KoreLogic SECURITY

PathWell: Password Topology Histogram Wear-Leveling

May 2017
Rocky Mountain Information Security Conference

Hank Leininger – KoreLogic

https://www.korelogic.com/

Agenda

My Background

Classic Password Attacks and Defenses

Recent Password Cracking Trends

PathWell Concepts

PathWell Code

Demo

Next Steps

My Background

Hank Leininger <hlein@korelogic.com>
5F6D DCC8 FF53 8093 EC39 127B 091E 7F7C E898 E86C

Played defense as a sysadmin / security admin since the mid 90's.

Wrote some Linux kernel hardening patches in the late 90’s that later became part of GRSecurity.

Have been doing security consulting since 2000; co-founded KoreLogic in 2004.

We created the Crack Me If You Can contest at DEFCON; 2015 was its 6th year running.

I also run the MARC mailing list archive site: https://marc.info/

PathWell Background

- I had the ideas for the following analysis, and the enforcement approach described later, in late 2010 or so.
- In 2013 we won a DARPA Cyber FastTrack contract to flesh out the research, design, and build a proof of concept.
- My coworkers did most of the actual work developing the PathWell PoC.

Agenda

My Background

Classic Password Attacks and Defenses

Recent Password Cracking Trends

PathWell Concepts

PathWell Code

Demo

Next Steps

Classic Password Cracking

Offline cracking

Classic Password Cracking

Offline cracking:

- Naive bruteforce (impractical)
- Wordlists
- Mangling rules

Classic Password Cracking

Offline cracking:

- Naive bruteforce (impractical)
- Wordlists
- Mangling rules

Popular classic tools: Crack, L0phtCrack, John the Ripper

Classic Defenses

- Password complexity rules
  - Minimum length
  - Character classes
- Password rotation
  - History retention
- Better hash types (rarely implemented)

Agenda

My Background

Classic Password Attacks and Defenses

Recent Password Cracking Trends

PathWell Concepts

PathWell Code

Demo

Next Steps

Recent Trends: Attacker Advantage

Today the deck is stacked in the attackers' favor.

- Enterprise software vendors haven't moved to stronger hash types.
- Moore's law has helped attackers tremendously.
- Existing defenses (password policies) have lead to exploitable predictability.
- Systems with design flaws are vulnerable to pass-the-hash attacks, which can make password cracking unnecessary.

Recent Trends: Attacker Advantage

- **Legacy systems** mean we are still using hash types we have known were too weak for many years now.
  - **UNIX DES** was replaced with better things in free UNIXes since the 90's, but it's only fairly recently that commercial UNIXes have gotten better options.
  - **NTLM**, the strongest hash type offered by the latest Microsoft products, was too weak to use even when it was new in 1993.
  - **{SSHA}**, single-round salted SHA-1, is the best offered by many enterprise LDAP directories.
- **GPU power has made selective brute-forcing practical** for these weak hashes, even for quite long password lengths.

Recent Trends: Attacker Advantage

Password Policies create new exploitable predictability:

- **Complexity rules** result in users choosing and placing their uppers, lowers, numbers, and specials in predictable ways:
  - Capitalize the first letter(s) of words (**WeakSauce**)
  - Numbers likely to be at the end, and to be a year (**WeakSauce2017**)
  - Add specials to the end (**WeakSauce2017!**)
  - Predictable character choice - '**!**' is the most common special character by a huge margin
- **Password rotation** results in users simply modifying their old passwords in predictable ways:
  - "**Oct0b3r!**" → "**N0v3mb3r!**"
  - "**Winter2016!**" → "**Spring2017!**"
  - "**qWErt78()**" → "**wERty89)_**"

Naive Brute Force

For about $2,000 you could build a machine with two NVidia 1080 Ti GPUs.

Each GPU can try over 64,000,000,000 candidate plaintexts per second against a list of NTLM hashes, which means about 128 billion per second for the system.

That machine could try all possible 8-character NTLM passwords using printable ASCII (95^8) in 14 hours.

But as you add length, the time gets longer quickly:

- 9 characters: 57 days (or less than 3 days for 20 machines!)
- 10 characters: 15 years
- 11 characters: 1,400+ years
- 15 characters: 114 billion years

Selective Brute Force – Password Patterns

- Rather than testing all possible passwords, pick some specific subsets, or patterns, and try all passwords that fit that pattern (“topology”).
- For instance, "P4ssword17!", "N0vember24@", "B7onchos99#" all use the same pattern: Uppercase, number, 6 lowercase, 2 numbers, special.
- We will use the same notation as the Hashcat tools:
  - 'u' to represent "any uppercase letter"
  - 'l' for "lowercase letter"
  - 'd' for "digit"
  - 's' for "special" (punctuation)
- The above example is then “?u?d?l?l?l?l?l?l?d?d?s”, or just “udlllllldds” for short.

Selective Brute Force – Password Patterns

- u, l, d, s = four possible character sets per password character.
- 8 character password: 4^8, or 65,536 possible topologies
- 9 character: 4^9 = 262,144
- 10 character: 4^10 = 1,048,576
- 11 character: 4^11 = 4,194,304
- The 11-character topology udllllldds has 265 trillion possible passwords (A0aaaaaa00! - Z9zzzzzz99~):
  - 26 * 10 * 26^6 * 10^2 * 33 = 265,049,735,808,000

- Our example cracking machine, which would take 1,400 years to exhaust the entire 11-character space, could bruteforce that one topology in just **35 minutes**.

Predictable Password Topologies

- The question then is: do users bias towards certain common password topologies?
- If you can guess which patterns users have over-used, you can effectively bruteforce just those topologies, and crack a disproportionate number of passwords.
  - In reality you would likely combine that with wordlists, mangling rules, and character frequencies to further optimize your attack.
- We crack passwords for penetration tests or for organizations' audit teams, as part of our Password Recovery Service, all the time.
  - So we analyzed the passwords we had cracked from several different enterprises, looking for frequently used topologies.

Predictable Password Topologies

- The question then is: do users bias towards certain common password topologies? **[Spoiler: OMG YES THEY DO.]**
- If you can guess which patterns users have over-used, you can effectively bruteforce just those topologies, and crack a disproportionate number of passwords.
  - In reality you would likely combine that with wordlists, mangling rules, and character frequencies to further optimize your attack.
- We crack passwords for penetration tests or for organizations' audit teams, as part of our Password Recovery Service, all the time.
  - So we analyzed the passwords we had cracked from several different enterprises, looking for frequently used topologies.

Sample Organization #1: Fortune 100 Company

- 263,356 of 263,888 NTLM logins cracked (including histories) – over 99%
- 7,308 unique topologies found

Sample Organization #1: Fortune 100 Company

- 263,356 of 263,888 NTLM logins cracked (including histories) – over 99%
- 7,308 unique topologies found
- Most popular topologies:
  - 33,458 ullllldd (8 character)
  - 33,394 ullllllldd (9 character)
  - 27,898 ulllddddd
  - 19,190 ullllllldd
  - 13,204 ullllddddd

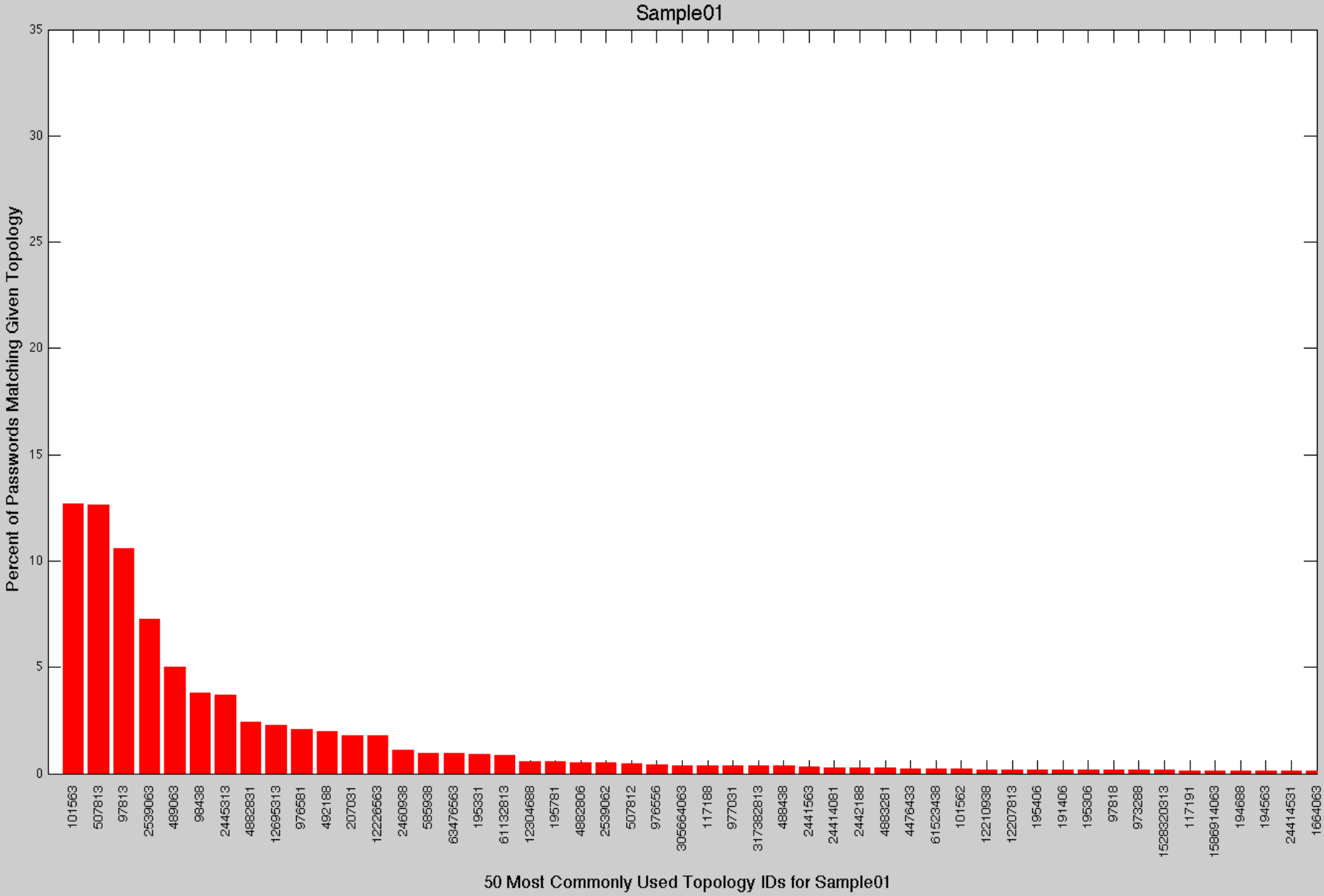
Sample Organization #1: Fortune 100 Company

- 263,356 of 263,888 NTLM logins cracked (including histories) – over 99%
- 7,308 unique topologies found
- Most popular topologies:
  - 33,458 ullllldd (8 character) – 12.7%
  - 33,394 ullllllldd (9 character) – 12.7%
  - 27,898 ulllddddd – 10.6%
  - 19,190 ullllllldd – 7.3%
  - 13,204 ullllddddd – 5.0%

Sample Organization #1: Fortune 100 Company

- 263,356 of 263,888 NTLM logins cracked (including histories) – over 99%
- 7,308 unique topologies found
- Most popular topologies:
  - 33,458 ullllldd (8 character) – 12.7%
  - 33,394 ullllldd (9 character) – 12.7%
  - 27,898 ulllllll – 10.6%
  - 19,190 ullllldd – 7.3%
  - 13,204 ulllldddd – 5.0%
- The top 5 patterns are used by a total of 48% of all users.
- The top 100 patterns are used by a total of 85% of all users.
- 99.9% of passwords meet their complexity requirements
  - They had recently increased their min length to 9.
  - Some history entries still had 8-char passwords.
  - Look at how similar the top 8-char topologies are to the top 9-char ones! They just added one lowercase letter (used a longer word).

Sample Organization #1:



Sample Organization #2: Fortune 500 Company

- 419,287 of 449,192 NTLM logins cracked (including histories) – 93%
- 14,266 unique topologies found

Sample Organization #2: Fortune 500 Company

- 419,287 of 449,192 NTLM logins cracked (including histories) – 93%
- 14,266 unique topologies found
- Most popular topologies:
  - 19,200 ullllldd (8 character)
  - 17,914 ullllldds (9 character)
  - 14,025 ulldddds
  - 12,477 ullllds
  - 9,216 ullsdddd

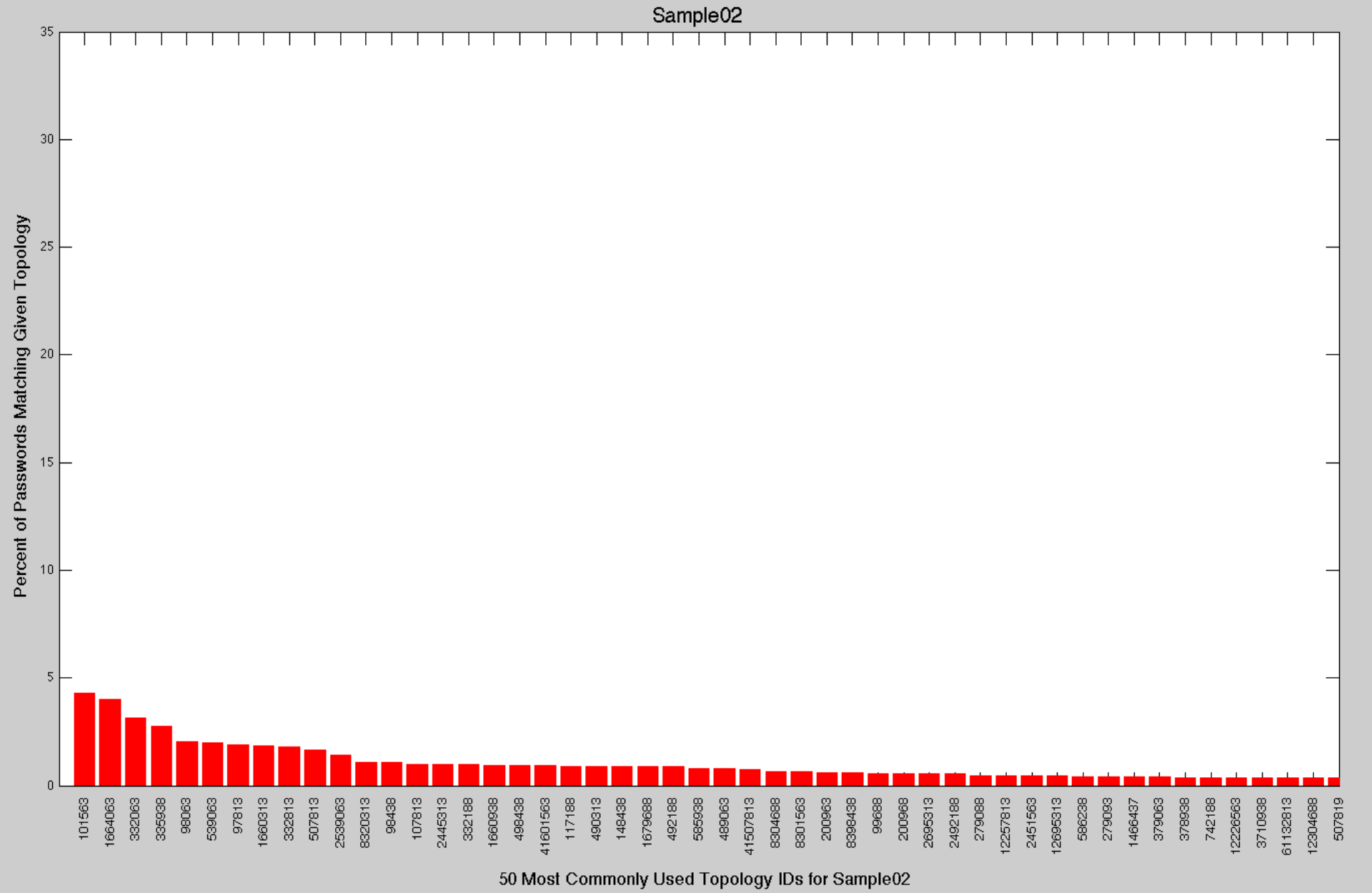
Sample Organization #2: Fortune 500 Company

- 419,287 of 449,192 NTLM logins cracked (including histories) – 93%
- 14,266 unique topologies found
- Most popular topologies:
  - 19,200 ullllldd (8 character) – 4.3%
  - 17,914 ullllldds (9 character) – 4.0%
  - 14,025 ulldddds – 3.1%
  - 12,477 ullllds – 2.8%
  - 9,216 ullsdddd – 2.1%

Sample Organization #2: Fortune 500 Company

- 419,287 of 449,192 NTLM logins cracked (including histories) – 93%
- 14,266 unique topologies found
- Most popular topologies:
  - 19,200 ullllldd (8 character) – 4.3%
  - 17,914 ullllldd**s** (9 character) – 4.0%
  - 14,025 ulldddds – 3.1%
  - 12,477 ullllld**s** – 2.8%
  - 9,216 ullsdddd – 2.1%
- Top 5 topologies crack 16% of all passwords.
- The top 100 topologies are used by a total of 62% of all users.
- They too had recently strengthened their requirements – longer minimum and required a special.

Sample Organization #2:

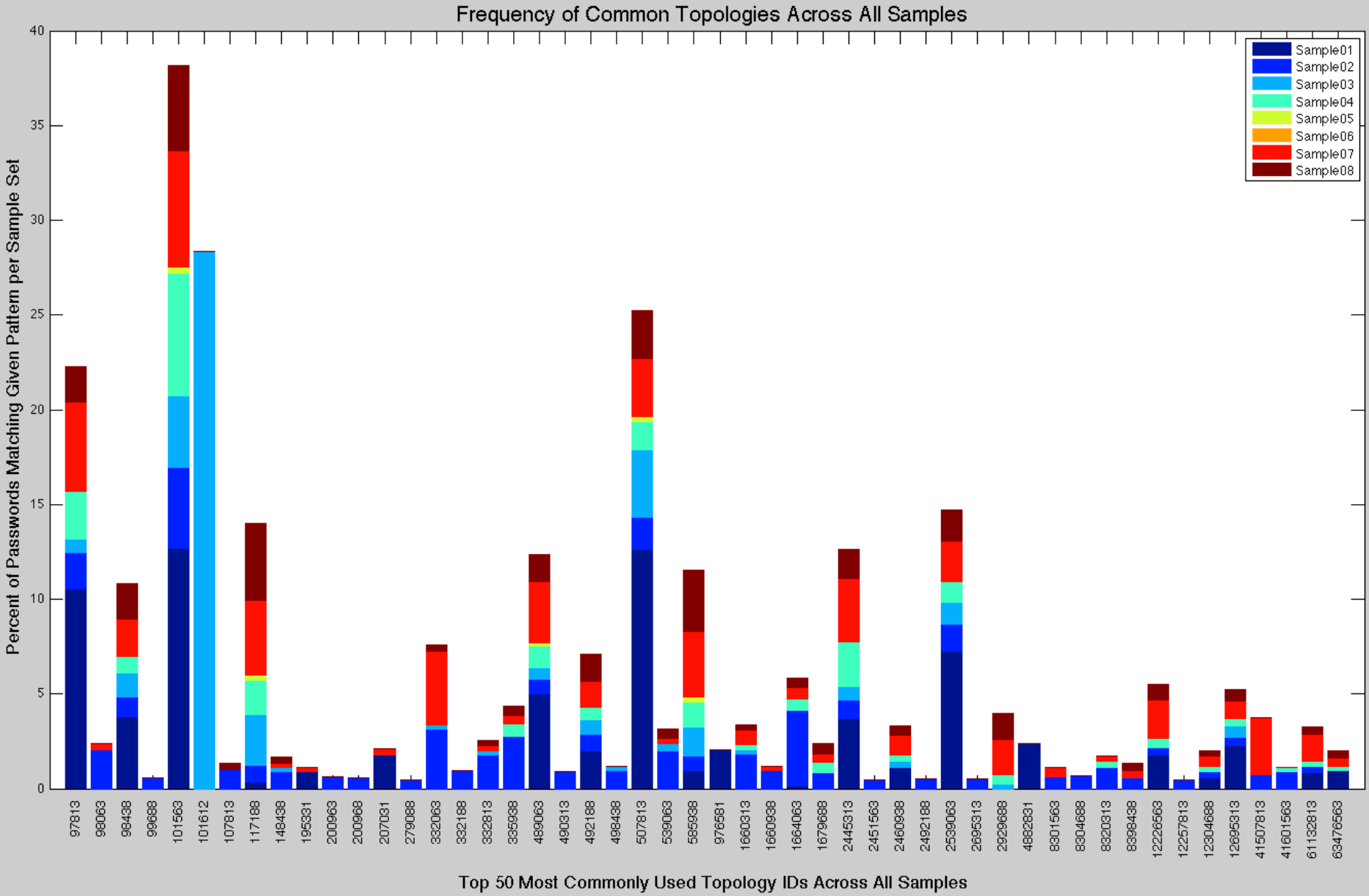


Similarities Across Organizations

We analyzed the password topologies used in 8 different enterprises of 4,000 or more logins where we had cracked more than 90% of all password hashes.

We found that they had many popular topologies in common.

Similarities Across Organizations



Things the Data Told Us

This data confirmed things we had long observed anecdotally:

- Users will pick the lowest-common-denominator that will be allowed by policies.
- When required to use 3 of 4 character classes, the most popular is: one upper, then several lowers, then 2-4 digits.
- If required to use 4 of 4 charsets, users just add a special to the end. (And most often that special character is '!')
- If the minimum length increases, users are most likely to just use a longer base word, adding a lowercase letter.
- User behavior trends apply across organizations.

Things the Data Told Us

This data confirmed things we had long observed anecdotally:

- Users will pick the lowest-common-denominator that will be allowed by policies.
- When required to use 3 of 4 character classes, the most popular is: one upper, then several lowers, then 2-4 digits.
- If required to use 4 of 4 charsets, users just add a special to the end. (And most often that special character is '!')
- If the minimum length increases, users are most likely to just use a longer base word, adding a lowercase letter.
- User behavior trends apply across organizations.

Bottom line: Complexity rules don't help as much as enterprises think they do.

How about harder passwords?

How about 15-character passwords with minimum 2 uppercase, 2 lowercase, 2 digits, 2 specials?

- To brute-force the entire keyspace would take hundreds of billions of years. It is tempting to think that they “can't be cracked”.

How about harder passwords?

How about 15-character passwords with minimum 2 uppercase, 2 lowercase, 2 digits, 2 specials?

- To brute-force the entire keyspace would take hundreds of billions of years. It is tempting to think that they “can't be cracked”.
- But what if the attacker targets popular topologies?
  - I would guess one of the top-5 patterns would be two words, first letters capitalized, separated by a special, with some numbers and another special at the end:
    - Kore.Rules2017! (ulllsullllddddds, 5.9 x 10^19 possibilities)

How about harder passwords?

How about 15-character passwords with minimum 2 uppercase, 2 lowercase, 2 digits, 2 specials?

- To brute-force the entire keyspace would take hundreds of billions of years. It is tempting to think that they “can't be cracked”.
- But what if the attacker targets popular topologies?
  - I would guess one of the top-5 patterns would be two words, first letters capitalized, separated by a special, with some numbers and another special at the end:
    - Kore.Rules2017! (ulllsullllddddds, 5.9 x 10^19 possibilities)
  - That topology would take 15 compute-years to exhaust.
    - Or, 1% every 53 days.

How about harder passwords?

How about 15-character passwords with minimum 2 uppercase, 2 lowercase, 2 digits, 2 specials?

- To brute-force the entire keyspace would take hundreds of billions of years. It is tempting to think that they “can't be cracked”.
- But what if the attacker targets popular topologies?
  - I would guess one of the top-5 patterns would be two words, first letters capitalized, separated by a special, with some numbers and another special at the end:
    - Kore.Rules2017! (ulllsullllddddds, 5.9 x 10^19 possibilities)
  - That topology would take 15 compute-years to exhaust.
    - Or, 1% every 53 days.
  - For 100,000 users with 9 history records, even if only 1% use this pattern, **you will average cracking one password every 13 hours**.

Agenda

My Background

Classic Password Attacks and Defenses

Recent Password Cracking Trends

PathWell Concepts

PathWell Code

Demo

Next Steps

Defenses Need to Evolve

- We need to add a new dimension to password strength enforcement.
- Rules like minimum length, minimum character sets required, no dictionary words, etc are still needed.
- But we also need a way to prevent users from gravitating towards the same password patterns (topologies) and overusing them.

Topology-Related Defenses

What are some ways we could use this knowledge to take away this attacker advantage?

- **Blacklist** the most common, predictable topologies.
- Require a **minimum topology change** between old and new passwords.
- Don't allow multiple users to stack up on the same topology – force them to spread out. "**Wear-Level**" them across the possible topology space.

Topology-Related Defenses

What are some ways we could use this knowledge to take away this attacker advantage?

- **Blacklist** the most common, predictable topologies.
- Require a **minimum topology change** between old and new passwords.
- Don't allow multiple users to stack up on the same topology – force them to spread out. "**Wear-Level**" them across the possible topology space.

The primary costs of these are keyspace reduction, and user rebellion.

PathWell: Topology Blacklisting

PathWell: Topology Blacklisting

- Identify the worst (most common) topologies, and do not let any users set passwords that use them.
- The top-5 lists I showed earlier are a good start.
- We further data-mined our enterprise password cracks and built a longer list. Find the list of the top 100 here:
  - https://blog.korelogic.com/blog/2014/04/04/pathwell_topologies
- Of course, these are also a good place to *start* if you are attempting to crack corporate passwords!
- As with most things we figure out, we assume bad guys had already worked this out for themselves.

PathWell: Blacklisting Effectiveness

- Attackers used to taking a top-N approach, either standalone or in combination with wordlist and other mangling-rule techniques, will suddenly get zero cracks from their early efforts, instead of ~25% in minutes.
- Attackers who figure out what's being done will have to figure out what users did when they weren't allowed to use the same old same old.
- However, it is likely that user populations would still find some new common topologies to converge on. The effectiveness of blacklist-only will decay if it is the only, static, new defense.

PathWell: Minimum Topology Change

PathWell: Minimum Topology Change

- Without wear-leveling, a user with password 'Kw#46_Ya' is most likely to set their next password to (say) 'Kw#47_Ya'
- Likewise, with wear-leveling, that user would likely chose 'Kw#46_YA' – the smallest allowable topology change.
- So: the attacker who knows what a user's password topology was in the past, should search the topologies that are “nearest” to it.
- The KoreLogicRulesReplaceNumbers ruleset published back in 2010 can easily crack these variations.

Measuring Topology Change: Levenshtein Distance

Measuring Topology Change: Levenshtein Distance

- "...[T]he Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertion, deletion, substitution) required to change one word into the other."
  - http://en.wikipedia.org/wiki/Levenshtein_distance
    - Michael Scott

Measuring Topology Change: Levenshtein Distance

- "...[T]he Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertion, deletion, substitution) required to change one word into the other."
  - http://en.wikipedia.org/wiki/Levenshtein_distance
    - Michael Scott
- Sometimes also referred to as "edit distance."
- kitten → **m**itten = 1
- abounds → abound**ed** = 2
- des**s**ert → desert = 1

Measuring Topology Change: Levenshtein Distance

For our examples earlier:

- Kw#46_Ya → Kw#4**7**_Ya
  ulsddsul → ulsddsul = Lev distance 0

- Kw#46_Ya → Kw#46_Y**A**
  ulsddsul → ulsddsu**u** = 1

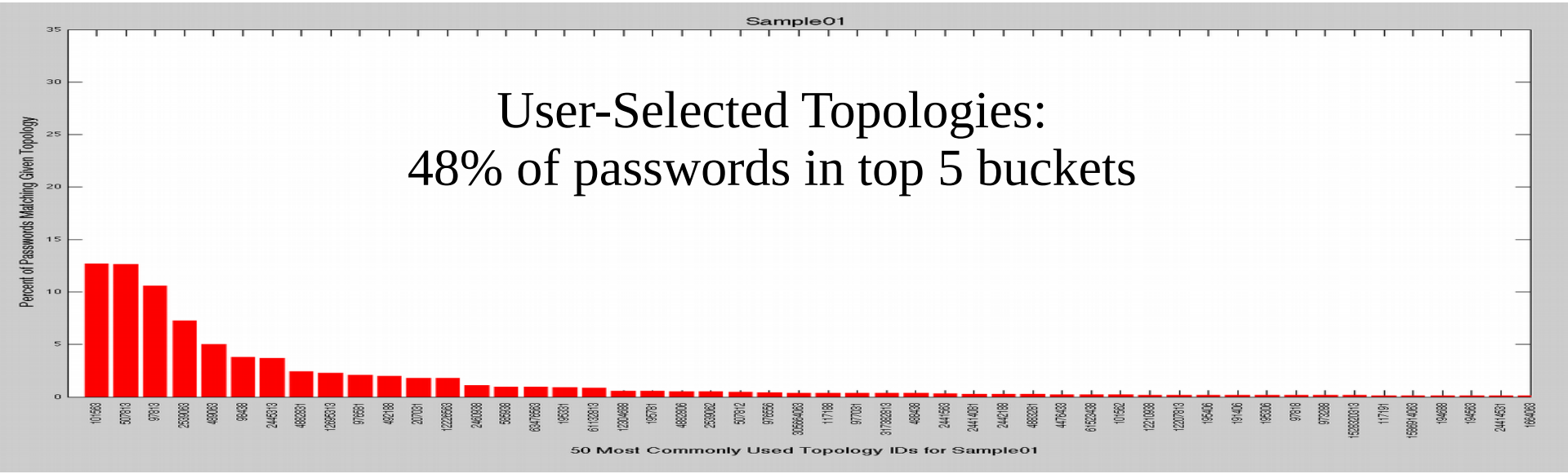
Measuring Topology Change: Levenshtein Distance

For our examples earlier:

- Kw#46_Ya → Kw#4**7**_Ya
  ulsddsul → ulsddsul = Lev distance 0
- Kw#46_Ya → Kw#46_Y**A**
  ulsddsul → ulsddsu**u** = 1
- P4ssword17! → P4sswords17!
  udlllllldds → udlllllll**l**dds = 1
- P4ssword17! → P@ssword1**8z**
  udlllllldds → u**s**llllldd**l** = 2

PathWell: Topology Histogram Wear-Leveling

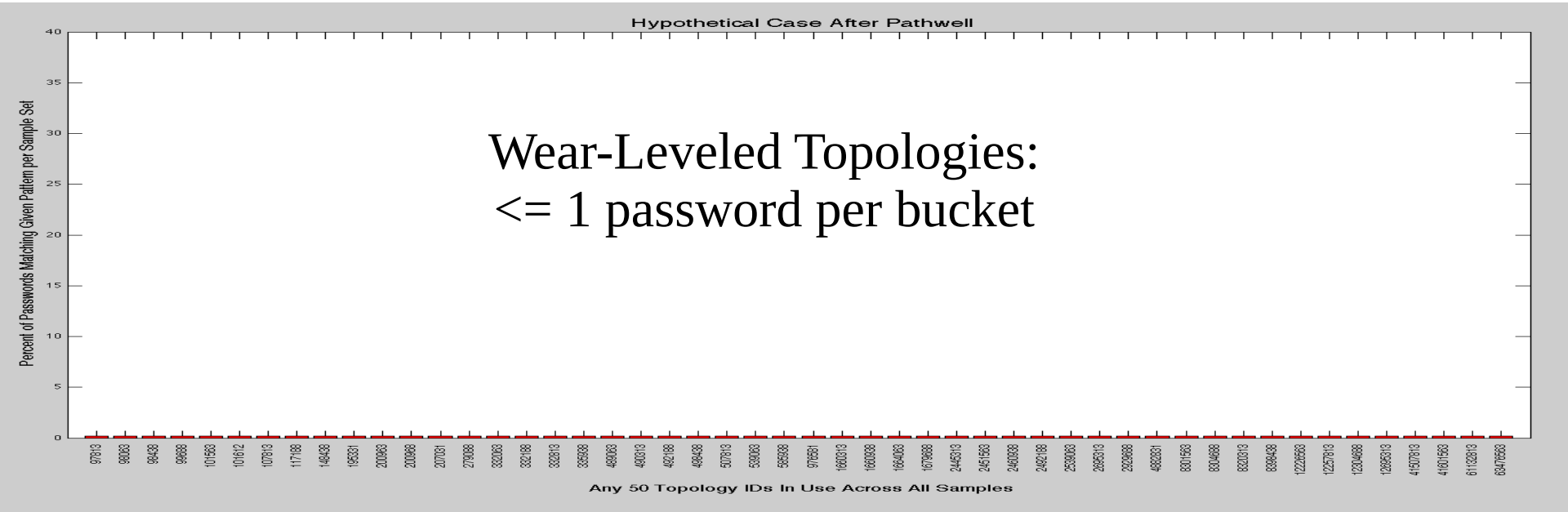
PathWell: Topology Histogram Wear-Leveling



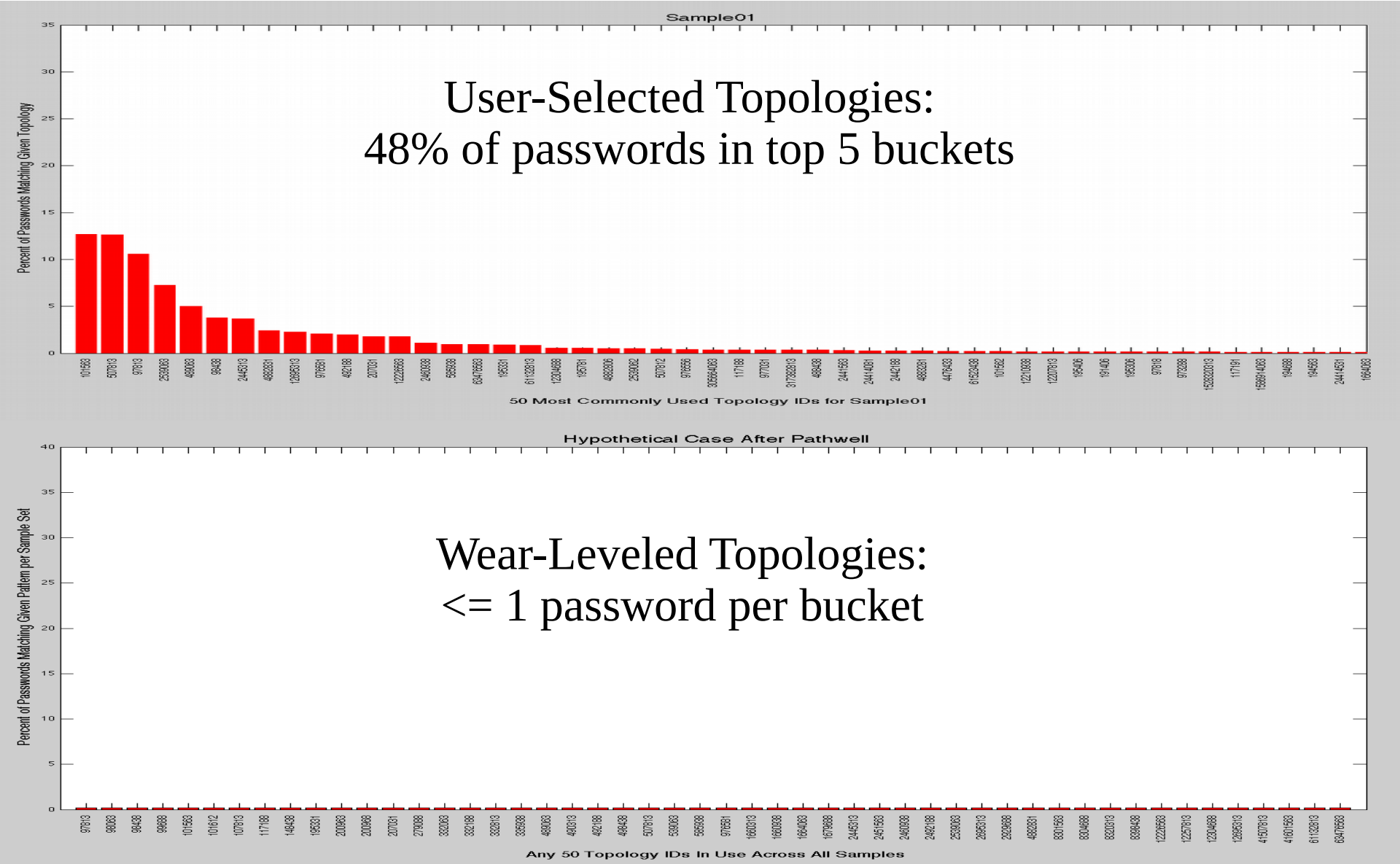
We want to turn this...

PathWell: Topology Histogram Wear-Leveling

Into this!



PathWell: Topology Histogram Wear-Leveling



Topology Wear-Leveling Effectiveness

How much does topology wear-leveling increase the attacker's work-factor?

- Attacker's work-factor thought of as "**work needed to get the same percentage of cracks**" or "**cracks for the same work.**"
- **Best-case** (fully random topologies): **6 orders of magnitude more work** (one million times as long to get the same number of cracks, or one millionth as many cracks in the same time spent).
- **Worst-case** (attacker knows and goes after only those topologies in use): **2-3 orders of magnitude more work**.
- **Realistic case** (topologies not fully random, attacker makes educated guesses): **4-5 orders of magnitude more work**.

Cost of Topology-Related Defense: Keyspace Reduction

Don't blacklisting and topology wear-leveling reduce the keyspace that an attacker would have to test for valid passwords?

How much does this keyspace reduction help the attacker?

Cost of Topology-Related Defense: Keyspace Reduction

- **Blacklisting**: For 8-character, 4-charset passwords, there are 4^8, or 65,536 topologies. 100 of them is less than 0.2% of the keyspace. That is a trivial cost and we should gladly pay it. (The percentage cost drops for longer passwords, too.)

Cost of Topology-Related Defense: Keyspace Reduction

- **Blacklisting**: For 8-character, 4-charset passwords, there are 4^8, or 65,536 topologies. 100 of them is less than 0.2% of the keyspace. That is a trivial cost and we should gladly pay it. (The percentage cost drops for longer passwords, too.)
- **Forcing unique topology use:** has the downside that the odds that any one randomly selected topology will contain a password go *up*.
  - This effect is worse for larger user populations.
  - However, this is vanishingly small compared to the cost of, say, 5-10% of all users using a single topology that the attacker can easily predict.

Cost of Topology-Related Defense: Mutiny?

Will users revolt?

Cost of Topology-Related Defense: Mutiny?

Will users revolt?

- Any new control that adds work for them will be resisted.
- Could be mitigated by user-hinting and training (which have their own costs).
- Need to figure out how users respond to new requirements, and how to best explain them to minimize difficulty.
- What kind of hints can be provided to a user who tries a password that is not strong enough?

Cost of Topology-Related Defense: Mutiny?

Will users revolt?

- Any new control that adds work for them will be resisted.
- Could be mitigated by user-hinting and training (which have their own costs).
- Need to figure out how users respond to new requirements, and how to best explain them to minimize difficulty.
- What kind of hints can be provided to a user who tries a password that is not strong enough?
  - **Hints to the user might be hints to attackers, too.**

Agenda

My Background

Classic Password Attacks and Defenses

Recent Password Cracking Trends

PathWell Concepts

PathWell Code

Demo

Next Steps

PathWell PAM Module

We released a PAM module that implements (all optional, administrator-controlled):

- Auditing
- Blacklisting
- Minimum Levenshtein distance
- Maximum use-count
- User hinting

PathWell PAM Module

- Developed and tested on multiple Linux distributions; not yet tested on any other OS's with PAM support.
- Download the source code from https://git.korelogic.com/libpathwell.git/
  - PathWell techniques described here are patented.
  - Released open-source under the AGPL.
  - Following the code license grants a patent license.
  - Basically, free if you aren’t directly making money from using it.
  - If you want to make money from using it, talk to us ;)
- The PAM module is actually a fairly thin wrapper around calls to the libpathwell library (also included). The library could be used directly by LDAP servers, Java SSO implementations, etc.

PathWell Audit Mode

Audit mode:

- Each time a password is changed, increment a counter for that password's topology.
- Usage counters are not decremented when a password is changed (history lasts forever, unless zapped by an admin).
- Useful “standalone” (without enforcement) in order to quantify the problem in a given enterprise.
- Historical data is used by use-count enforcement.
- This DB is sensitive! An attacker who captures it gets some nice hints.
- Current implementation can track topologies up to 29 characters long.

PathWell Enforcement Mode

There are several Enforcement mode options, so that features can be enabled and configured independently.

PathWell Enforcement Mode: blacklist

Enforcement mode option: **blacklisting**

- Do not allow any user to set a password that uses a known-overused topology.
- We include that list of common topologies I mentioned earlier.
- Administrators can replace or augment our default list with their own (enabling audit mode can help build up a local, organization-specific list).
- Can also be used to enforce minimum-complexity requirements (blacklist all topologies that do not use 4 of 4 character classes, etc).

PathWell Enforcement Mode: blacklist

- Note, blacklisting is not enough!
  - If users are just denied their top 100 overused choices, they will probably make similar choices about what to switch to instead.
  - We call that herding, and it is bad... in the long run, attackers just need to learn and adapt to the next-top-100 topologies and start over.
  - Instead, we want mechanisms to not herd users in a group, but rather, shoo them and disperse them more widely across the possible topology space.

PathWell Enforcement Mode: blacklist

- Note, blacklisting is not enough!
  - If users are just denied their top 100 overused choices, they will probably make similar choices about what to switch to instead.
  - We call that herding, and it is bad... in the long run, attackers just need to learn and adapt to the next-top-100 topologies and start over.
  - Instead, we want mechanisms to not herd users in a group, but rather, shoo them and disperse them more widely across the possible topology space.
- ...But it is better than nothing. You don't have to run faster than the bear...

PathWell Enforcement Mode: minlev

Enforcement mode option: **minlev**

- PathWell's minlev enforcement compares a user's old password's topology to the requested new one.
- **minlev=1**: new password's topology must not be the same as the old. For a 10-char password, there are 30 topologies of the same length of Lev distance 1 for the attacker to target.
- **minlev=2**: new topology must be at least two changes away from the old. For a 10-char password, there are 405 possible 10-char topologies that are 2 Lev distance away (and more if the length is changed).
- This does not need audit mode to be enabled.

PathWell Enforcement Mode: maxuse

Enforcement mode option: **maxuse**

- Requires that Audit Mode is enabled, because it needs to track things over time.
- Sets the maximum number of passwords that can use any given topology.
- Typically set to 1 (each password must use a unique topology... until exhaustion/rollover and admins increment it to 2, etc).
- If **maxuse=1**, then an attacker who bruteforces a topology will score at most one plaintext.

PathWell Enforcement Mode: hintinfolevel

Enforcement mode option: **hintinfolevel**

- Out of the box, the user gets no details about why a too-weak password was refused.
- In the latest release of libpathwell we added multiple admin-tunable levels of user hinting, from nothing, to generic “try adding an uppercase letter as the fourth character and changing the last character to a number” to “You tried ‘**Kw#46_Ya**’, how about ‘**Kw#G46_Y7**’ instead?”

PathWell Enforcement Mode: hintinfolevel

Enforcement mode option: **hintinfolevel**

- Out of the box, the user gets no details about why a too-weak password was refused.
- In the latest release of libpathwell we added multiple admin-tunable levels of user hinting, from nothing, to generic “try adding an uppercase letter as the fourth character and changing the last character to a number” to “You tried ‘**Kw#46_Ya**’, how about ‘**Kw#G46_Y7**’ instead?”
  - Choose carefully! Higher hint levels not appropriate for all environments, such as if shoulder surfing is likely, etc.
  - These are supported by the API, so other things besides PAM modules could make use of them.
- Note: the hint engine is only hooked up for blacklist violations so far.

Agenda

My Background

Classic Password Attacks and Defenses

Recent Password Cracking Trends

PathWell Concepts

PathWell Code

Demo?

Next Steps

Not A Demo (Canned Examples)

Example /etc/pam.d settings:

- Default:

```
password  required  pam_cracklib.so difok=2 minlen=8 dcredit=2 ocredit=2 retry=3
password  required  pam_unix.so try_first_pass use_authtok nullok sha512 shadow
password  optional  pam_permit.so
```

- Audit mode adds:

```
password  optional  pam_pathwell.so mode=monitor use_authtok
```

- Blacklist mode adds:

```
password  required  pam_pathwell.so mode=enforce use_authtok blacklist
```

- Minlev mode adds:

```
password  required  pam_pathwell.so mode=enforce use_authtok minlev=2
```

- Maxuse mode adds:

```
password  required  pam_pathwell.so mode=enforce use_authtok maxuse=1
password  optional  pam_pathwell.so mode=monitor use_authtok
```

- Enabling the hint engine:

```
password  required  pam_pathwell.so mode=enforce use_authtok blacklist hintinfolevel=2
```

Full examples are included in README.PAM in the source code distribution.

Example Logs & Output: Success

- Successful password change user output:

```
patsy@marklar ~ $ passwd
Changing password for patsy.
Current password:
New password: Kw#46_Ya
Retype new password: Kw#46_Ya
passwd: password updated successfully
```

- Successful password change syslog message:

```
Apr 25 14:12:44 marklar passwd[27103]:
  pam_pathwell(passwd:chauthtok): Release='0.7.0'; Library='2:0:0';
  Module='0:3:0'; PamFlags='0x00002000'; Mode='enforce';
  User='patsy'; Status='pass'; Reason='Password accepted.';
```

Example Logs & Output: Minlev

- Failed (minlev=2 violation):

```
patsy@marklar ~ $ passwd
Changing password for patsy.
Current password: Kw#46_Ya
New password: Kw#47_Y4
Retype new password: Kw#47_Y4
pam_pathwell: The new password failed the minlev check.
passwd: Authentication token manipulation error
passwd: password unchanged
```

- Failed password change syslog messages:

```
Apr 25 14:14:16 marklar passwd[27121]:
  pam_pathwell(passwd:chauthtok): Release='0.7.0'; Library='2:0:0';
  Module='0:3:0'; PamFlags='0x00002000'; Mode='enforce';
  User='patsy'; Status='fail'; Reason='Password rejected.';
```

Example Logs & Output: Maxuse

- Failed (maxuse=1 violation, minlev not enabled):

```
patsy@marklar ~ $ passwd
Changing password for patsy.
Current password: Kw#46_Ya
New password: Le$57+Us
Retype new password: Le$57+Us
pam_pathwell: The new password failed the maxuse check.
passwd: Authentication token manipulation error
passwd: password unchanged
```

- Failed password change syslog messages:

```
Apr 25 14:21:18 marklar passwd[27455]:
  pam_pathwell(passwd:chauthtok): Release='0.7.0'; Library='2:0:0';
  Module='0:3:0'; PamFlags='0x00002000'; Mode='enforce';
  User='patsy'; Status='fail'; Reason='Password rejected.';
```

Example Logs & Output: Blacklist Hinting

- Blacklist failure with hintinfolevel=2:

```
patsy@marklar ~ $ passwd
Changing password for patsy.
Current password:
New password: April2017!
Retype new password: April2017!
pam_pathwell:
                                              ullllddd ds
                                                       |
  insert a digit ------------------------------------+

 This should produce the following topology: ullllddddds

passwd: Authentication token manipulation error
passwd: password unchanged
```

Example Logs & Output: Blacklist Hinting

- Blacklist failure with hintinfolevel=3:

```
patsy@marklar ~ $ passwd
Changing password for patsy.
Current password:
New password: April2017!
Retype new password: April2017!
pam_pathwell:
                                                 April2017!
                                                       |   |
replace with a lower case character ------------+   |
                                                    |
  insert a digit ----------------------------------+

 This should produce the following topology: ullllldlddsd

passwd: Authentication token manipulation error
passwd: password unchanged
```

(Reminder: hintinfolevels greater than 2 are not appropriate for most corporate environments.)

Example Logs & Output: Blacklist Hinting

- Blacklist failure with hintinfolevel=4:

```
patsy@marklar ~ $ passwd
Changing password for patsy.
Current password:
New password: April2017!
Retype new password: April2017!
pam_pathwell:
                          A pril2017!
                          ||
  replace with '7' -----+|
                         |
    insert a ',' --------+

 This should produce the following password: 7,pril2017!

passwd: Authentication token manipulation error
passwd: password unchanged
```

(Reminder: hintinfolevels greater than 2 are not appropriate for most corporate environments.)

Agenda

My Background

Classic Password Attacks and Defenses

Recent Password Cracking Trends

PathWell Concepts

PathWell Code

Demo

Next Steps

PathWell: Next Steps for the Project

- More hints!
  - Implement the hint engine for other modes – minlev, maxuse.

PathWell: Next Steps for the Project

- More hints!
  - Implement the hint engine for other modes – minlev, maxuse.
- More platforms!
  - We have an Active Directory version in the works.
    - Has some trade-offs for data storage, user interface, etc.
    - We do not use Windows in production, so don’t have a good environment for long-term testing/support.
    - Currently talking to some of our clients we do password audits for about being pilot sites.
    - Anybody want to volunteer?

PathWell: Next Steps for the Project

- More hints!
  - Implement the hint engine for other modes – minlev, maxuse.
- More platforms!
  - We have an Active Directory version in the works.
    - Has some trade-offs for data storage, user interface, etc.
    - We do not use Windows in production, so don’t have a good environment for long-term testing/support.
    - Currently talking to some of our clients we do password audits for about being pilot sites.
    - Anybody want to volunteer?
  - Other platforms that don’t have PAM support?
    - Very large websites, SSO platforms, LDAP servers, etc.?

PathWell: Next Steps for the Project

- More hints!
  - Implement the hint engine for other modes – minlev, maxuse.
- More platforms!
  - We have an Active Directory version in the works.
    - Has some trade-offs for data storage, user interface, etc.
    - We do not use Windows in production, so don't have a good environment for long-term testing/support.
    - Currently talking to some of our clients we do password audits for about being pilot sites.
    - Anybody want to volunteer?
  - Other platforms that don't have PAM support?
    - Very large websites, SSO platforms, LDAP servers, etc.?
- More enforcement options!
  - Regular expression-based blacklisting should be easy to add.

    `[Dd].?[Ee].?[Nn].?[Vv].?[Ee].?[Rr]`

PathWell: Next Steps for Attackers?

How will attackers – cracking tools, techniques – adjust and adapt to PathWell?

That's all folks

Questions?

Hank Leininger <hlein@korelogic.com>
5F6D DCC8 FF53 8093 EC39 127B 091E 7F7C E898 E86C

PathWell Project <pathwell-project@korelogic.com>
9CCF 2BA6 4444 E8AA 36D5 315B 2ECC 5A37 25B2 CC97

Thanks to:

Klayton Monroe	Sean Segreti
Shawn Wilson	Mick Wollman
BITSys	DARPA!
CMIYC Teams	Hashcat / JTR

Other Reading

- https://blog.korelogic.com/ ← has links to various other talks
- https://git.korelogic.com/libpathwell.git/ ← get the library & PAM module code
- @CrackMeIfYouCan on Twitter
- CMIYC contest sites; past years have teams' writeups. Start at http://contest.korelogic.com and follow links to each year.
- My coworker Rick Redman has given a number of talks about advanced password cracking techniques:
  - Passwords13: http://www.youtube.com/watch?v=5i_Im6JntPQ
  - ISSA: http://infosec-summit.issa-balt.org/html/2010_agenda.html

  Rick goes into detail about advanced cracking techniques, where to download various rules we've written for different tools & how to write your own.
- An interesting study about studying password selection: “On The Ecological Validity of a Password Study”: http://cups.cs.cmu.edu/soups/2013/proceedings/a13_Fahl.pdf