nothing stopping you
  - Except you clearly know better…

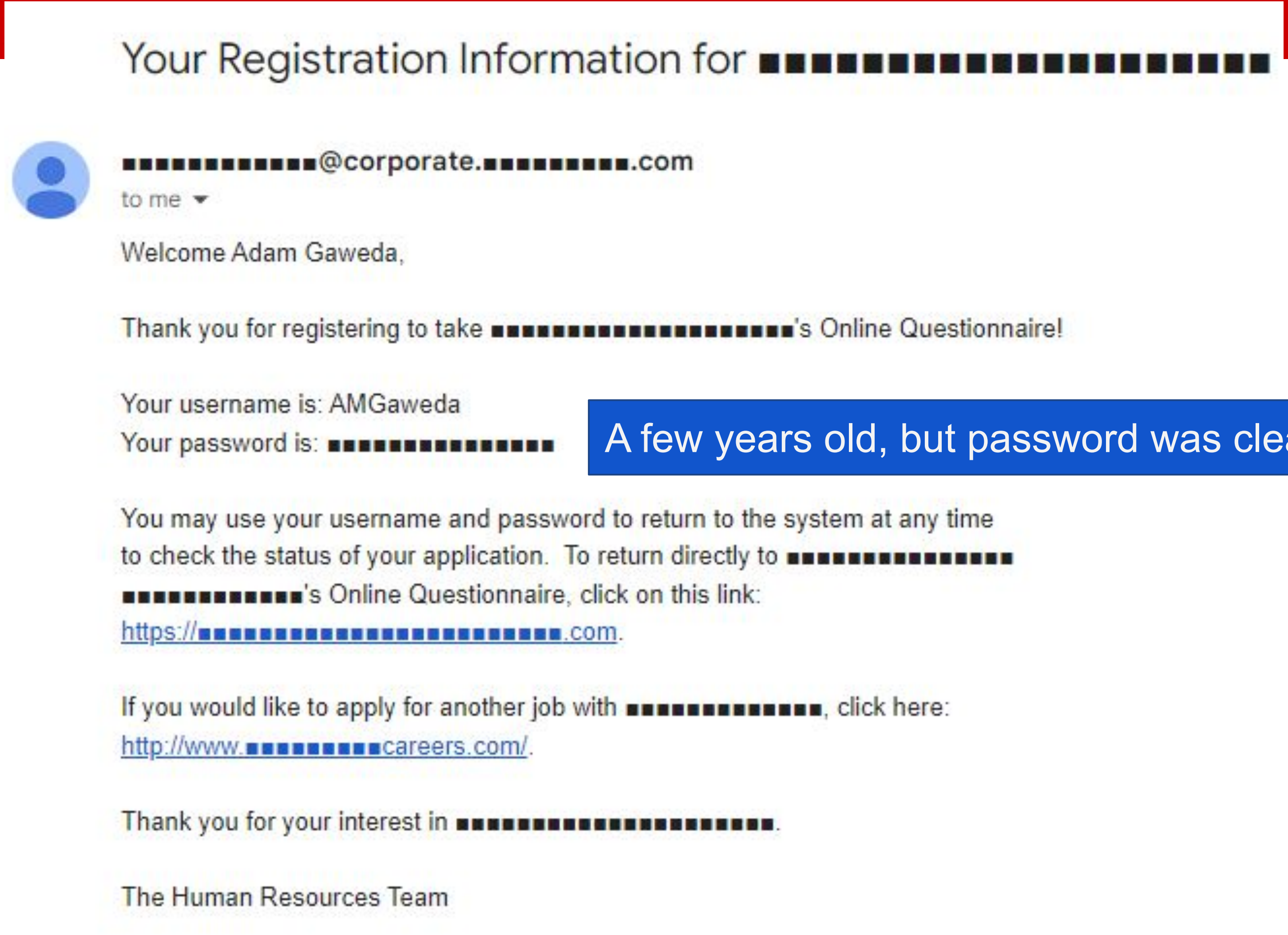
FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	password
Nancy	Edwards	nancy@chinookcorp.com	password1
Jane	Peacock	jane@chinookcorp.com	hunter22
Robert	King	robert@chinookcorp.com	robert123!@#

The Naive Approach - Just Store Them!

- Nothing stopping you
  - Except you clearly know better…

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	password
Nancy	Edwards	nancy@chinookcorp.com	password1
Jane	Peacock	jane@chinookcorp.com	hunter22
Robert	King	robert@chinookcorp.com	robert123!@#

But there are still companies that use this approach!



Your Registration Information for [redacted]

[redacted]@corporate.[redacted].com

to me

Welcome Adam Gaweda,

Thank you for registering to take [redacted]'s Online Questionnaire!

Your username is: AMGaweda
Your password is: [redacted]

You may use your username and password to return to the system at any time to check the status of your application. To return directly to [redacted]'s Online Questionnaire, click on this link: https://[redacted].com.

If you would like to apply for another job with [redacted], click here: http://www.[redacted]careers.com/.

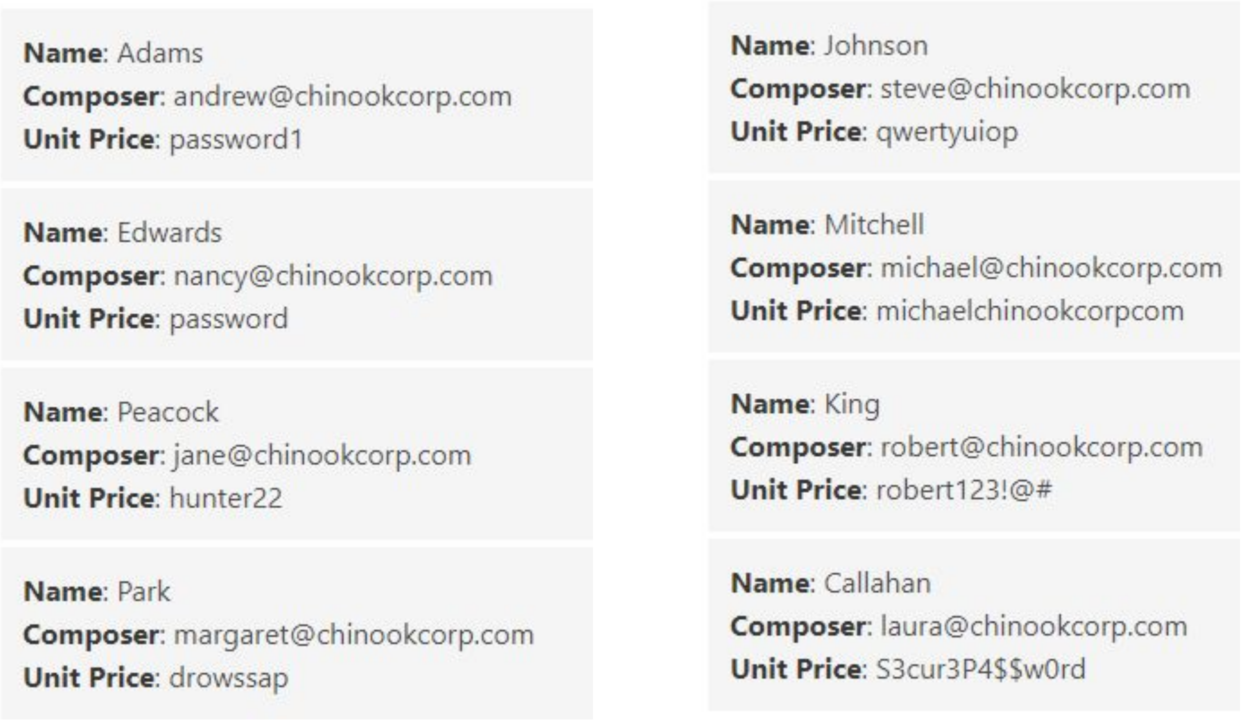
Thank you for your interest in [redacted].

The Human Resources Team

A few years old, but password was clearly plaintexted

Storing Password in Plaintext is BAD

- So… **never** do it.



Name: Adams
Composer: andrew@chinookcorp.com
Unit Price: password1

Name: Edwards
Composer: nancy@chinookcorp.com
Unit Price: password

Name: Peacock
Composer: jane@chinookcorp.com
Unit Price: hunter22

Name: Park
Composer: margaret@chinookcorp.com
Unit Price: drowssap

Name: Johnson
Composer: steve@chinookcorp.com
Unit Price: qwertyuiop

Name: Mitchell
Composer: michael@chinookcorp.com
Unit Price: michaelchinookcorpcom

Name: King
Composer: robert@chinookcorp.com
Unit Price: robert123!@#

Name: Callahan
Composer: laura@chinookcorp.com
Unit Price: S3cur3P4$$w0rd

Less Naive Approach - Encrypt It

- Good intentions… bad execution

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	cGFzc3dvcmQ=
Nancy	Edwards	nancy@chinookcorp.com	cGFzc3dvcmQx
Jane	Peacock	jane@chinookcorp.com	aHVudGVyMjI=
Robert	King	robert@chinookcorp.com	cm9iZXJ0MTIzIUAj

Base64

Less Naive Approach - Encrypt It

- Good intentions… bad execution
- Similar passwords will have similar encryptions

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	cGFzc3dvcmQ= (password)
Nancy	Edwards	nancy@chinookcorp.com	cGFzc3dvcmQx (password1)
Jane	Peacock	jane@chinookcorp.com	aHVudGVyMjI=
Robert	King	robert@chinookcorp.com	cm9iZXJ0MTIzIUAj

Base64

Less Naive Approach - Encrypt It

- Good intentions… bad execution
- Similar passwords will have similar encryptions
- Also, common encryptions have decoders online

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	cGFzc3dvcmQ= (password)
Nancy	Edwards	nancy@chinookcorp.com	cGFzc3dvcmQx (password1)
Jane	Peacock	jane@chinookcorp.com	aHVudGVyMjI=
Robert	King	robert@chinookcorp.com	cm9iZXJ0MTIzIUAj

Base64

Less Naive Approach - Encrypt It

- Good intentions… bad execution
- Similar passwords will have similar encryptions
- Also, common encryptions have decoders online

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	cGFzc3dvcmQ= (password)
Nancy	Edwards	nancy@chinookcorp.com	cGFzc3dvcmQx (password1)
Jane	Peacock	jane@chinookcorp.com	aHVudGVyMjI=
Robert	King	robert@chinookcorp.com	cm9iZXJ0MTIzIUAj

- Another way to think about it: **Encryption = Reversible**

Still Naive Approach - Hash It

- Better…
- **Hashing = Irreversible***

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	5f4dcc3b5aa7...
Nancy	Edwards	nancy@chinookcorp.com	7c6a180b3689...
Jane	Peacock	jane@chinookcorp.com	cb95015a436f...
Robert	King	robert@chinookcorp.com	3f94b11a9f70...

MD5

Password Cracking - Hashcat

```
$ hashcat --potfile-disable -m 0 pw.txt
          /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6) starting
...
Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
...
5f4dcc3b5aa765d61d8327deb882cf99:password
7c6a180b36896a0a8c02787eeafb0e4c:password1
cb95015a436fe976eb38e4545372032:hunter22
```

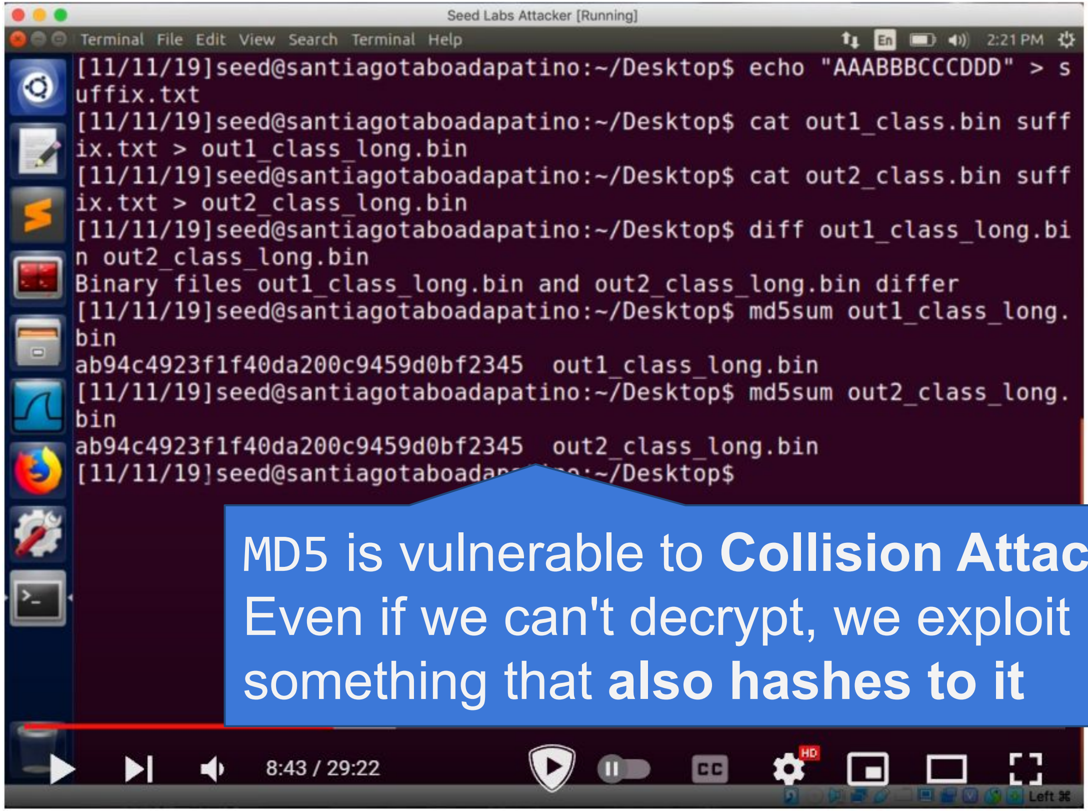
Password Cracking - Hashcat

```
$ hashcat --potfile-disable -m 0 pw.txt
          /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6) starting
...
Dictionary cache hit:
* Filename..: /usr/share/wordlist
* Passwords.: 14344385
* Bytes.....: 139921507
...
5f4dcc3b5aa765d61d8327deb882cf99:password
7c6a180b36896a0a8c02787eeafb0e4c:password1
cb95015a436fe976eb38e4545537 2032:hunter22
```

Didn't catch robert123!@# but you can add rules to append numbers/ symbols to common words

MD5 = BAD


Seed Labs Attacker [Running]
Terminal File Edit View Search Terminal Help
En
2:21 PM
8:43 / 29:22
HD
CC
Left ⌘

```
[11/11/19]seed@santiagotaboadapatino:~/Desktop$ echo "AAABBBCCCDDD" > s
uffix.txt
[11/11/19]seed@santiagotaboadapatino:~/Desktop$ cat out1_class.bin suff
ix.txt > out1_class_long.bin
[11/11/19]seed@santiagotaboadapatino:~/Desktop$ cat out2_class.bin suff
ix.txt > out2_class_long.bin
[11/11/19]seed@santiagotaboadapatino:~/Desktop$ diff out1_class_long.bi
n out2_class_long.bin
Binary files out1_class_long.bin and out2_class_long.bin differ
[11/11/19]seed@santiagotaboadapatino:~/Desktop$ md5sum out1_class_long.
bin
ab94c4923f1f40da200c9459d0bf2345  out1_class_long.bin
[11/11/19]seed@santiagotaboadapatino:~/Desktop$ md5sum out2_class_long.
bin
ab94c4923f1f40da200c9459d0bf2345  out2_class_long.bin
[11/11/19]seed@santiagotaboadap    o:~/Desktop$
```

MD5 is vulnerable to **Collision Attacks**
Even if we can't decrypt, we exploit it to find something that **also hashes to it**

Seed Labs: MD5 Collision Attack https://www.youtube.com/watch?v=qi10gDfeoU4

Still Naive Approach - Hash It

- Obviously the issue was I used MD5 instead something like SHA-128 or SHA-256!

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	5e884898da28...
Nancy	Edwards	nancy@chinookcorp.com	0b14d501a594...
Jane	Peacock	jane@chinookcorp.com	20d2fe5e369d...
Robert	King	robert@chinookcorp.com	2feb713a06cd...

SHA-256

Still Naive Approach - Hash It

- O or
  S

FirstName	
Andrew	.
Nancy	.
Jane	.
Robert	.

56

SHA-1 vs SHA-2

- The same but different (block ciphers)

- SHA-1
  - 160-bit hash
  - Can have a collision with 110 years of GPU time
    - Not super feasible for most entities, but possible

- SHA-2
  - Bit size can range from 256 to 512
  - Varying codes (SHA-224, SHA-256, SHA-384, SHA-512) refer to their output bit size

SHA-1 vs SHA-2

- The same but different (block ciphers)

- SHA-1
  - 160-bit hash
  - Can have a collision with 110 years of GPU time
    - Not super feasible for most entities, but possible

- SHA-2
  - Bit size can range from 256 to 512
  - Varying codes (SHA-224, SHA-256, SHA-384, SHA-512) refer to their output bit size

- SHA-3 is coming…

Dictionary Attacks FTW

```
$ hashcat --potfile-disable -m 1400
          pw_sha256.txt /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6) starting
...
Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
...
5e884898da28...42d8:password
0b14d501a594...c94e:password1
20d2fe5e369d...eb0b:hunter22
```

Common Passwords are super easy to attack

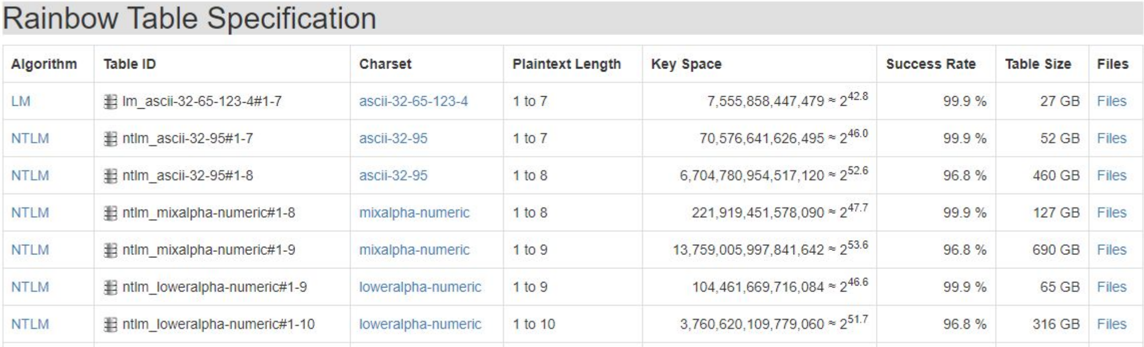
Rainbow Tables

- However, passwords like `robert123!@#` can still avoid cracking…
- Unless Robert uses it somewhere else that was hacked.

Rainbow Tables

- However, passwords like `robert123!@#` can still avoid cracking…
- Unless Robert uses it somewhere else that was hacked.

- **Rainbow Tables** are stored hash decryptions done on other passwords and stored
  - Trades computational time for hard disk space
  - LARGE file sizes



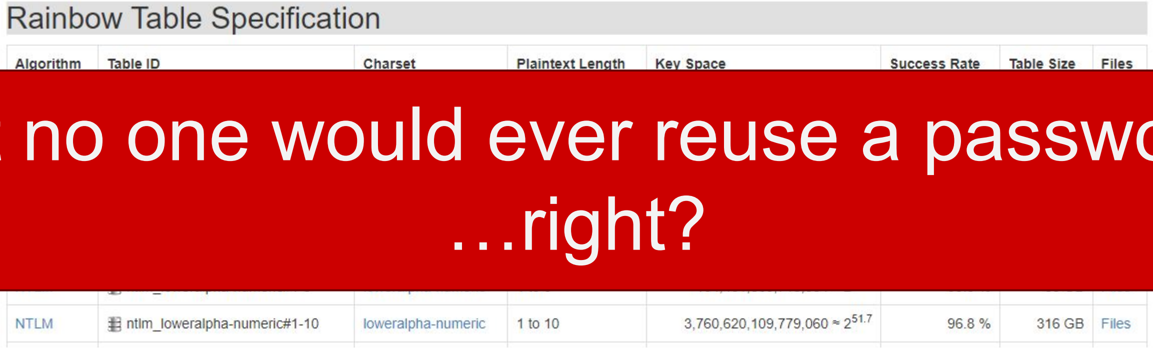
Rainbow Table Specification

Algorithm	Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files
LM	lm_ascii-32-65-123-4#1-7	ascii-32-65-123-4	1 to 7	7,555,858,447,479 ≈ $2^{42.8}$	99.9 %	27 GB	Files
NTLM	ntlm_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495 ≈ $2^{46.0}$	99.9 %	52 GB	Files
NTLM	ntlm_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120 ≈ $2^{52.6}$	96.8 %	460 GB	Files
NTLM	ntlm_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090 ≈ $2^{47.7}$	99.9 %	127 GB	Files
NTLM	ntlm_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642 ≈ $2^{53.6}$	96.8 %	690 GB	Files
NTLM	ntlm_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084 ≈ $2^{46.6}$	99.9 %	65 GB	Files
NTLM	ntlm_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060 ≈ $2^{51.7}$	96.8 %	316 GB	Files

Rainbow Tables

- However, passwords like `robert123!@#` can still avoid cracking…
- Unless Robert uses it somewhere else that was hacked.

- **Rainbow Tables** are stored hash decryptions done on other passwords and stored
  - Trades computational time for hard disk space
  - LARGE file sizes

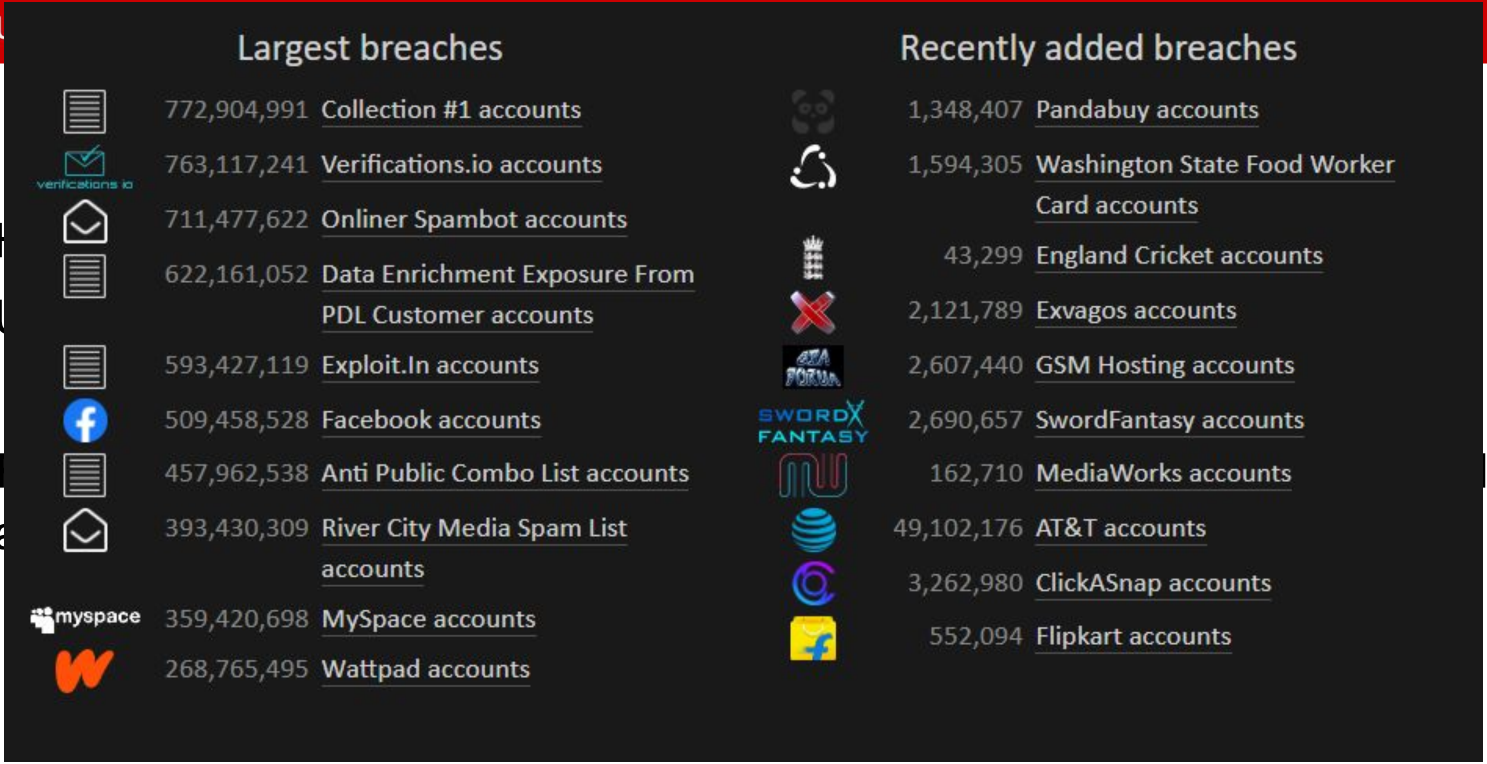


Rainbow Table Specification

Algorithm	Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files
NTLM	ntlm_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060 ≈ $2^{51.7}$	96.8 %	316 GB	Files

But no one would ever reuse a password, …right?

- H
- U
- I ls
  a



Largest breaches

772,904,991	Collection #1 accounts
763,117,241	Verifications.io accounts
711,477,622	Onliner Spambot accounts
622,161,052	Data Enrichment Exposure From PDL Customer accounts
593,427,119	Exploit.In accounts
509,458,528	Facebook accounts
457,962,538	Anti Public Combo List accounts
393,430,309	River City Media Spam List accounts
359,420,698	MySpace accounts
268,765,495	Wattpad accounts

Recently added breaches

1,348,407	Pandabuy accounts
1,594,305	Washington State Food Worker Card accounts
43,299	England Cricket accounts
2,121,789	Exvagos accounts
2,607,440	GSM Hosting accounts
2,690,657	SwordFantasy accounts
162,710	MediaWorks accounts
49,102,176	AT&T accounts
3,262,980	ClickASnap accounts
552,094	Flipkart accounts

';--have i been pwned?

haveibeenpwned.com

Current Best Approach - Salted Hash It

- Since SHA-256 will always encrypt robert123!@# to 2feb713a06…, we can mitigate this be **adding in some extra text**

FirstName	LastName	Email	Password	Salt
Andrew	Adams	andrew...	ae69caf5f4b4...	cxwnzrgwos
Nancy	Edwards	nancy...	c7bc75baf50a...	lgocdjosyn
Jane	Peacock	jane...	511dec4125ee...	bqkxuuqmbj
Robert	King	robert...	7ae0cd4700a3...	ctkwwudnyx

Current Best Approach - Salted Hash It

- Since SHA-256 will always encrypt robert123!@# to 2feb713a06…, we can mitigate this be **adding in some extra text**
- Storing the salt in the database is "fine"
  - Having the attacker know the salt does not make the task easier and still protects "`robert123!@#`" from other attacks

FirstName	LastName	Email	Password	Salt
Andrew	Adams	andrew...	ae69caf5f4b4...	cxwnzrgwos
Nancy	Edwards	nancy...	c7bc75baf50a...	lgocdjosyn
Jane	Peacock	jane...	511dec4125ee...	bqkxuuqmbj
Robert	King	robert...	7ae0cd4700a3...	ctkwwudnyx

Current Best Approach - Salted Hash It

- Since SHA-256 will always encrypt robert123!@# to 2feb713a06…, we can mitigate this be **adding in some extra text**
- Storing the salt in the database is "fine"
  - Having the attacker know the salt does not make the task easier and still protects "`robert123!@#`" from other attacks

FirstName	**LastName**	**Email**	**Password**	**Salt**
Andrew	Adams	andrew...	ae69caf5f4b4	cxvnzngves
Nancy	Edwards	nancy...		
Jane	Peacock	jane...		
Robert	King	robert...	7ae0cd4700a3...	ctkwwudnyx

Instead of hashing "**robert123!@#**", you hash "**ctkwwudnyxrobert123!@#**"

Making Salted Passwords

```
import hashlib, random, string

def make_salt(length=120):
  salt = ''
  for i in range(length):
    salt += random.choice(string.ascii_letters)
  return salt

def make_pw_hash(name, pw):
  salt = make_salt()
  to_encode = str(pw + salt).encode('utf-8')
  hashed = hashlib.sha256(to_encode).hexdigest()
  return hashed
```

Validating Salted Passwords

```
def valid_user(email, password):
  user = User.query.filter_by(email=email).first()
  salt = user.salt
  hashed_pw = make_pw_hash(password, salt)
```

Fine to store **salt** in DB, since we still need the user's input to make the hash

```
  if (user.password == hashed_pw):
    return user
  return False
```

If the hashed password doesn't equal the stored, hashed password, then invalid login

Clear Takeaways

- Salt passwords
  - Maybe add a little pepper

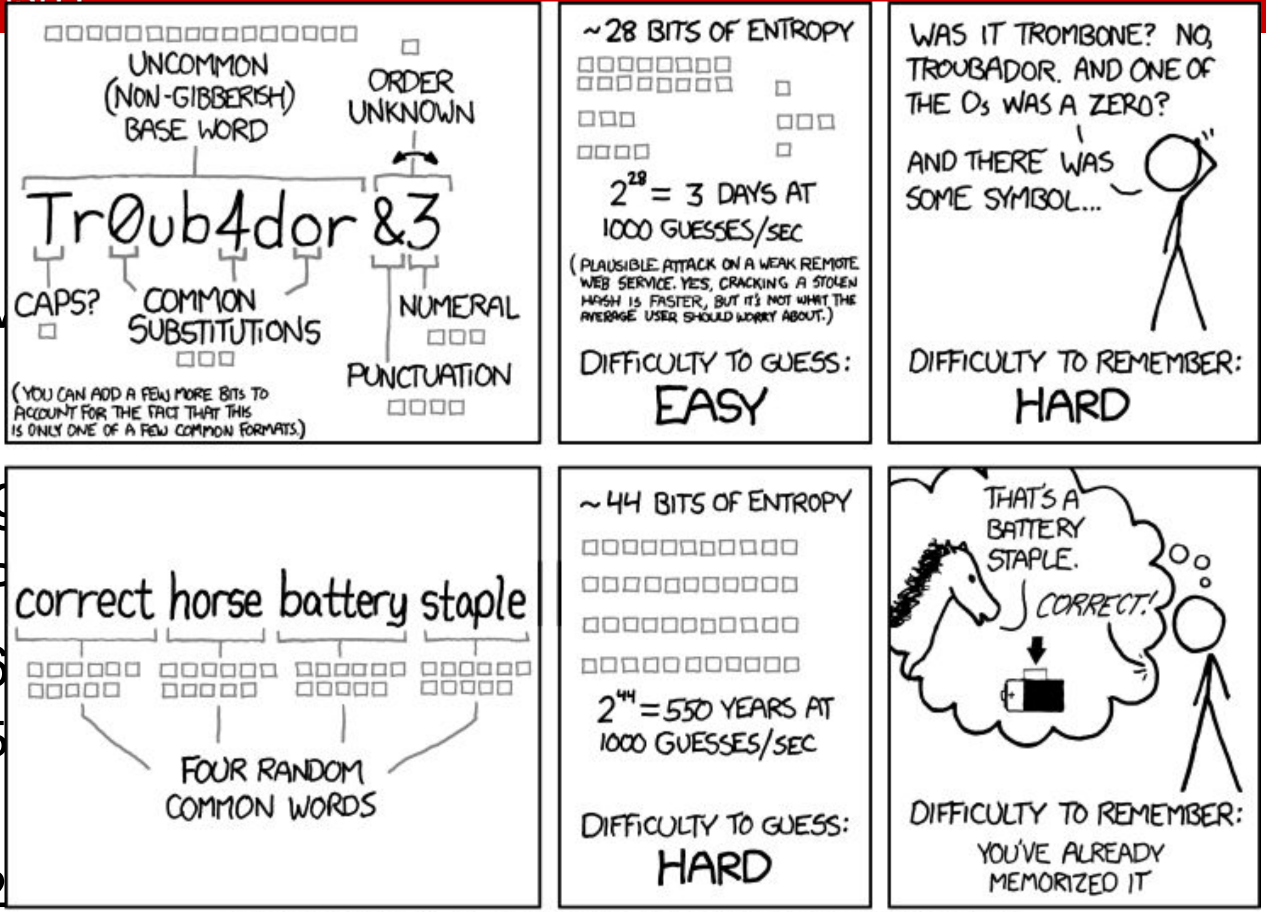
Clear Takeaways

- Salt passwords
  - Maybe add a little pepper
- Length > Complexity
  - Possibilities = complexity $^{\text{length}}$
  - 6 character password with `a-z, A-Z, 0-9` characters
    `62`^{6} `=` `56,800,235,584` possibilities

Clear Takeaways

- Salt passwords
  - Maybe add a little pepper
- Length > Complexity
  - Possibilities = complexity $^{\text{length}}$
  - 6 character password with `a-z, A-Z, 0-9` characters
    62^6 = 56,800,235,584 possibilities
  - 10 character password with only `a-z` characters
    26^{10} = 141,167,095,653,376 possibilities

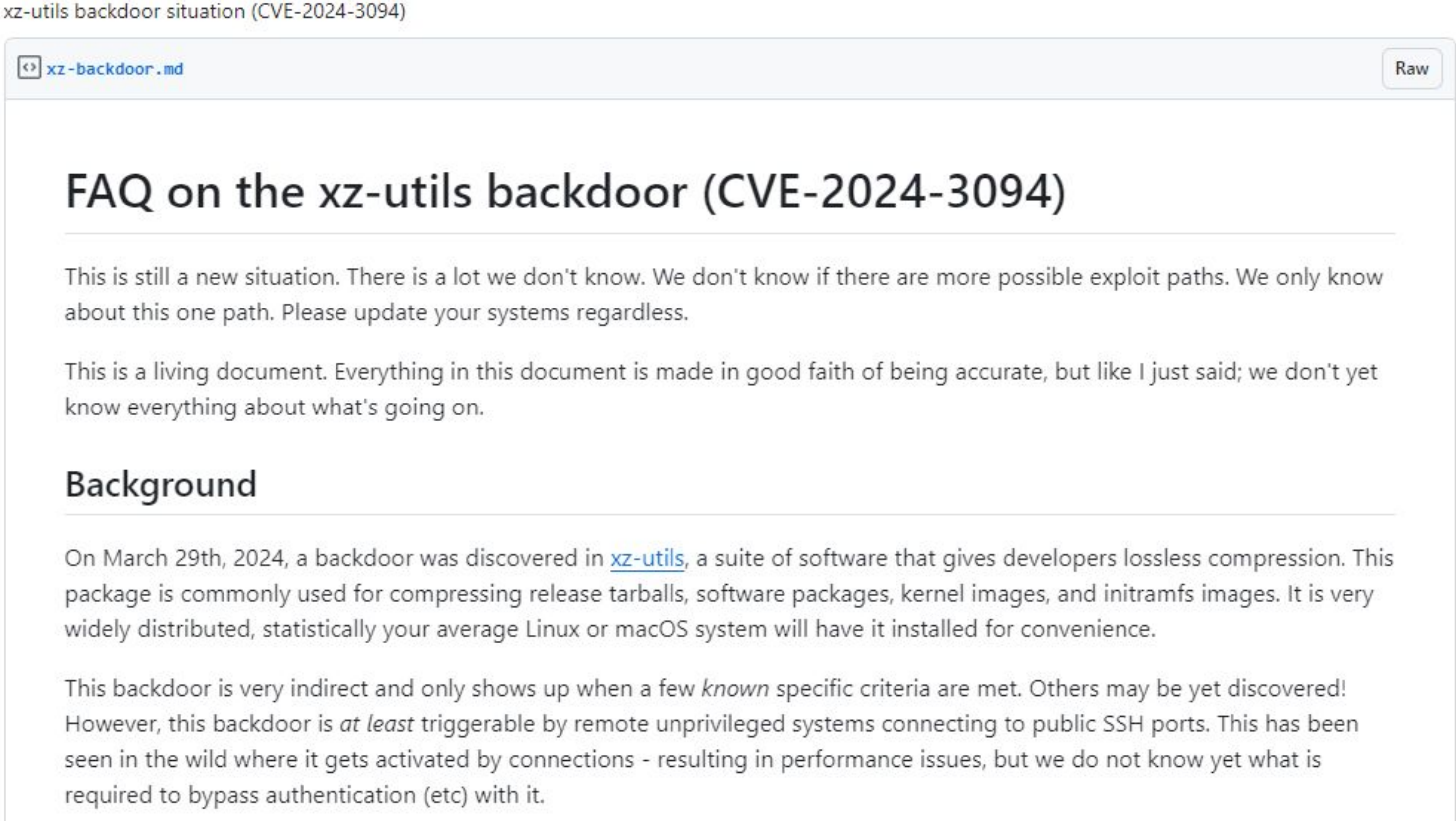
- Salt
  - M
- Leng
  - P
  - 6
    6
  - 1
    2


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

https://xkcd.com/936/

Security Zen - New RCE Weekend Drop!



FAQ on the xz-utils backdoor (CVE-2024-3094)

This is still a new situation. There is a lot we don't know. We don't know if there are more possible exploit paths. We only know about this one path. Please update your systems regardless.

This is a living document. Everything in this document is made in good faith of being accurate, but like I just said; we don't yet know everything about what's going on.

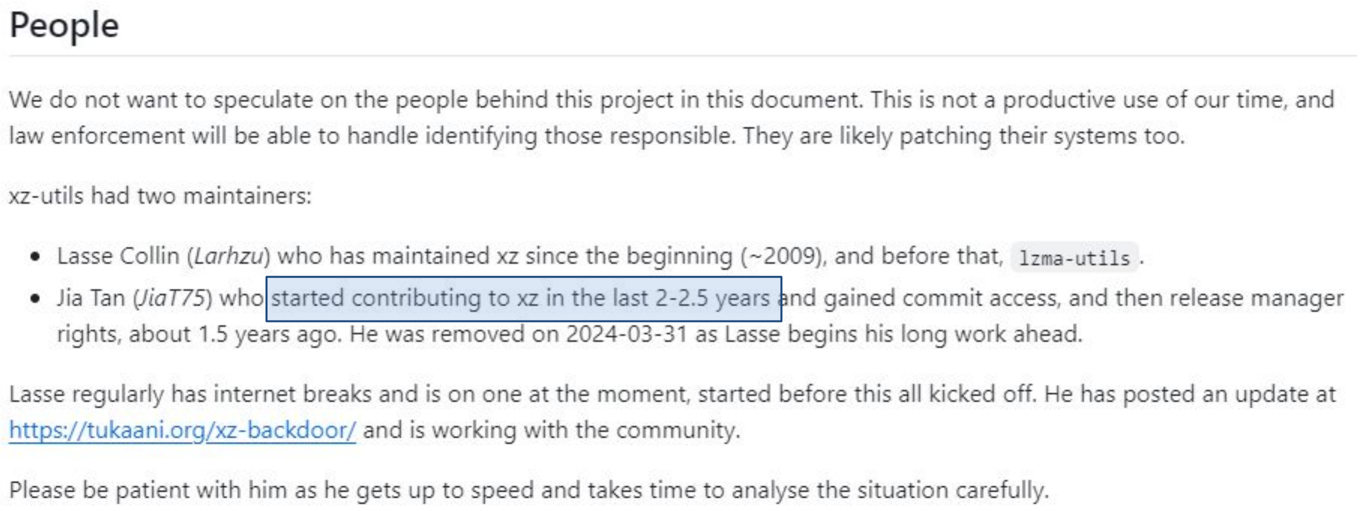
Background

On March 29th, 2024, a backdoor was discovered in xz-utils, a suite of software that gives developers lossless compression. This package is commonly used for compressing release tarballs, software packages, kernel images, and initramfs images. It is very widely distributed, statistically your average Linux or macOS system will have it installed for convenience.

This backdoor is very indirect and only shows up when a few *known* specific criteria are met. Others may be yet discovered! However, this backdoor is *at least* triggerable by remote unprivileged systems connecting to public SSH ports. This has been seen in the wild where it gets activated by connections - resulting in performance issues, but we do not know yet what is required to bypass authentication (etc) with it.

https://gist.github.com/thesamesam/223949d5a074ebc3dce9ee78baad9e27 Demo

Security Zen - New RCE Weekend Drop!



People

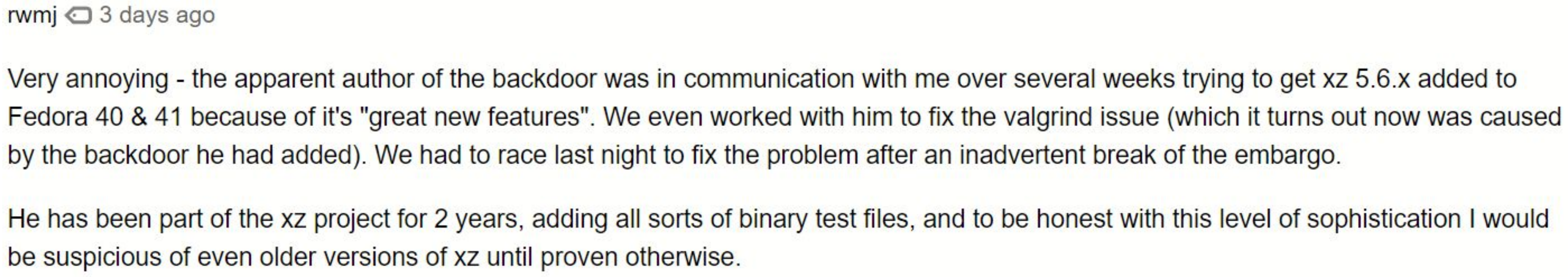
We do not want to speculate on the people behind this project in this document. This is not a productive use of our time, and law enforcement will be able to handle identifying those responsible. They are likely patching their systems too.

xz-utils had two maintainers:

- Lasse Collin (*Larhzu*) who has maintained xz since the beginning (~2009), and before that, `lzma-utils`.
- Jia Tan (*JiaT75*) who started contributing to xz in the last 2-2.5 years and gained commit access, and then release manager rights, about 1.5 years ago. He was removed on 2024-03-31 as Lasse begins his long work ahead.

Lasse regularly has internet breaks and is on one at the moment, started before this all kicked off. He has posted an update at https://tukaani.org/xz-backdoor/ and is working with the community.

Please be patient with him as he gets up to speed and takes time to analyse the situation carefully.

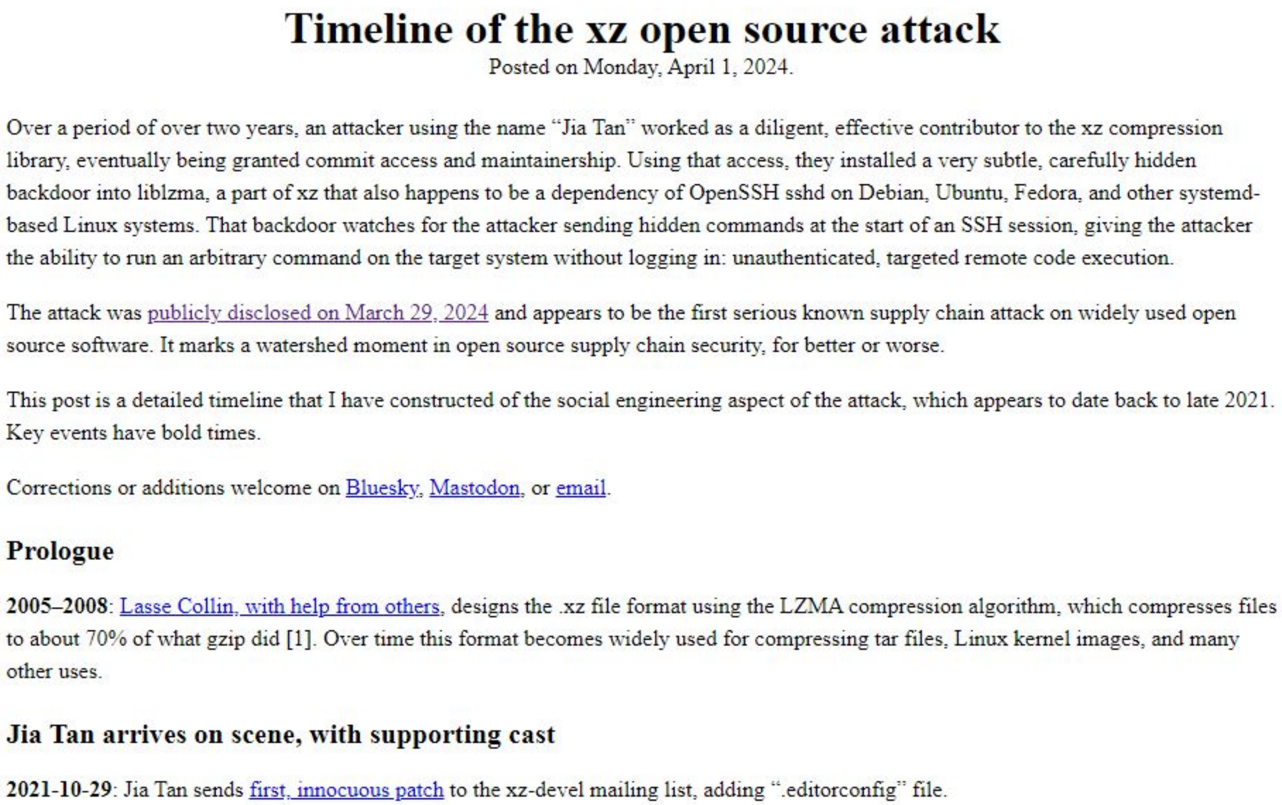


rwmj 3 days ago

Very annoying - the apparent author of the backdoor was in communication with me over several weeks trying to get xz 5.6.x added to Fedora 40 & 41 because of it's "great new features". We even worked with him to fix the valgrind issue (which it turns out now was caused by the backdoor he had added). We had to race last night to fix the problem after an inadvertent break of the embargo.

He has been part of the xz project for 2 years, adding all sorts of binary test files, and to be honest with this level of sophistication I would be suspicious of even older versions of xz until proven otherwise.

Security Zen - New RCE Weekend Drop!



Timeline of the xz open source attack

Posted on Monday, April 1, 2024.

Over a period of over two years, an attacker using the name “Jia Tan” worked as a diligent, effective contributor to the xz compression library, eventually being granted commit access and maintainership. Using that access, they installed a very subtle, carefully hidden backdoor into liblzma, a part of xz that also happens to be a dependency of OpenSSH sshd on Debian, Ubuntu, Fedora, and other systemd-based Linux systems. That backdoor watches for the attacker sending hidden commands at the start of an SSH session, giving the attacker the ability to run an arbitrary command on the target system without logging in: unauthenticated, targeted remote code execution.

The attack was publicly disclosed on March 29, 2024 and appears to be the first serious known supply chain attack on widely used open source software. It marks a watershed moment in open source supply chain security, for better or worse.

This post is a detailed timeline that I have constructed of the social engineering aspect of the attack, which appears to date back to late 2021. Key events have bold times.

Corrections or additions welcome on Bluesky, Mastodon, or email.

Prologue

2005–2008: Lasse Collin, with help from others, designs the .xz file format using the LZMA compression algorithm, which compresses files to about 70% of what gzip did [1]. Over time this format becomes widely used for compressing tar files, Linux kernel images, and many other uses.

Jia Tan arrives on scene, with supporting cast

2021-10-29: Jia Tan sends first, innocuous patch to the xz-devel mailing list, adding “.editorconfig” file.

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