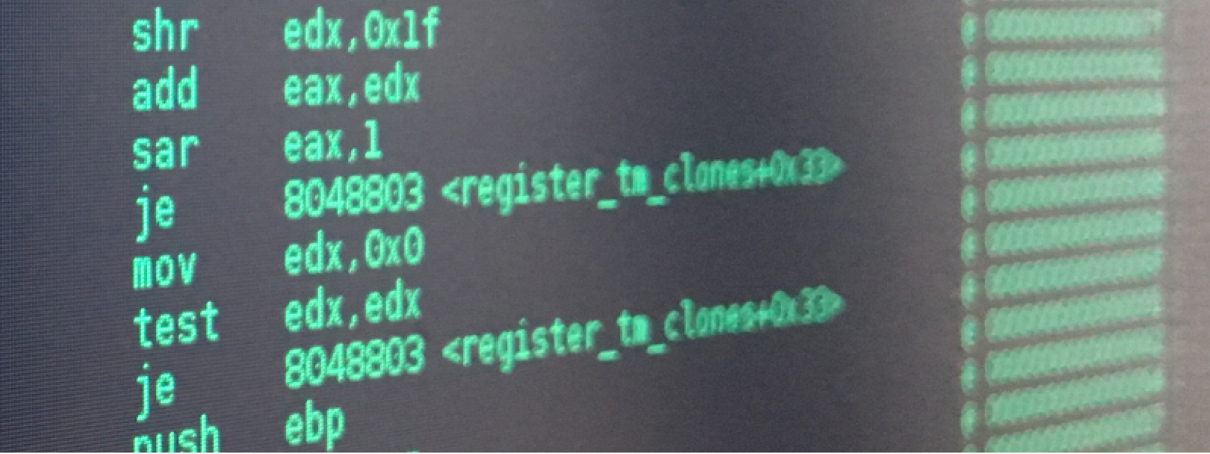


Darmstadt University of Applied Sciences, da/sec Security Group

Towards Exact and Inexact Approximate Matching of Executable Binaries

DFRWS-EU 2019, Oslo, Norway

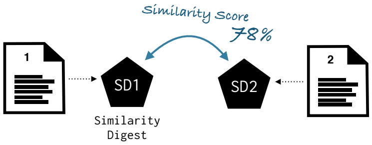
Lorenz Liebler, Harald Baier

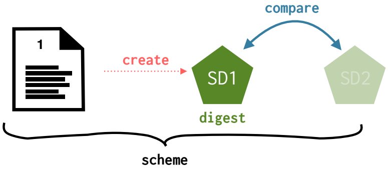
da/sec
BIOMETRICS AND INTERNET-SECURITY
RESEARCH GROUP

CRISP
Center for Research
in Security and Privacy

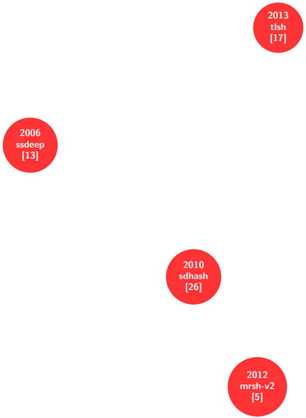
General

- a.k.a Approximate Matching:
  is a *similarity preserving hash function*
- in contrary to cryptographic hash functions
  → determines similarity of two files
- introduced more than a decade ago
  → deal with spam
  → forensic challenges
- simple to implement, few computational resources





Overview and History



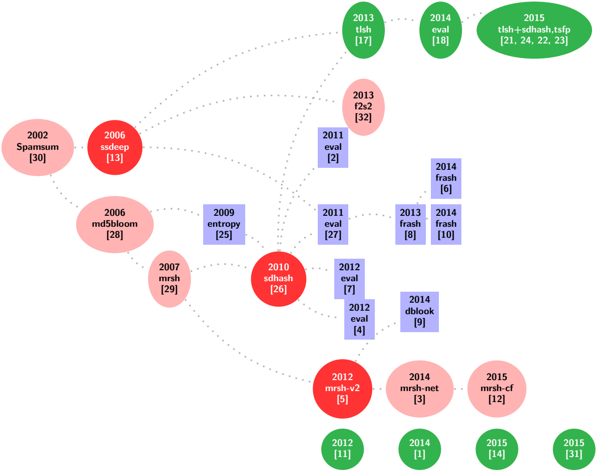
ssdeep [13]: Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. Digital investigation, 3:91–97, 2006

sdhash [26]: Vassil Roussev. Data fingerprinting with similarity digests. In IFIP International Conference on Digital Forensics, pages 207–226. Springer, 2010

mrsh-v2 [5]: Frank Breitinger and Harald Baier. Similarity preserving hashing: Eligible properties and a new algorithm mrsh-v2. In International conference on digital forensics and cyber crime, pages 167–182. Springer, 2012

tlsh [17]: Jonathan Oliver, Chun Cheng, and Yanggui Chen. TLSH–A Locality Sensitive Hash. In Cybercrime and Trustworthy Computing Workshop (CTC). 2013

Overview and History



→ especially in the field of malware or binary analysis

Schemes

Internal implementations differ heavily

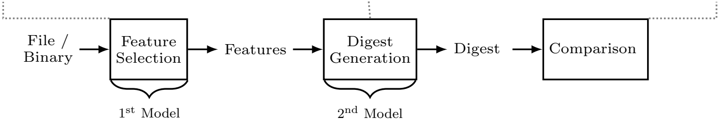
- Context-Triggered Piecewise Hashing (`ssdeep`, `mrsh-v2`)
- Statistically Improbable Features (`sdhash`)
- N-Grams (`tlsh`)

Simplified overview similar to Ren, Liwei [21] (DFRWS EU 2015):

ssdeep: chunks of sequences (splitted string)
mrsh-v2: chunks of sequences (extracted by PRF)
sdhash: bag of 64-byte blocks (selected by entropy)
tlsh: bag of triplets (selected from all 5-grams)

ssdeep: mapped chunks into 80 byte digest
mrsh-v2: chunks hashed into Bloom filter
sdhash: blocks mapped into Bloom filter
tlsh: mappend into 32 byte container

ssdeep: Levenshtein distance (0-100)
mrsh-v2: Hamming distance (0-100)
sdhash: Hamming distance (0-100)
tlsh: Distance score (0-1000+)

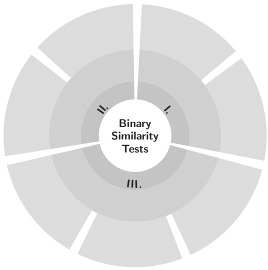


Binary Analysis in Academia (Pagani et al. [19])

Fabio Pagani, Matteo Dell'Amico, and Davide Balzarotti. Beyond precision and recall: Understanding uses (and misuses) of similarity hashes in binary analysis.
In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 354–365. ACM, 2018

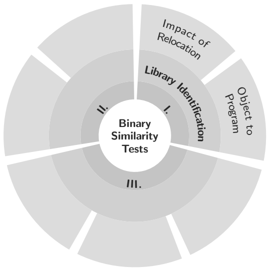
- no academic consensus about usefulness
- different evaluation datasets lead to different conclusions for same approaches (e.g., `ssdeep`)
- avoid: yet another large scale experiment
- inspect reasons for results (*why*; not *if*):
  $\rightarrow$ four different schemes
  $\rightarrow$ in **three binary analysis case studies**

Scenarios - Pagani et al. [19]



Scenarios - Pagani et al. [19]

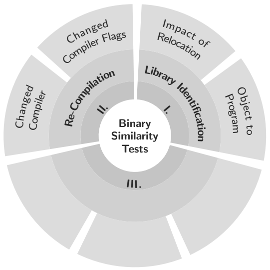
- I **Library Identification:** detecting embedded object files inside a binary
  - I.1 Object-to-Program Comparison;
    → whole executable (.o)
    → .text segment only.
  - I.2 Impact of Relocation considers
    → relocations performed by linker / dynamic loader
    → original and relocated object file / final executable



Scenarios - Pagani et al. [19]

II **Re-Compilation:** detection of the same program after Re-Compilation

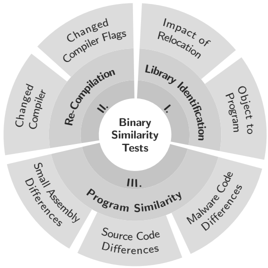
- II.1 Effect of Compiler Flags (same compiler; i.e., O0, O1, O2, O3, Os)
- II.2 Effect of Different Compilers



Scenarios - Pagani et al. [19]

III **Program Similarity:** three tests which consider adaptations to the underlying code

- III.1 Small Assembly Differences:
  - → randomly inserts an increasing amount of NOPs
  - → increasing amount of instructions are swapped
- III.2 Minor Source Code Modifications:
  - → Different Comparison Operator
  - → New Condition
  - → Change a constant value
- III.3 Malware Code Modifications (Mirai, Grum):
  - → C2 Domain Adaptation
  - → Evasion and New Functionality



Fuzzy Hashing - Pagani et al. [19] Summary

- the distinction between **data and code** is of crucial importance
- even **small changes** on the (source) code / additional insertions → influence the overall binary and code structure in a broad way
  → especially has a **great impact on CTPH**-based approaches
  → similarity is not just a consequence of the size of the change
- Summarized, **sdhash and tlsh clearly outperformed CTPH**-based schemes.
  → Each of both have their strengths and weaknesses in different disciplines.
- CTPH (**ssdeep**) - the de-facto industry standard — is **not very well suited** to binary analysis in general

RQs

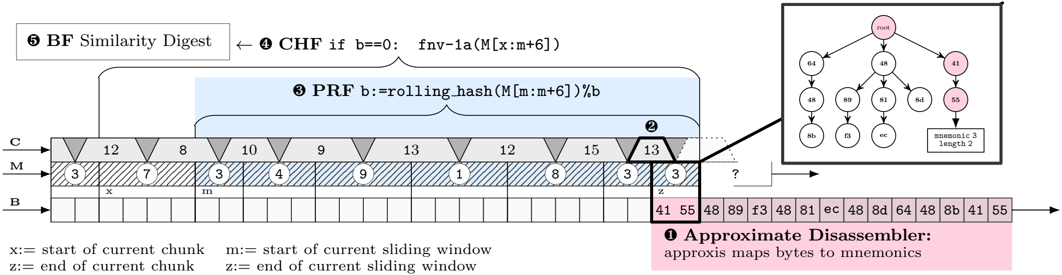
mrsh-mem
→ approxis x86/x64 instruction carver
→ interfaced with mrsh-v2
→ bulk extraction / identification of code

- What impact has the discrimination of code and data?
- Could we utilize an additional layer of approximate disassemble?
- What is the actual improvement in the case of a CTPH-based approaches?
  Could we improve the performance only by refining the feature selection?

mrsh-mem [16]: Lorenz Liebler and Frank Breitinger. mrsh-mem: Approximate matching on raw memory dumps. In International Conference on IT Security Incident Management and IT Forensics, pages 47–64. IEEE, 2018

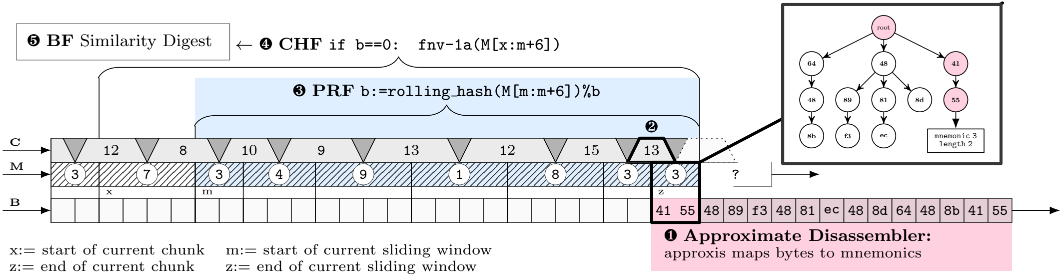
approxis [15]: Lorenz Liebler and Harald Baier. Approxis: A fast, robust, lightweight and approximate disassembler considered in the field of memory forensics. In International Conference on Digital Forensics and Cyber Crime, pages 158–172. Springer, 2017

apx-bin: Utilizing mrsh-mem



x:= start of current chunk
z:= end of current chunk
m:= start of current sliding window
z:= end of current sliding window

apx-bin: Utilizing mrsh-mem



Original mrsh-mem approach:

- extraction of **Code**-related fragments only
  → now also **Data**-related
- no **scoring** of the different buffers
  → **scoring** of different streams; use Mnemonic- (M) and Byte-Stream (B)

Naive Approach

Prove impact of data- or code-related features by adapted score-model:

1. Extract chunks of code and data by parametrization via τ_{min} and τ_{max};
   number of all chunks (from both buffers) defined as z
2. Multi-layered extraction - processing mapped buffer of mnemonics (M) and its byte representation (B)

$$sim_{pre} = min\left(\frac{\left(\sum_{i=1}^{y} f_c(b_i)\right) \cdot 1.5 + \sum_{i=1}^{y} f_c(m_i)}{z}, 100\right),$$

$$\text{where } f_c(x) = \begin{cases} 1, & \text{if } \tau_{min} \leq x \leq \tau_{max} \\ 0, & \text{else} \end{cases} . \quad (1)$$

Naive Approach

Prove impact of data- or code-related features by adapted score-model:

1. Extract chunks of code and data by parametrization via τ_{min} and τ_{max};
   number of all chunks (from both buffers) defined as z
2. Multi-layered extraction - processing mapped buffer of mnemonics (M) and its byte representation (B)

τ-values filter each extracted chunk:

τ_{max}-τ_{min}	Meaning
100-0	All chunks
100-80	Code chunks
20-0	Data chunks

$$sim_{pre} = min\left(\frac{\left(\sum_{i=1}^{y} f_c(\ b_i\ )\right) \cdot\ 1.5\ + \sum_{i=1}^{y} f_c(\ m_i\ )}{z}, 100\right),$$

$$\text{where } f_c(x) = \begin{cases} 1, & \text{if } \tau_{min} \leq x \leq \tau_{max} \\ 0, & \text{else} \end{cases}. \qquad (1)$$

Naive Approach

Prove impact of data- or code-related features by adapted score-model:

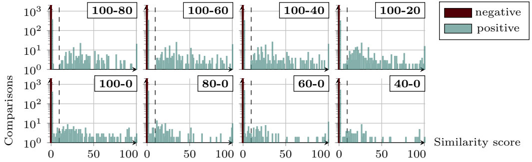
1. Extract chunks of code and data by parametrization via τ_{min} and τ_{max};
   number of all chunks (from both buffers) defined as z
2. Multi-layered extraction - processing mapped buffer of mnemonics (M) and its byte representation (B)

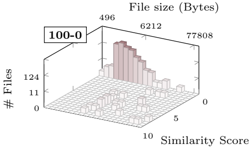
$$sim_{pre} = min\left(\frac{\left(\sum_{i=1}^{y} f_c(b_i)\right)\cdot 1.5 + \sum_{i=1}^{y} f_c(m_i)}{z}, 100\right),$$

$$\text{where } f_c(x) = \begin{cases} 1, & \text{if } \tau_{min} \leq x \leq \tau_{max} \\ 0, & \text{else} \end{cases} . \quad (1)$$

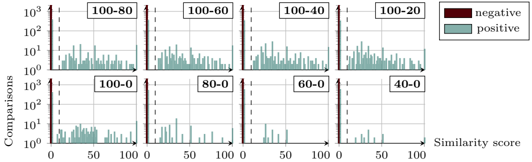
Scenario I. Library Identification

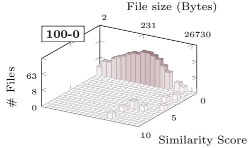
Object-to-Program (Whole Object File)





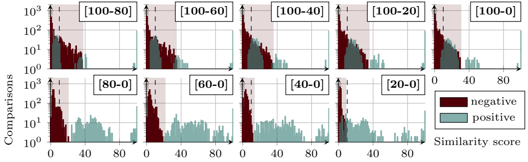
Object-to-Program (Text Section)

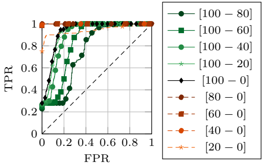




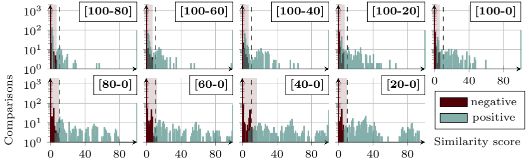
Scenario II. Recompilation

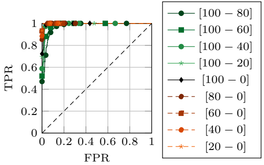
Optimization flags





Compiler Variation





Conclusion

- prove of **ambivalence** of the different **use cases**
  - **matching code**: primarily detecting used libraries
  - **matching data**: different compilers / configurations
- small constant **data** fragments **versus** large amount of **code** per binary
- a match on the **byte level** should be considered as more meaningful
- we stick to the extraction of sequences

Approach

Results and observations from the pre-evaluation have been used:

- τ now defines center bounds of selected chunks (ignore vague chunks)

$$f_d(x) = \begin{cases} 1, & \text{if} \quad 0 \leq x \leq \tau_{min} \\ 0, & \text{else} \end{cases}$$

$$f_c(x) = \begin{cases} 1, & \text{if} \quad 100 \geq x \geq \tau_{max} \\ 0, & \text{else} \end{cases}$$

- Score model

$$sim_{bm} = \frac{\gamma_d + \gamma_c}{2} \text{ ,where}$$

$$\gamma_d = min\left(\frac{\left(\sum_{i=0}^{y-1} f_d(b_i) \cdot 2 + \sum_{i=0}^{z-1} f_d(m_i)\right) \cdot 1.5}{2 \cdot \sum_{i=0}^{n} f_d(c_i)}, 0.99\right),$$

$$\gamma_c = min\left(\frac{\sum_{i=0}^{y-1} f_c(b_i) \cdot 2 + \sum_{i=0}^{z-1} f_c(m_i)}{2 \cdot \sum_{i=0}^{n-1} f_c(c_i)}, 0.99\right).$$

sequence of extracted chunks represented by their specific *code coverage*:
$\langle c_0, c_1, \ldots, c_{n-1} \rangle$

hits are defined by their values of code coverage for matching byte chunks
$\langle b_0, b_1, \ldots, b_{y-1} \rangle)$
and for matching mnemonic chunks
$\langle m_0, m_1, \ldots, m_{z-1} \rangle$

match either as a sequence of mnemonics and bytes, or as a sequence of mnemonics only
$y \leq z$

Approach

Results and observations from the pre-evaluation have been used:

- τ now defines center bounds of selected chunks (ignore vague chunks)

$$f_d(x) = \begin{cases} 1, & \text{if } 0 \leq x \leq \tau_{min} \\ 0, & \text{else} \end{cases}$$

$$f_c(x) = \begin{cases} 1, & \text{if } 100 \geq x \geq \tau_{max} \\ 0, & \text{else} \end{cases}$$

- Score model

$$sim_{bm} = \frac{\gamma_d + \gamma_c}{2} \text{ ,where}$$

$$\gamma_d = min\left(\frac{\left(\sum_{i=0}^{y-1} f_d(b_i) \cdot 2 + \sum_{i=0}^{z-1} f_d(m_i)\right) \cdot 1.5}{2 \cdot \sum_{i=0}^{n} f_d(c_i)}, 0.99\right),$$

$$\gamma_c = min\left(\frac{\sum_{i=0}^{y-1} f_c(b_i) \cdot 2 + \sum_{i=0}^{z-1} f_c(m_i)}{2 \cdot \sum_{i=0}^{n-1} f_c(c_i)}, 0.99\right).$$

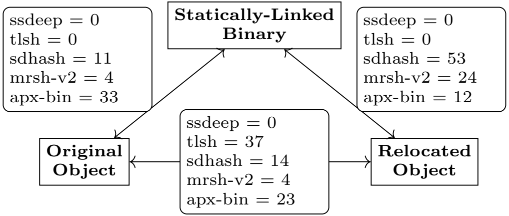
τ-values filter code **and** data chunks:

Range	Meaning
100 - τ_{max}	Code chunks
τ_{min} - 0	Data chunks

Scenario I - 1. Object-to-Program Comparison

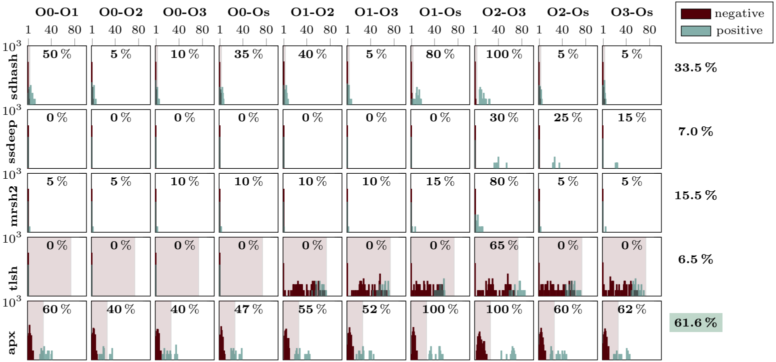
Alg.	.o TPR %	.o FPR %	.text TPR %	.text FPR %	.text Err %
ssdeep	0	0	0	0	0
mrsh-v2	11.7	0.5	7.7	0.2	0
sdhash	12.8	0	24.4	0.1	53.9
tlsh	0.4	0.1	0.2	0.1	41.7
apx-bin	**48.9**	**0.0**	**44.1**	**0.0**	**0**

Scenario I - 2. Impact of Relocation



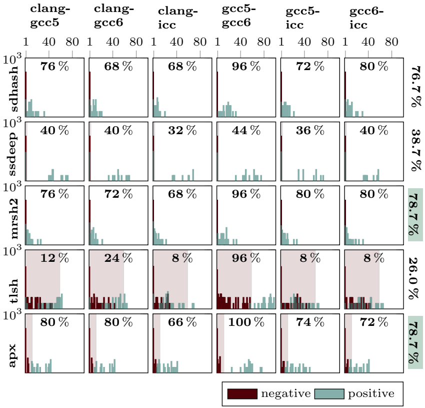
Scheme	Average Score (%)
`ssdeep`	0.0
`tlsh`	12.3
`sdhash`	26.0
`mrsh-v2`	10.67
`apx-bin`	22.67

Scenario II Re-Compilation - 1. Optimization Flags



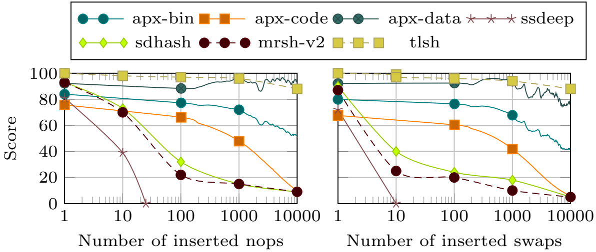
→ no comparisons of same configuration (same flags)
→ zero false positives
→ coloured area: highest score of a false match

Scenario II Re-Compilation - 2. Different Compilers



→ no comparisons of same configuration (same compiler)
→ zero false positives
→ coloured area: highest score of a false match

Scenario III Program Similarity - 1. Small Assembly Differences



Scenario III Program Similarity

III.2 Minor Source Code Modifications

Change	ssdeep	mrsh-v2	tlsh	sdhash	apx
Operator	0-100	21-100	99-100	22-100	76-99
Condition	0-100	22-99	96-99	37-100	83-99
Constant	0-97	28-99	97-99	35-100	81-99

→ score ranges (min-max)

III.3 Minor Source Code Modifications on Malware

Change	ssdeep		mrsh-v2		tlsh		sdhash		apx	
	M	G	M	G	M	G	M	G	M	G
C2 domain (r)	0	0	97	10	99	88	98	24	78	99
C2 domain (l)	0	0	44	13	94	84	72	22	76	86
Evasion	0	0	17	0	93	87	16	34	49	99
Functionality	0	0	9	0	88	84	22	7	34	79

→ real (r) or long (l) domain
→ evasion: anti-debugger, anti-VM techniques
→ functionality: enumerate users on infected system

Conclusion

- reassessed previous research
- demonstrate relevance of feature selection for matching-success
  $\rightarrow$ apx-bin still relies on CTPH
- outperform several of the existing (also non-CTPH) approaches
- stable scores in all scenarios

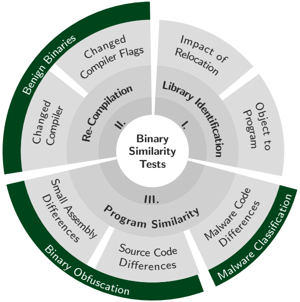
Current Work 1/2

- extend approxis-engine:
  $\rightarrow$ extend carver / approxis-engine
  $\rightarrow$ interface with other schemes (non-CTPH)
- extend scenarios / break approaches:
  $\rightarrow$ gather different schemes and evaluation data
  $\rightarrow$ create online-repository to publish results
  $\rightarrow$ please feel free to contact me

Current Work 2/2

- benign binaries for different architectures, compilers and languages
- malicious binaries curated by Malpedia [20]
- create a controlled set of obfuscated binaries with the help of Obfuscator-LLVM[a]

jmpcond_100pct	Conversion of conditional jumps
bcf_5_10pct	Bogus Control Flow; across 5 runs
bcf_1_100pct	Bogus Control Flow; across 1 run
noop_15_10pct	max. 15 multi-byte NOPs per Basic Block
noop_X_Ypct	max. X multi-byte NOPs per Basic Block.
fla	Control-Flow-Flattening
split_5	Splitting of Basic Blocks; max. 5 splits
sub_2	Substitution of commands with 2 runs
dead	Insertion of Dead-Code
split_dead	Splitting of Basic-Blocks; max 5 splits; insertion of Dead-Code
	...



[a] https://github.com/obfuscator-llvm/obfuscator/wiki

Thank you.

`lorenz.liebler@h-da.de`
`https://dasec.h-da.de/staff/lorenz-liebler/`

Bibliography I

[1] Ahmad Azab, Robert Layton, Mamoun Alazab, and Jonathan Oliver. Mining malware to detect variants. In *2014 Fifth Cybercrime and Trustworthy Computing Conference*, pages 44–53. IEEE, 2014.

[2] Harald Baier and Frank Breitinger. Security aspects of piecewise hashing in computer forensics. In *2011 Sixth International Conference on IT Security Incident Management and IT Forensics*, pages 21–36. IEEE, 2011.

[3] Frank Breitinger and Ibrahim Baggili. File detection on network traffic using approximate matching. 2014.

[4] Frank Breitinger and Harald Baier. Properties of a similarity preserving hash function and their realization in sdhash. In *2012 Information Security for South Africa*, pages 1–8. IEEE, 2012.

[5] Frank Breitinger and Harald Baier. Similarity preserving hashing: Eligible properties and a new algorithm mrsh-v2. In *International conference on digital forensics and cyber crime*, pages 167–182. Springer, 2012.

[6] Frank Breitinger and Vassil Roussev. Automated evaluation of approximate matching algorithms on real data. *Digital Investigation*, 11: S10–S17, 2014.

[7] Frank Breitinger, Harald Baier, and Jesse Beckingham. Security and implementation analysis of the similarity digest sdhash. In *First international baltic conference on network security & forensics (nesefo)*, 2012.

[8] Frank Breitinger, Georgios Stivaktakis, and Harald Baier. Frash: A framework to test algorithms of similarity hashing. *Digital Investigation*, 10: S50–S58, 2013.

[9] Frank Breitinger, Harald Baier, and Douglas White. On the database lookup problem of approximate matching. *Digital Investigation*, 11:S1–S9, 2014.

[10] Frank Breitinger, Georgios Stivaktakis, and Vassil Roussev. Evaluating detection error trade-offs for bytewise approximate matching algorithms. *Digital Investigation*, 11(2):81–89, 2014.

Bibliography II

[11] David French and William Casey. 2 fuzzy hashing techniques in applied malware analysis. *Results of SEI Line-Funded Exploratory New Starts Projects*, page 2, 2012.

[12] Vikas Gupta and Frank Breitinger. How cuckoo filter can improve existing approximate matching techniques. In *International conference on digital forensics and cyber crime*, pages 39–52. Springer, 2015.

[13] Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital investigation*, 3:91–97, 2006.

[14] Yuping Li, Sathya Chandran Sundaramurthy, Alexandru G Bardas, Xinming Ou, Doina Caragea, Xin Hu, and Jiyong Jang. Experimental study of fuzzy hashing in malware clustering analysis. In *8th Workshop on Cyber Security Experimentation and Test ({CSET} 15)*, 2015.

[15] Lorenz Liebler and Harald Baier. Approxis: A fast, robust, lightweight and approximate disassembler considered in the field of memory forensics. In *International Conference on Digital Forensics and Cyber Crime*, pages 158–172. Springer, 2017.

[16] Lorenz Liebler and Frank Breitinger. mrsh-mem: Approximate matching on raw memory dumps. In *International Conference on IT Security Incident Management and IT Forensics*, pages 47–64. IEEE, 2018.

[17] Jonathan Oliver, Chun Cheng, and Yanggui Chen. Tlsh a locality sensitive hash. pages 7–13, 2013.

[18] Jonathan Oliver, Scott Forman, and Chun Cheng. Using randomization to attack similarity digests. In *International Conference on Applications and Techniques in Information Security*, pages 199–210. Springer, 2014.

[19] Fabio Pagani, Matteo Dell'Amico, and Davide Balzarotti. Beyond precision and recall: Understanding uses (and misuses) of similarity hashes in binary analysis. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 354–365. ACM, 2018.

[20] Daniel Plohmann, Martin Clauss, Steffen Enders, and Elmar Padilla. Malpedia: A Collaborative Effort to Inventorize the Malware Landscape. *The Journal on Cybercrime & Digital Investigations*, 3(1), 2018. URL `https://journal.cecyf.fr/ojs/index.php/cybin/article/view/17`.

[21] Liwei Ren. A theoretic framework for evaluating similarity digesting tools, 03 2015. URL `https://www.dfrws.org/file/127/download?token=5YOUdHpY`. visited on 2019-01-20.

Bibliography III

[22] Liwei Ren. Binary similarity : Theory, algorithms and tool evaluation, 09 2015.

[23] Liwei Ren. Bytewise approximate match: Theory, algorithms and applications, 09 2015.

[24] Liwei Ren and Ray Cheng. Bytewise approximate matching, searching and clustering, 08 2015.

[25] Vassil Roussev. Building a better similarity trap with statistically improbable features. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2009.

[26] Vassil Roussev. Data fingerprinting with similarity digests. In *IFIP International Conference on Digital Forensics*, pages 207–226. Springer, 2010.

[27] Vassil Roussev. An evaluation of forensic similarity hashes. *digital investigation*, 8:S34–S41, 2011.

[28] Vassil Roussev, Yixin Chen, Timothy Bourg, and Golden G Richard III. md5bloom: Forensic filesystem hashing revisited. *digital investigation*, 3:82–90, 2006.

[29] Vassil Roussev, Golden G Richard III, and Lodovico Marziale. Multi-resolution similarity hashing. *digital investigation*, 4:105–113, 2007.

[30] Andrew Tridgell. Spamsum readme, 2002. URL `https://www.samba.org/ftp/unpacked/junkcode/spamsum/README`.

[31] Jason Upchurch and Xiaobo Zhou. Variant: a malware similarity testing framework. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 31–39. IEEE, 2015.

[32] Christian Winter, Markus Schneider, and York Yannikos. F2s2: Fast forensic similarity search through indexing piecewise hash signatures. *Digital Investigation*, 10(4):361–371, 2013.