Beyond Precision and Recall: Understanding Uses (and Misuses) of Similarity Hashes in Binary Analysis

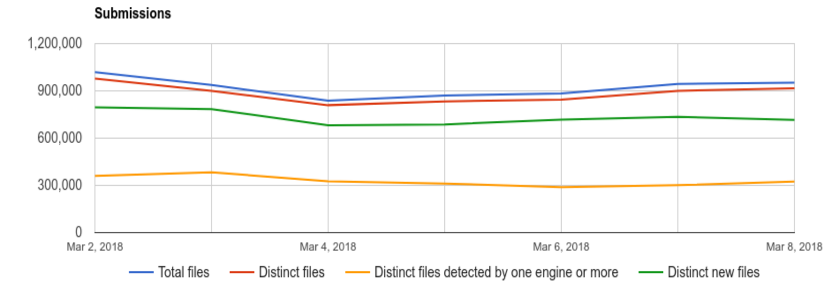
Fabio Pagani [1], Matteo Dell'Amico [2], Davide Balzarotti [1]

[1]EURECOM

[2]Symantec Research Labs

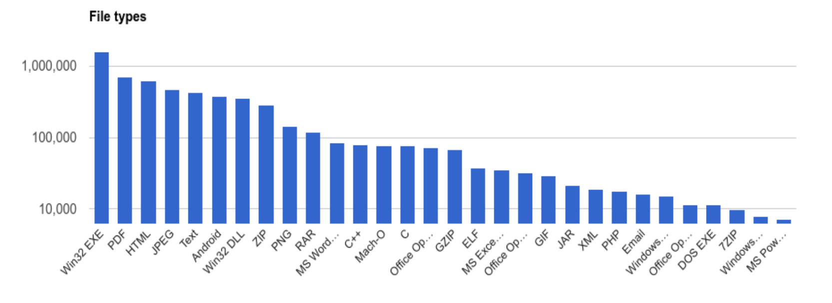
ACM Conference on Data and Application Security and Privacy 2018

The need to compare files is stronger than ever before



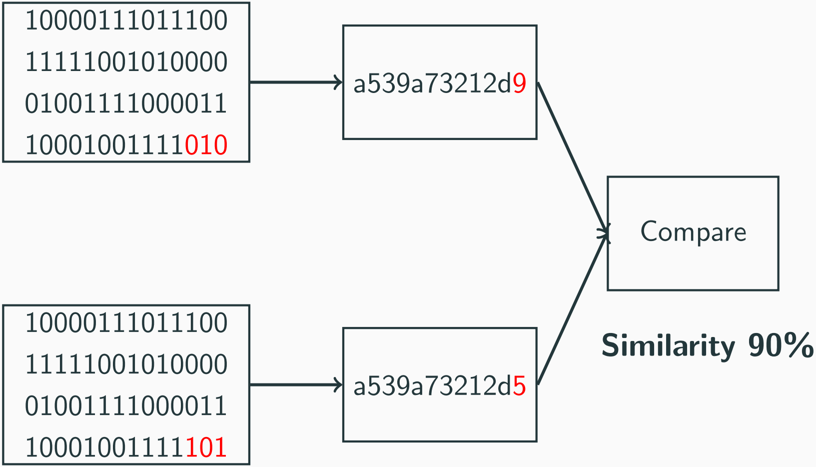
(Source: VirusTotal)

The need to compare files is stronger than ever before



(Source: VirusTotal)

Fuzzy Hash - Intro



Fuzzy Hash - Intro

- File Agnostic (*no* static analysis)
- Fast
- Hash comparison

Fuzzy Hash - Intro

File identification

MD5	5948462211d00c9cec468fd194e76c5f
SHA1	f152202769725a0f8bdfc104e17e3521dd1d05cc
SHA256	db16ba4b3029244b4d900648e443a3f0c71bef835987c44476d1f3817a1c629d
ssdeep	96:cexhkyqVGRlbk+xuM3cTd3pTdTKHOirQ4ypCWVK//u094MZ:cecyqcRlbkKdsRXEOirQ4gCP
File size	3.5 KB (3582 bytes)
File type	PDF
Magic literal	PDF document, version 1.4
TrID	Adobe Portable Document Format (100.0%)
Tags	cve-2007-5659 exploit js-embedded cve-2009-0927 invalid-xref pdf

Fuzzy Hash - Tools

- ssdeep (2006) and mrsh-v2 (2012)
  - Context Triggered Piecewise Hashing
  - Match if large part are in common (*chapter in a text file*)
- sdhash (2010)
  - Statistically Improbable Features - 64-byte strings
  - Match if such strings are in common (*phrases in a text file*)
- tlsh (2013)
  - N-Grams frequencies
  - Match if frequency is common (*similar words, same language*)

Motivation

TABLE V. PUBLISHED MEASUREMENTS VS TESTED MEASUREMENTS

MALWARE

	Published		Tested (Peak $FMeasure$)	
	Recall	Precision	Recall	Precision
BitShred[1]	.928	.932	.348	.914
TLSH[6]	.945	.935	.260	.583
sdhash[6]	.371	.995	.295	.910
ssdeep[6]	.312	.999	.337	.898
First Byte	na	na	.602	.872

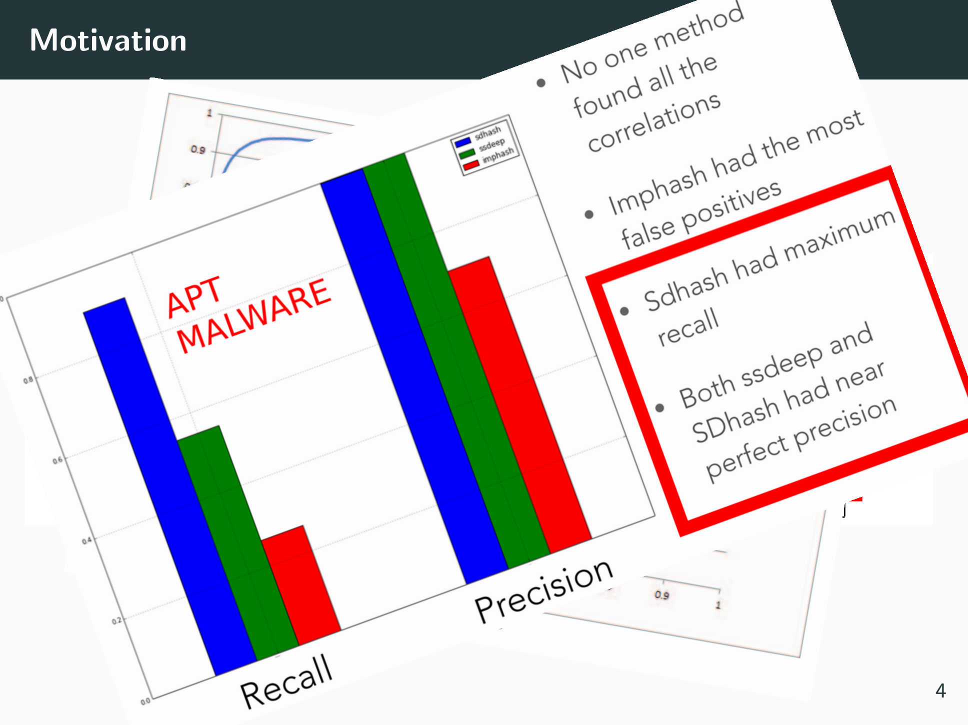
Motivation

TABLE ... NTS

MA

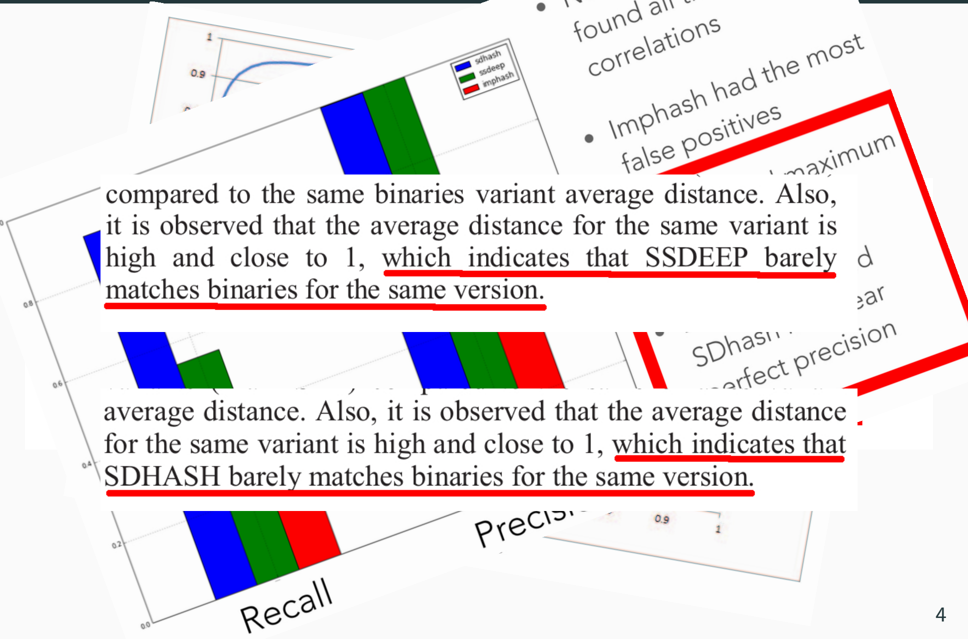
Fig. 1. ROC Curve

Motivation



No one method found all the correlations
Imphash had the most false positives
Sdhash had maximum recall
Both ssdeep and SDhash had near perfect precision
APT MALWARE
sdhash
ssdeep
imphash
Recall
Precision
0.8
0.6
0.4
0.2
0.0
1
0.9

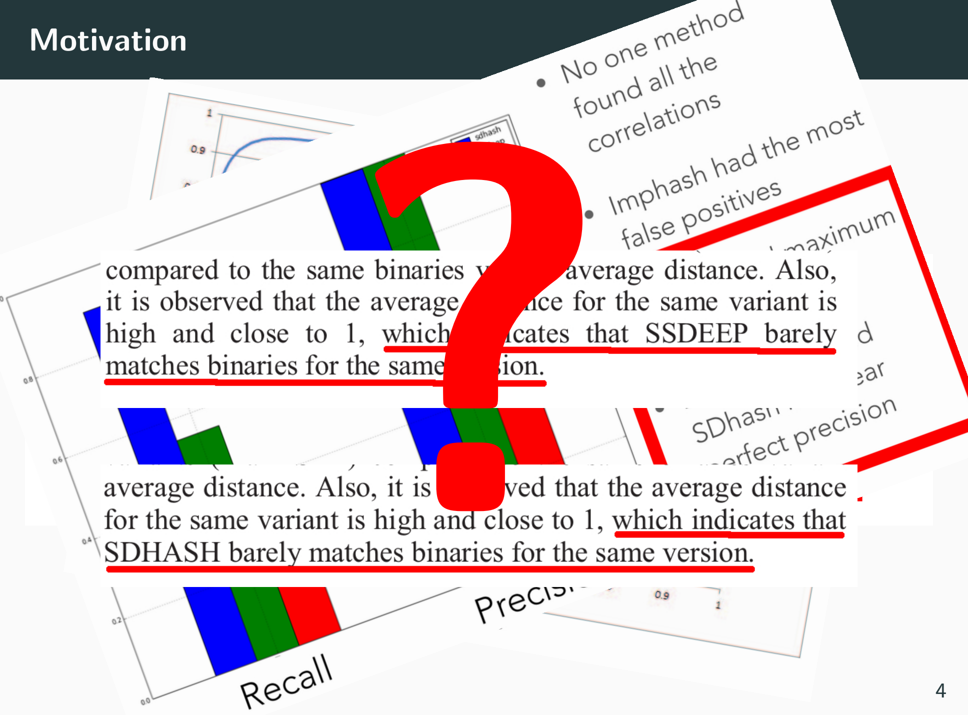
Motivation



compared to the same binaries variant average distance. Also, it is observed that the average distance for the same variant is high and close to 1, which indicates that SSDEEP barely matches binaries for the same version.

average distance. Also, it is observed that the average distance for the same variant is high and close to 1, which indicates that SDHASH barely matches binaries for the same version.

Motivation



No one method found all the correlations
Imphash had the most false positives
maximum
SDhash
perfect precision
compared to the same binaries w
average distance. Also,
it is observed that the average
nce for the same variant is
high and close to 1, which
icates that SSDEEP barely
matches binaries for the same
ion.
average distance. Also, it is
ved that the average distance
for the same variant is high and close to 1, which indicates that
SDHASH barely matches binaries for the same version.
sdhash
Recall
0.9
1

Binary Analysis Scenarios

- Scenario 1: library identification in statically linked binaries
- Scenario 2: applications compiled with different toolchains
- Scenario 3: different versions of the same application

Scenario 1: Library Identification

- 5 Linux libraries statically compiled in a C program
- Two test: entire object file, `.text` section only

Scenario 1: Library Identification

- 5 Linux libraries statically compiled in a C program
- Two test: entire object file, `.text` section only

Algorithm	Entire object			.text segment		
	TP%	FP%	Err%	TP%	FP%	Err%
`ssdeep`	0	0	-	0	0	-
`mrsh-v2`	11.7	0.5	-	7.7	0.2	-
`sdhash`	**12.8**	**0**	-	**24.4**	**0.1**	**53.9**
`tlsh`	0.4	0.1	-	0.2	0.1	**41.7**

Scenario 1: Library Identification

- 5 Linux libraries statically compiled in a C program
- Two test: entire object file, `.text` section only

Algorithm	Entire object			.text segment		
	TP%	FP%	Err%	TP%	FP%	Err%
`ssdeep`	0	0	-	0	0	-
`mrsh-v2`	11.7	0.5	-	7.7	0.2	-
`sdhash`	**12.8**	**0**	-	**24.4**	**0.1**	**53.9**
`tlsh`	0.4	0.1	-	0.2	0.1	**41.7**

Potential Problems

- Library Fragmentation (1MB binary vs 13KB object)
- Relocations

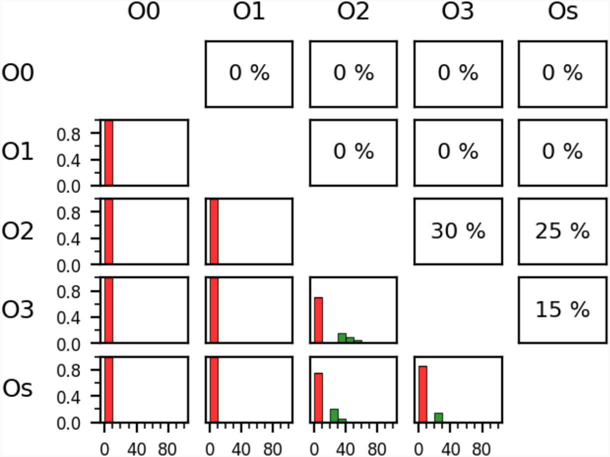
Scenario 1: Library Identification - Takeaways

- Matching statically linked libraries is a difficult task
- Major Problems:
  - Size binary $\gg$ size object file (impacts CTPH and `tlsh`)
  - Relocations ($\sim$ 10% of bytes changed) (impacts `sdhash`)

Scenario 2: Re-compilation

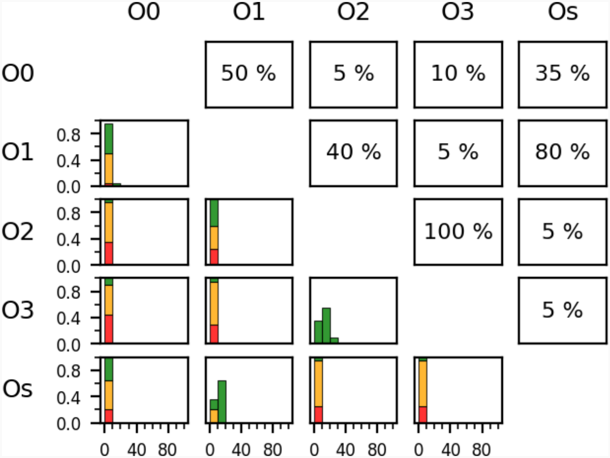
- Two dataset:
  - Small: `ls`, `sort`, `tail`, `base64`, `cp`
  - Large: `wireshark`, `ssh`, `sqlite3`, `openssl`, `httpd`
- 5 compiler flags (`O0..Os`)
- 4 compiler (`gcc-5`, `gcc-6`, `clang`, `icc`)

Scenario 2: Re-compilation - Flags Results



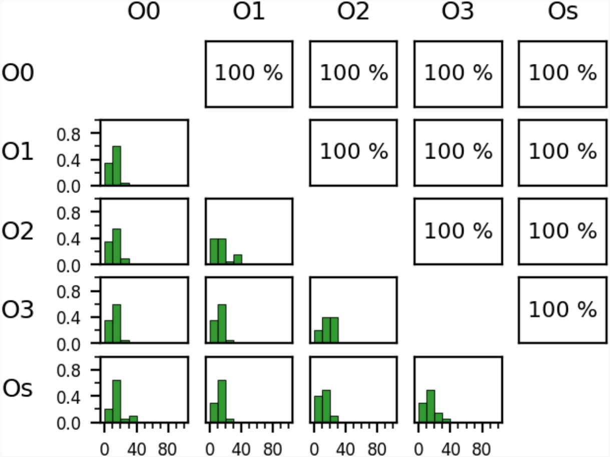
ssdeep (0% FP)

Scenario 2: Re-compilation - Flags Results



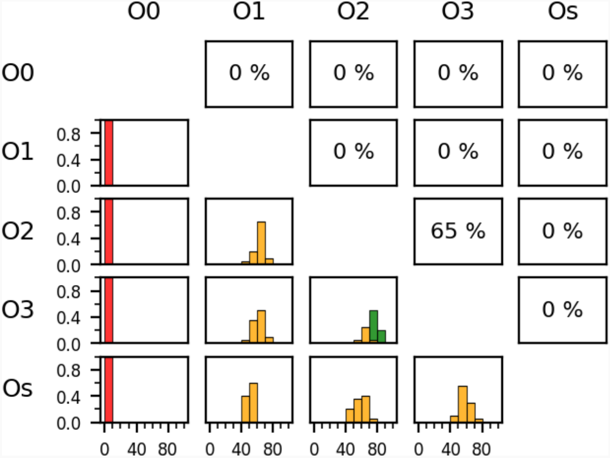
sdhash (0% FP) Small Dataset

Scenario 2: Re-compilation - Flags Results



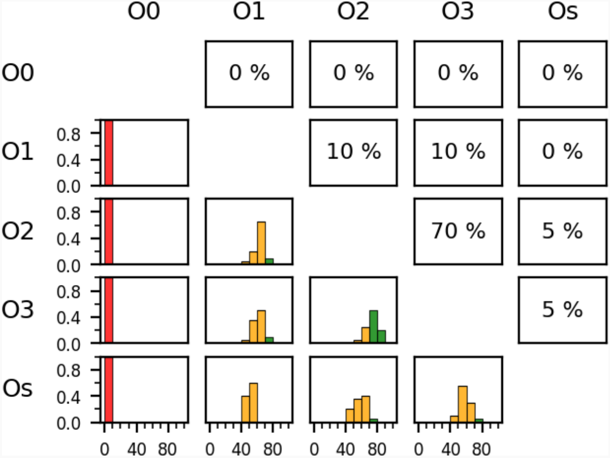
sdhash (0% FP) Large Dataset

Scenario 2: Re-compilation - Flags Results



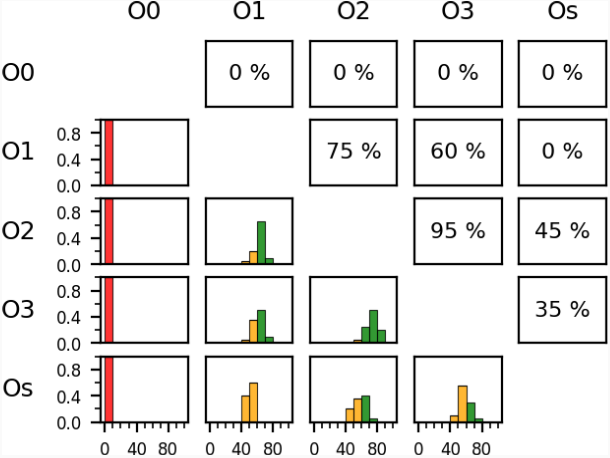
`tlsh` (0% FP)

Scenario 2: Re-compilation - Flags Results



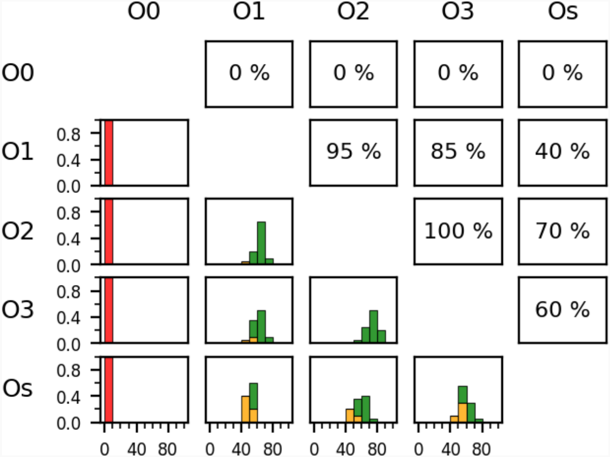
`tlsh` (1% FP)

Scenario 2: Re-compilation - Flags Results



`tlsh` (5% FP)

Scenario 2: Re-compilation - Flags Results



`tlsh` (10% FP)

Scenario 2: Re-compilation - Takeaways

- `sdhash` shines in this scenario
- `tlsh` is suitable as well, but has higher FP rate
- Programs compiled with O0 are the hardest to match

Scenario 3: Program Similarity

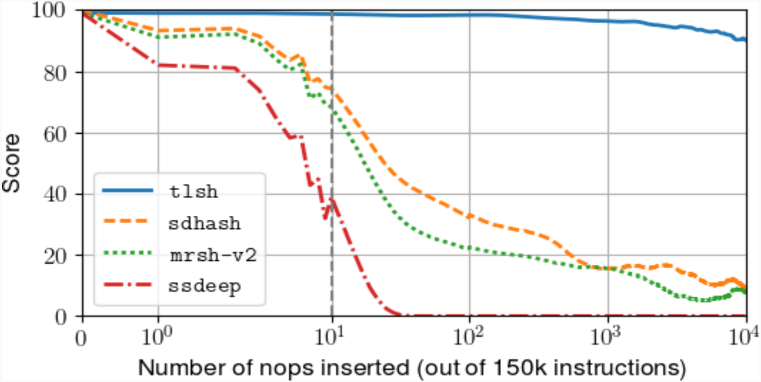
Keeping the toolchain constant we tested:

- Small differences at assembly level (benign)
- Small differences at source level (benign)
- Different version of the same application (malware)

Scenario 3: Program Similarity - Assembly Level

- Program under test: `ssh-client`
- Applied transformations:
  - random insertion of `NOPs`
  - random swapping of two instruction

Scenario 3: Program Similarity - Assembly Level



Scenario 3: Program Similarity - Assembly Level

We found cases where only 2 nops were enough to **zero** the similarity

What happened

1. some function are shifted down → intra-code references needs to be adjusted
2. `.text` section size increases → following sections are shifted down
3. references to this sections need to be adjusted (`.rodata`)
4. In total **8** sections changed

Scenario 3: Program Similarity - Source Level

- Program under test: `ssh-client`
- Applied modifications:
    - Different comparison *operator* ($< \rightarrow \leq$)
    - New *condition*
    - Change of a *constant*

Results are hard to predict because the compiler has *aggressive* optimization

Scenario 3: Program Similarity - Source Level

Change	ssdeep	mrsh-v2	tlsh	sdhash
Operator	**0 – 100**	**21 – 100**	**99 – 100**	**22 – 100**
Condition	**0 – 100**	**22 – 99**	**96 – 99**	**37 – 100**
Constant	**0 – 97**	**28 – 99**	**97 – 99**	**35 – 100**

Scenario 3: Program Similarity - Different version

- Malware under test:
  - `Grum` (Windows)
  - `Mirai` (Linux)
- Applied modifications:
  - New C&C domain (*real* and *long*)
  - *Evasion*: real anti-analysis tricks to detect debugger and virtualization
  - New *functionality*: collect and send the list of user present in the system

Scenario 3: Program Similarity - Different version

Change	ssdeep		mrsh-v2		tlsh		sdhash	
	M	G	M	G	M	G	M	G
C&C domain (real)	**0**	**0**	**97**	**10**	**99**	**88**	**98**	**24**
C&C domain (long)	**0**	**0**	**44**	**13**	**94**	**84**	**72**	**22**
Evasion	**0**	**0**	**17**	**0**	**93**	**87**	**16**	**34**
Functionality	**0**	**0**	**9**	**0**	**88**	**84**	**22**	**7**

"M" and "G" stand respectively for "Mirai" and "Grum"

Scenario 3: Program Similarity - Takeaways

- `tlsh` shines in this scenario
- If binary sections are moved expect a **low** similarity

Conclusion

Today we sheds light on the behavior of fuzzy hashing.

- `CTPH` $\rightarrow$ falls short in most tasks (**used** by VirusTotal)
- `sdhash` $\rightarrow$ same program compiled in different ways
- `tlsh` $\rightarrow$ different version of the same program