Nails in the Crypt()

Rainbow Tables for Old School Password Files

Darth Null
(David Schuetz)

www.darthnull.org
intrepidusgroup.com/insight/2010/12/crypt3-rainbow-tables

Introduction

- Who I Am:
  - Guerilla programmer, security tester, general geek, 20 years of UNIX, Perl, and other technologies
  - Dabble with password cracking off and on
- Who I Am Not:
  - Professional coder or rainbow table 'expert'
- Why I Am Here Today
  - Love the concept of rainbow tables and annoyed that they're not available for UNIX crypt()

Meta-Background

- First contemplated crypt() tables in 2007
- Early last year, decided nobody had tried
- At last minute, built some PoC code and submitted to DEFCON - rejected
- Cleaned up, resubmitted for ShmooCon
  - I got on the closing plenary!
- Released results as paper on Intrepidus Group's blog (intrepidusgroup.com/insight)

Background

- Cleartext bad - attackers can read
- Use one-way hash - attackers see gibberish
- User enters password, it's hashed, if hashes match then access granted
- Salt added to make harder

Crack!

- Dan Farmer, COPS - included ‘pwc’ (1989)
- Alec Muffett, crack - improved (1990)
- Takes dictionary and tries each word against a target hash
- Included substitutions, permutations, etc.
- Even in parallel, still takes time

Just Pre-Compute Keys?

- Way too many keys for UNIX
  - Hashes are 13 characters
  - Max passwords are 8
  - 8 char mixed alphanum: 62^8 * (13+8)
  - 4.1 PETABYTES
- Even today, that's far too big to store

Enter Rainbow Tables

- Compute all passwords, but store only a fraction
- Re-create missing ones on-the-fly
- Key Dates:
  - 1980: Martin Hellman (yes, *that* Hellman) - original Time-Memory Tradeoff paper
  - 2003: Philippe Oechslin - Faster approach, dealing with many of the problems
  - 2005: Public repositories begin appearing

Generating

Start with a 4-byte index

Convert it to plaintext

Apply the hash function

Convert hash into a new index

Index to plaintext

Plaintext to hash

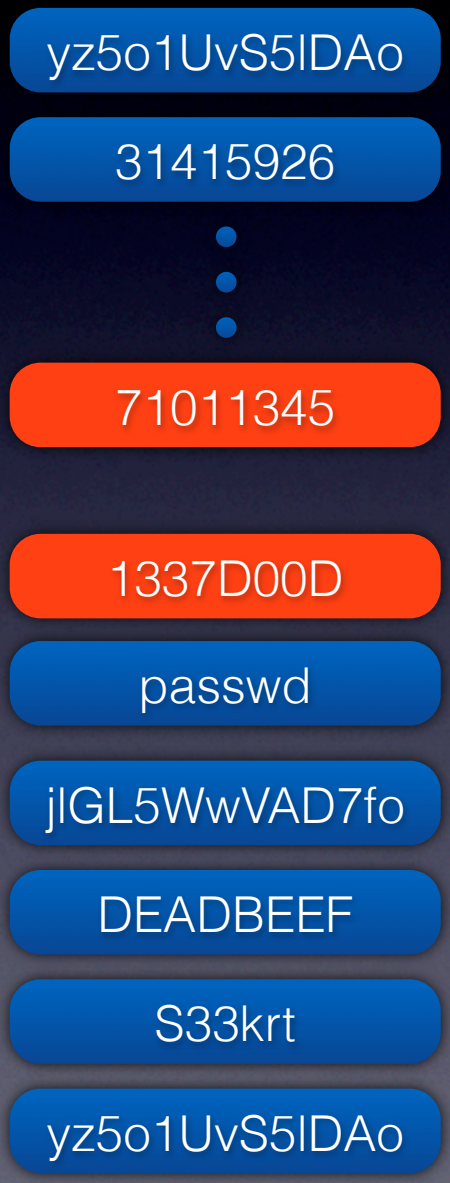
Hash to index

(etc.)

End of chain

Add
“1337D00D 71011345”
to table in progress

Cracking



Starting hash

Convert to index. In table?

No. Plain, hash, index, check, etc.

Yes, this index is in the table!

Get start of that chain and continue

Index to plaintext...

...plaintext to hash.... Match target?

No. Hash to index,

index to plain, etc.

Yes! Our password is “S33krt”

Why "Rainbow?"

- Change index-to-plain function at each step
- Reduces collisions, loops, repeats, etc.
- Continuous variation of functions, almost like a "rainbow" of functions
- See Philippe Oechslin's paper

Tables Available Today

- Several sites provide pre-built tables
- lm, ntlm, md5, sha, wpa, etc.
- Varying alphabets and password lengths
- Distributed generation
- Range from single CD (LM alpha) to multiple TB

Tables Not Available

UNIX crypt()

Why Not?

- “Nobody uses it anymore, anyway.”
- "Thousands of times more work at hundreds of times slower speeds."
- "Increases keyspace by billions, to billions of billions."
- "Need a rainbow table for every salt."

O RLY?

Nobody uses it anymore

- Websites: .htaccess
- I've personally seen it used in:
  - Tacacs
  - OpenLDAP
  - Anywhere legacy software or lazy admins are found
- Compare: "Is anyone even using LM hash or HTTP basic anymore?" ("Yes.")

Gawker Media

- Site attacked in December 2010
- 1.3 million email addresses
- 750,000+ crypt() Password Hashes

Too Slow?

	1980	2003	2010
Typical CPU	68000	P4	i7
Speed (MIPS)	1	9700	147600
Relative to 2003	0.01%	1	15 x

Cipher Speeds

(3.6 GHz P4)

	lm	md5	crypt
Hashes / sec	2.7 mil	2.7 mil	298000
Relative to lm	1	1	0.11

Not really a problem.

- Crypt is 9x slower than lm and md5
- But CPUs are 15x FASTER than in 2003
  - So, still a net performance gain
- Multi-core boxes and distributed networks
- Game changer: using the GPU

Keyspace

Charset	Size	4	6	8	10
a-z	26	475 k	321 m	217 b	146 t
a-z0-9	36	1.7 m	2.2 b	2.9 t	3.7 e 15
A-Za-z	52	7.5 m	20 t	54 t	147 e 15
A-Za-z0-9	62	15 m	58 b	220 t	853 e 15
<sp>-~	96	85 m	800 b	7 e 15	6.7 e 19

Hard Disk Sizes

	1980	2003	2010
Typical Drive	5 MB	40 GB	2 TB
Relative to 2003	1/8000	1	50 x

Table Size

8-character password, lowercase letters + numbers
Chains 10,000 hashes long

Hash	Bytes	Number of Drives		
		1980	2003	2010
LM	4.32 G	885	< 1	<< 1
Crypt	17.3 T	3.6 M	443	9

O RIy?

- Usage: Anywhere legacy is found
- Speed: Slower, still faster than brute force (for large list of hashes)
- Size: Comparable to 10 character NTLM

- But what about that pesky salt?
  - "Need a rainbow table for every hash."

Salting the Table

- crypt() salt is two characters [A-Za-z0-9./]
- Don't make table of 1-8 char passwords, 4096 times...
- ...create one table of 3-10 char passwords...
- ....using the first two characters as the salt.

Generating - Salted

Start with a 4-byte index

Convert to plaintext. Call the 1st two “salt.”

Apply the hash function.

Convert hash into a new index

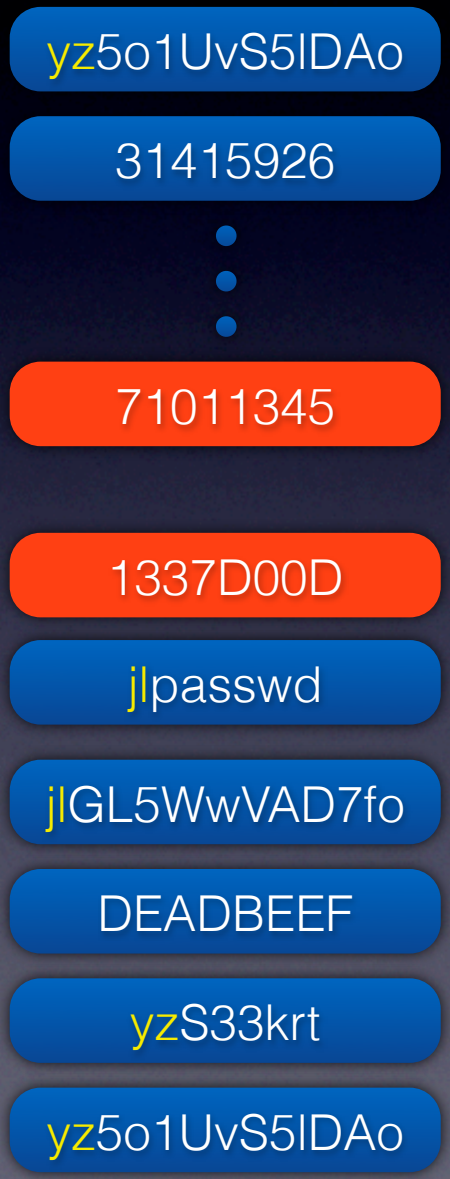
Index to plaintext

Hash to index

(etc.)

End of chain

Cracked Salt



Starting hash

Convert to index. In table?

No. Plain, hash, index, check, etc.

Yes, this index is in the table!

Get start of that chain and continue

Index to plaintext...

...plaintext to hash.... Match target?

No. Hash to index,
index to plain, etc.

Yes! Drop salt, password is “S33krt”

Code Changes

- Only a couple pages' worth
- Mostly about input and output
- Only a half-dozen lines to enable crypt()
- Details in white paper

Problems

- Are tables available? Not from me, sorry.
- Disk space - not insurmountable, but won’t fit on a DVD either
- Slower crypt - perhaps still room for improvements?
- Didn't even touch table optimization, and code is certainly messy
- More experienced rainbow geeks could do far more than I ever could

Conclusion

- Years ago, absolutely not an option
- Today, seems like it might be feasible
- Hope that this idea gets people playing

Questions?

Thanks!