Password Cracking

Sam Martin and Mark Tokutomi

CS466/566: Computer Security

April 22, 2012

The Basics

- What are passwords for?

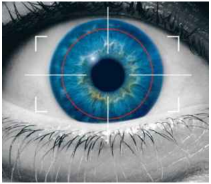
The Basics

- What are passwords for?
- Proving identity (Authentication)

The Basics

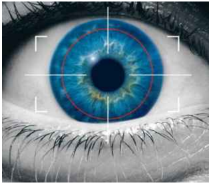
- What are passwords for?
- Proving identity (Authentication)
- There are multiple ways to authenticate yourself

Authentication



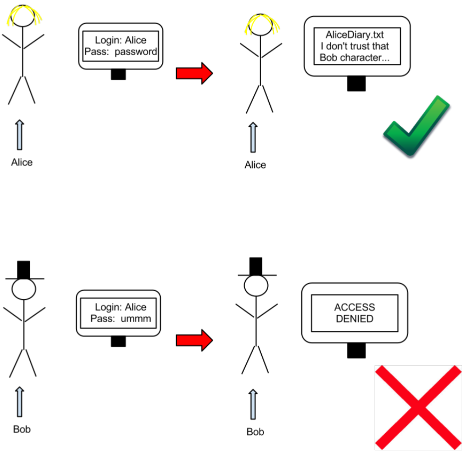
- Something you are
- Something you have

Authentication



- Something you are
- Something you have
- or...

- Something you know!



- Why would this be more or less useful?
  - Compromised authentication

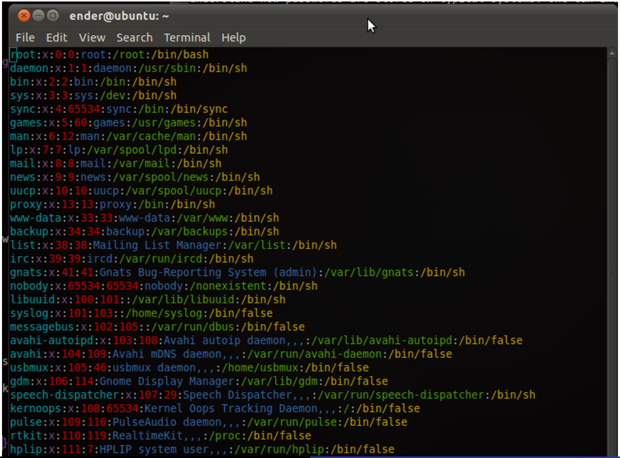
- Why would this be more or less useful?
  - Compromised authentication
  - Anonymity

- Why would this be more or less useful?
  - Compromised authentication
  - Anonymity
  - People are so bad at making passwords...

- Why would this be more or less useful?
  - Compromised authentication
  - Anonymity
  - People are so bad at making passwords...
  - Let alone keeping them secret!


Hey someone found out my password, I'd like to change it please
Not a problem sir, right away.
*computer
Oh also, my fingerprint was compromised...Can I get a new one of those too please?
BAD
POKER FACE

- Before we find out how to crack passwords, we need to know what we're fighting
- What does the Unix password file look like?



```
kernoops:x:108:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:109:116:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:110:119:RealtimeKit,,,:/proc:/bin/false
hplip:x:111:7:HPLIP system user,,,:/var/run/hplip:/bin/false
saned:x:112:121::/home/saned:/bin/false
ender:x:1000:1000:Ender,,,:/home/ender:/bin/bash
guest:x:1001:1001:guest,,,:/home/guest:/bin/bash
                                                              1,1
```

1. User or account name
2. Hash of password
3. User number
4. Group identifier
5. Gecos field
6. Home directory
7. Opening command

- But shouldn't the password file have passwords in it

- But shouldn't the password file have passwords in it
  - (I'm a well known liar)
- The actual hashes are in the shadow file

- But shouldn't the password file have passwords in it
    - (I'm a well known liar)
- The actual hashes are in the shadow file
    - The average user can't get his hands on the hashes

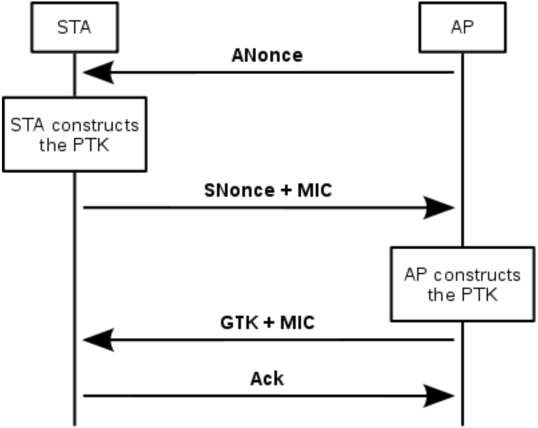
- The *S*ecurity *A*ccount *M*anager file is similar to the Unix passwd File

1. User or account name
2. User number
3. Encrypted password
4. Hash 1 of password
5. Hash 2 of password
6. Full name of user
7. Home directory

- Everyone can read the Unix passwd file
- The operating system has an exclusive lock on the Windows SAM file

- Everyone can read the Unix passwd file
- The operating system has an exclusive lock on the Windows SAM file
- Why are these different?

- WPA2 Passwords



- SQL tables for webservices

- SQL tables for webservices
- Encrypted?

- SQL tables for webservices
- Encrypted?
- Hashed?

- SQL tables for webservices
- Encrypted?
- Hashed?
- Cleartext?

- More randomness is more strength

- More randomness is more strength
- Decent systems will add randomness for you

- More randomness is more strength
- Decent systems will add randomness for you
- This strengthens passwords and makes precomputation attacks difficult

	Password	Hashed Value
No Salt	this1sAg00dPASSword!!	a5a5baa0c16166260e9ef8a48dbde112
Salted	6789o3uigtbgeat7this1sAg00dPASSword!!	53cffc58904a10b9dcc40345433862dc
Salted	v8734ihv6!nre432this1sAg00dPASSword!!	28b8f782262a890b4d730f8001d23bd5
No Salt	love	b5c0b187fe309af0f4d35982fd961d7e
Salted	12bg55tygsdf4gvi9yrdslove	65c96e15930d34dd9a9ce916b81fb044
Salted	879rughq2ebt5dfxcasedlove	a35436c0e0f2821db2703c1983a641ab

- Let's say we have access to an input screen

- Let's say we have access to an input screen
- If we want to try to crack a password, why don't we just try every one?
- There aren't very many to try right?

- Let's say we have access to an input screen
- If we want to try to crack a password, why don't we just try every one?
- There aren't very many to try right?

	lower case	lower/upper	lower/upper/digits	Ascii
1	26	52	62	95
2	676	2704	3844	9025
4	456,976	7,311,616	14,766,336	81,450,625
8	2.09×10^{11}	5.35×10^{13}	2.18×10^{14}	6.63×10^{15}
16	4.36×10^{22}	2.86×10^{27}	4.77×10^{28}	4.40×10^{31}

- Well ok, that looks like a lot...

- Well ok, that looks like a lot...
- But computers are super fast!
- Let's assume a desktop can try 1 million passwords per second

- Well ok, that looks like a lot...
- But computers are super fast!
- Let's assume a desktop can try 1 million passwords per second

	lower case	lower/upper	lower/upper/digits	Ascii
1	26 microseconds	52 microseconds	62 microseconds	95 microseconds
2	676 microseconds	2.704 milliseconds	3.844 milliseconds	9.025 milliseconds
4	$\approx$.5 seconds	$\approx$7 seconds	$\approx$14 seconds	$\approx$81 seconds
8	$\approx$2.42 days	$\approx$1.7 years	$\approx$6.9 years	$\approx$210 years
16	$\approx$1.38 billion years	$\approx$91 trillion years	$\approx$1.5 quadrillion years	$\approx$1.4 quintillion years

- Hm...Isn't there anything faster?

- Hm...Isn't there anything faster?
- Why yes there is! Some smart people programmed something that can try 2.8 billion passwords per second on a single machine.

	lower case	lower/upper	lower/upper/digits	Ascii
1	9 nanoseconds	19 nanoseconds	22 nanoseconds	34 nanoseconds
2	241 nanoseconds	966 nanoseconds	1.373 microseconds	3.223 microseconds
4	≈163 microseconds	≈2.61 milliseconds	≈5.28 milliseconds	≈29.1 milliseconds
8	≈74.6 seconds	≈5.307 hours	≈21.6 hours	≈27.4 days
16	≈.5 million years	≈32 billion years	≈.5 trillion years	≈.5 quadrillion years

- Not all passwords are created equally

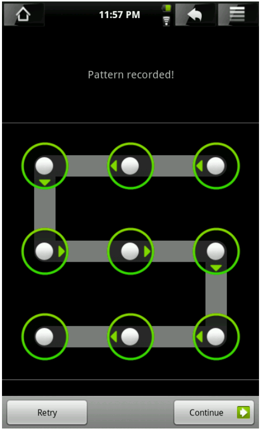
- Not all passwords are created equally
- abc123
- purple
- password
- 123456

- Not all passwords are created equally
- abc123
- purple
- password
- 123456
- We can try only common passwords

- What if we can get something like the shadow file

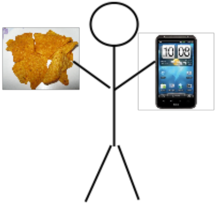
- What if we can get something like the shadow file
- Let's calculate the hashes of those common passwords

- What if we can get something like the shadow file
- Let's calculate the hashes of those common passwords
- Then we can just check for those



- How many possible passwords are there in a system where you connect only four dots?

- What are the pros of using graphical passwords?
- What are the potential drawbacks of them?
- How would you attack a graphical password scheme?



- Can you guess this person's password?

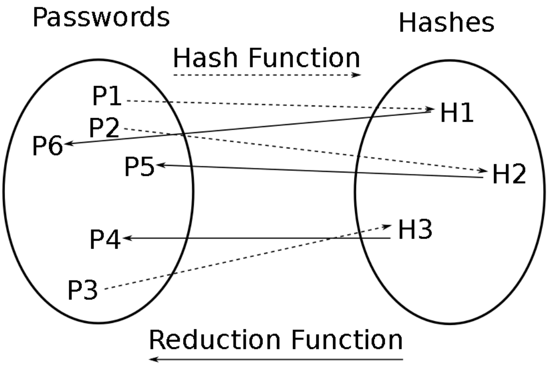
- It's been shown that we can make time-memory trade-offs when computing solutions to NP-complete problems

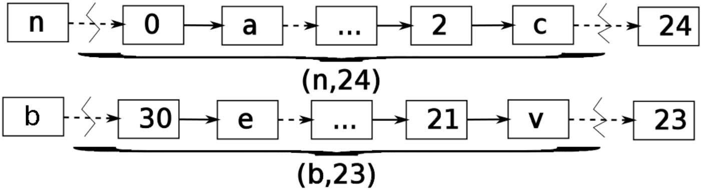
- It's been shown that we can make time-memory trade-offs when computing solutions to NP-complete problems
- Can we use the same approach here?
  - Fortunately, we can!

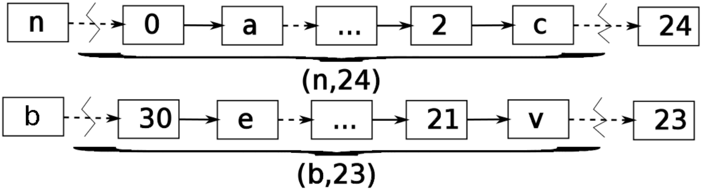
- In 1980, Martin Hellman proposed a method of creating "chains" of hashes, and storing them in a table

- In 1980, Martin Hellman proposed a method of creating "chains" of hashes, and storing them in a table
- The chains are built from the hash function and a reduction function, which maps hashes back into keyspace

- In 1980, Martin Hellman proposed a method of creating "chains" of hashes, and storing them in a table
- The chains are built from the hash function and a reduction function, which maps hashes back into keyspace
  - We can reduce/hash the hash we are attacking repeatedly, until we hit one of the table's end points
  - Once we know the row, we can chain from the start point to find the inverse of the hash

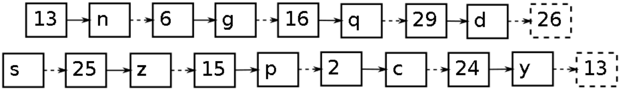

Passwords
Hashes
Hash Function
P1
P2
P6
P5
P4
P3
H1
H2
H3
Reduction Function


n
0
a
...
2
c
24
(n,24)
b
30
e
...
21
v
23
(b,23)



List of Endpoints

(23, 24, 19, 22, 26, 26, 28, 23, 24, 22, 21, 20, 30)

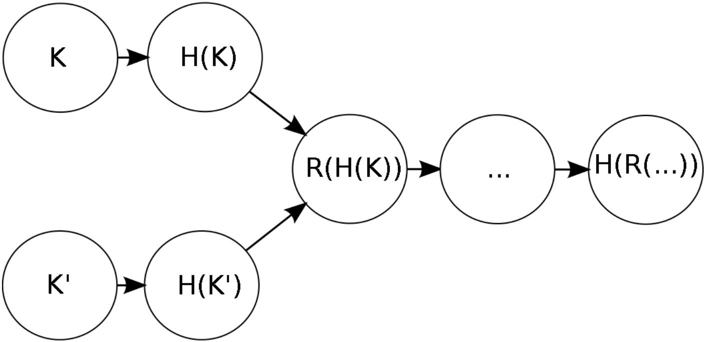


- This approach uses $N^{2/3}$ time, and $N^{2/3}$ space
- That's faster, but not fast enough:

- This approach uses $N^{2/3}$ time, and $N^{2/3}$ space
- That's faster, but not fast enough:
  - The 16-character ASCII password would still take over three hundred thousand years to crack!

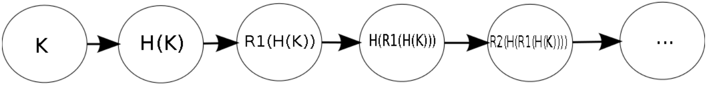
- Hellman's approach has other problems:
  - Chains can merge or loop
  - Use lots of small tables with different reduction functions
  - Distinguished points can solve these issues, as well as save time

- Hellman's approach has other problems:
  - Chains can merge or loop
  - Use lots of small tables with different reduction functions
  - Distinguished points can solve these issues, as well as save time



- Rainbow Tables offer the same improvements as distinguished points, with a greater speed increase
  - Instead of one reduction function, we'll use a family of them
  - We can only merge if the collision occurs at the same place now

- Rainbow Tables offer the same improvements as distinguished points, with a greater speed increase
  - Instead of one reduction function, we'll use a family of them
  - We can only merge if the collision occurs at the same place now



- We can use one large table instead of several smaller ones
- Because we don't need distinguished points, all rows can be the same length

- Although it may not sound significant, having chains of constant length makes application of the table substantially faster
- It both increases the lookup speed and decreases the time wasted detecting false alarms

- How can we avoid attacks which use Rainbow Tables?
    - Store salted passwords! (It really is *that* easy!)
- Why, then, are there so many tools which crack passwords using Rainbow Tables?

- How can we avoid attacks which use Rainbow Tables?
  - Store salted passwords! (It really is *that* easy!)
- Why, then, are there so many tools which crack passwords using Rainbow Tables?
  - Like most of life's problems, this can be attributed to Microsoft

- The **L**an **M**anager hash was used in early versions of Windows
- The hashes are not salted
- It also splits passwords into two sections before hashing

- The **L**an **M**anager hash was used in early versions of Windows
- The hashes are not salted
- It also splits passwords into two sections before hashing
    - Not only can we attack them in parallel, but...

- The **L**an **M**anager hash was used in early versions of Windows
- The hashes are not salted
- It also splits passwords into two sections before hashing
  - Not only can we attack them in parallel, but...
  - It halves the length of the search space (2^k vs $2^{k/2}$)
    - (that's a big difference!)

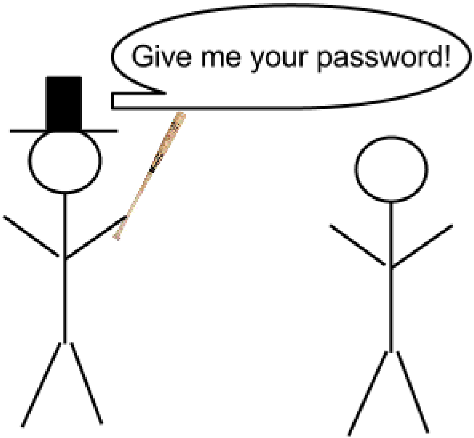
- The **L**an **M**anager hash was used in early versions of Windows
- The hashes are not salted
- It also splits passwords into two sections before hashing
  - Not only can we attack them in parallel, but...
  - It halves the length of the search space (2^k vs $2^{k/2}$)
    - (that's a big difference!)
  - It also casts all alphabetic characters to uppercase
    - This is also bad, but is pretty insignificant compared to splitting the password

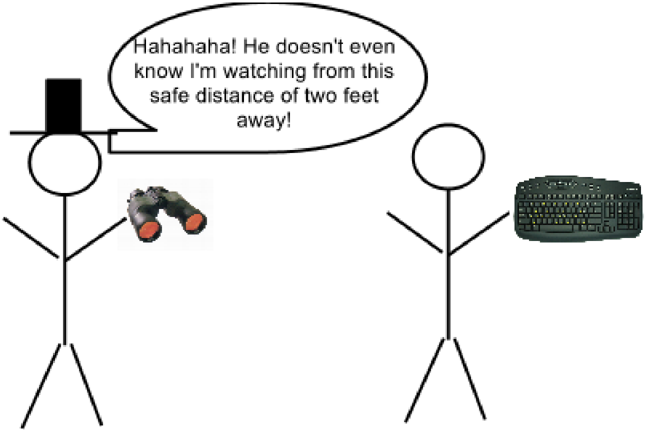
- If it's known to be terrible, why is it still used?

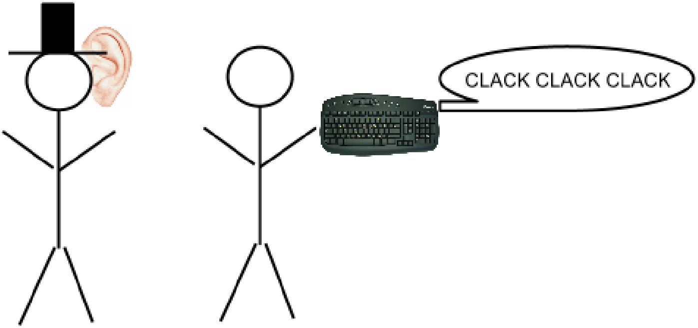
- If it's known to be terrible, why is it still used?
- Backward compatibility!

- If it's known to be terrible, why is it still used?
- Backward compatibility!
    - Versions of Windows up to (and including) XP still store it by default
    - It can't hash passwords longer than 14 characters
    - This behavior can also be disabled, but is not by default until Vista

- In-Class Exercise!
  - Assuming we can check 2.8 billion passwords per second, and they're 7-bit ASCII...
  - Approximately how long would it take to brute force a 14-character password?
  - What about a 7-character password?


Give me your password!


Hahahaha! He doesn't even know I'm watching from this safe distance of two feet away!


CLACK CLACK CLACK

Questions?