Malware Lineage in the Wild

Irfan Ul Haq*, Sergio Chica*, Somesh Jha+, and Juan Caballero*

*IMDEA Software Institute, Spain
+University of Wisconsin, USA

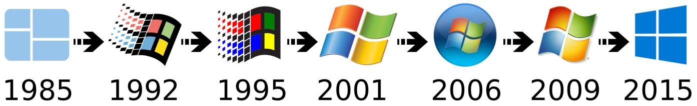
presented at IMDEA Siminar Series (July 18, 2017)

Under Submission

Lineage

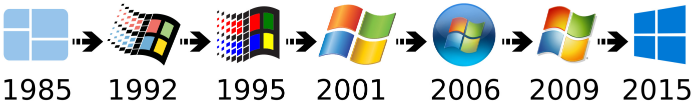
Lineage

- Software evolution over time
  - New features
  - Bug fixes
  - ...



Lineage

- Software evolution over time
  - New features
  - Bug fixes
  - ...



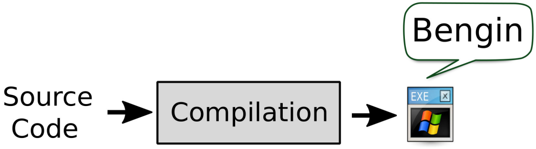
- Malware evolves similarly

Background and Motivation

- Unknown versions

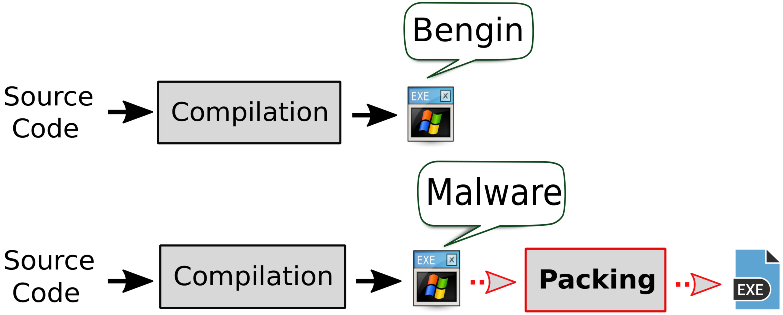
Background and Motivation

- Unknown versions
- Development



Background and Motivation

- Unknown versions
- Development



Packing

Challenges

- Unknown versions

Challenges

- Unknown versions
- Unpacking

Challenges

- Unknown versions
- Unpacking
- Disassembly

Challenges

- Unknown versions
- Unpacking
- Disassembly
- Granularity

Challenges

- Unknown versions
- Unpacking
- Disassembly
- Granularity
- Unknown development model

Challenges

- Unknown versions
- Unpacking
- Disassembly
- Granularity
- Unknown development model
- Incomplete data

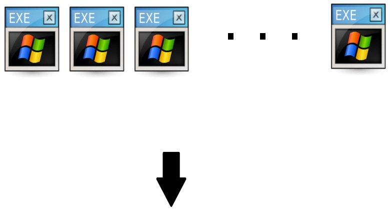
Why Lineage?

Malware analysis applications

- Triage
- Labeling
- Threat Intelligence
- Author attribution
- ...

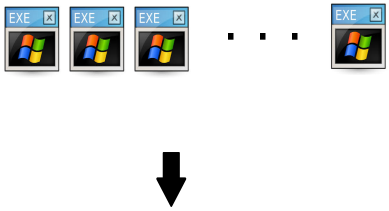
Problem Definition

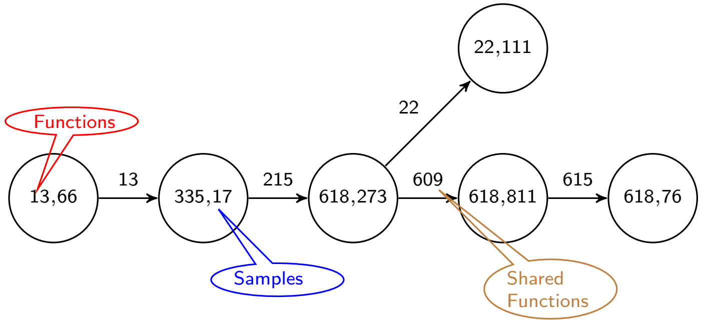
Problem Definition



$$\begin{tikzcd}
 & & & 22{,}111 & \\
13{,}66 \arrow[r, "13"] & 335{,}17 \arrow[r, "215"] & 618{,}273 \arrow[r, "609"] \arrow[ur, "22"] & 618{,}811 \arrow[r, "615"] & 618{,}76
\end{tikzcd}$$

Problem Definition





Approach → Overview

PhaseI → Code Recovery

1. Unpacking
2. Disassembly

Approach → Overview

PhaseI → Code Recovery

1. Unpacking
2. Disassembly

PhaseII → Lineage Inference

1. Identifying Version
2. Building a Lineage Tree
3. Adding Cross-Edges

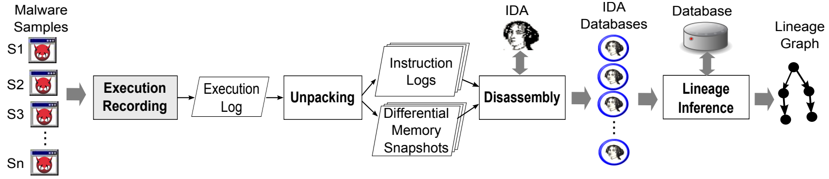
Approach → Overview

PhaseI → Code Recovery

1. Unpacking
2. Disassembly

PhaseII → Lineage Inference

1. Identifying Version
2. Building a Lineage Tree
3. Adding Cross-Edges



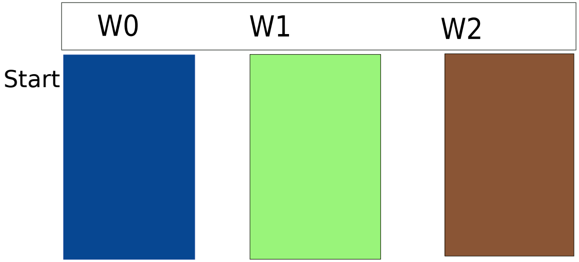
Code Recovery → Unpacking

Code Recovery → Unpacking

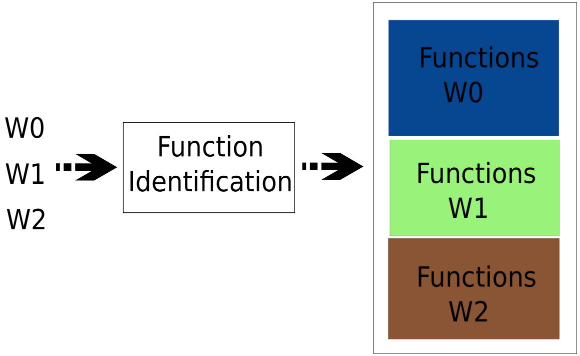
- Monitor each write and execute in memory

Code Recovery → Unpacking

– Monitor each write and execute in memory



Code Recovery → Disassembly



Lineage Algorithm→ Identifying Versions

- Same functions represent single version

Lineage Algorithm→ Identifying Versions

- Same functions represent single version
- BinDiff (Scalability?)

Lineage Algorithm→ Identifying Versions

- Same functions represent single version
- BinDiff (Scalability?)
- Function hash (RAW, SPP)

Lineage Algorithm→ Identifying Versions

- Same functions represent single version
- BinDiff (Scalability?)
- Function hash (RAW, SPP)

$$ProgramHash = md5sum(hash(f_1) + hash(f_2) + ... + hash(f_n))$$

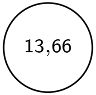
Lineage Algorithm→ Buiding Lineage Tree

	P13	**P22**	**P335**	**P618**	**P618_1**	**P618_2**
P13	-	12	13	13	13	13
P22	-	-	15	13	22	13
P335	-	-	-	214	215	214
P618	-	-	-	-	609	615
P618_1	-	-	-	-	-	609
P618_2	-	-	-	-	-	-

Lineage Algorithm$\rightarrow$ Buiding Lineage Tree

	P13	P22	P335	P618	P618_1	P618_2
P13	-	12	13	13	13	13
P22	-	-	15	13	22	13
P335	-	-	-	214	215	214
P618	-	-	-	-	609	615
P618_1	-	-	-	-	-	609
P618_2	-	-	-	-	-	-

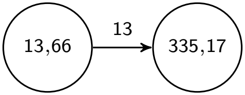
Identifying root $\rightarrow$ Lehman's 6*th* law of continuous growth.



Lineage Algorithm→ Buiding Lineage Tree

	P13	P22	P335	P618	P618_1	P618_2
P13	-	12	13	13	13	13
P22	-	-	15	13	22	13
P335	-	-	-	214	215	214
P618	-	-	-	-	609	615
P618_1	-	-	-	-	-	609
P618_2	-	-	-	-	-	-

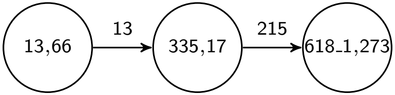
Selected $Nodes \rightarrow P335, P618, P618\_1, P618\_2$
Break tie: minimum number of functions



Lineage Algorithm→ Buiding Lineage Tree

	P13	P22	P335	P618	P618_1	P618_2
P13	-	12	13	13	13	13
P22	-	-	15	13	22	13
P335	-	-	-	214	215	214
P618	-	-	-	-	609	615
P618_1	-	-	-	-	-	609
P618_2	-	-	-	-	-	-

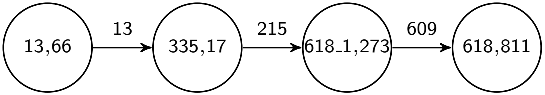
Selected *Node* → *P*618_1



Lineage Algorithm→ Buiding Lineage Tree

	P13	**P22**	**P335**	**P618**	**P618_1**	**P618_2**
P13	-	12	13	13	13	13
P22	-	-	15	13	22	13
P335	-	-	-	214	215	214
P618	-	-	-	-	609	615
P618_1	-	-	-	-	-	609
P618_2	-	-	-	-	-	-

Selected $Node \rightarrow P618$


$$\begin{tikzcd} 13,66 \arrow[r, "13"] & 335,17 \arrow[r, "215"] & 618\_1,273 \arrow[r, "609"] & 618,811 \end{tikzcd}$$

Lineage Algorithm→ Buiding Lineage Tree

	P13	P22	P335	P618	P618_1	P618_2
P13	-	12	13	13	13	13
P22	-	-	15	13	22	13
P335	-	-	-	214	215	214
P618	-	-	-	-	609	615
P618_1	-	-	-	-	-	609
P618_2	-	-	-	-	-	-

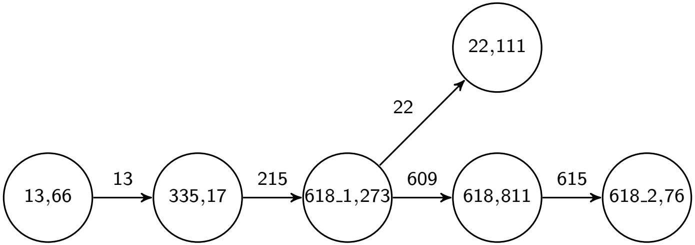
Selected *Node* → *P*22

$$\begin{tikzcd}
 & & & 22{,}111 \\
13{,}66 \arrow[r, "13"] & 335{,}17 \arrow[r, "215"] & 618{,}273 \arrow[r, "609"] \arrow[ur, "22"] & 618{,}811
\end{tikzcd}$$

Lineage Algorithm→ Buiding Lineage Tree

	P13	P22	P335	P618	P618_1	P618_2
P13	-	12	13	13	13	13
P22	-	-	15	13	22	13
P335	-	-	-	214	215	214
P618	-	-	-	-	609	615
P618_1	-	-	-	-	-	609
P618_2	-	-	-	-	-	-

Selected *Node* → *P*618_2

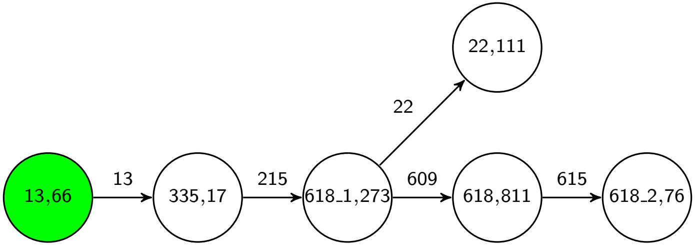


Lineage Algorithm→ Adding Cross-Edges

- Multiple parents, i.e., branching and merging
- Iterate over each node in topological order
- Ignore successor and predessors
- Non-inherited functions

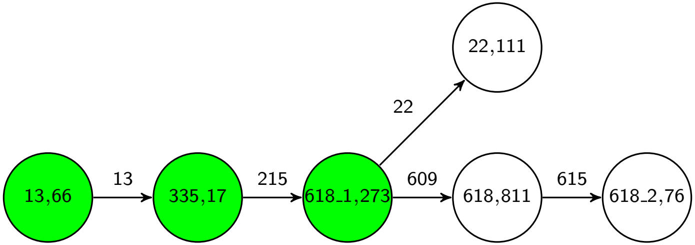
Lineage Algorithm→ Adding Cross-Edges

- Multiple parents, i.e., branching and merging
- Iterate over each node in topological order
- Ignore successor and predessors
- Non-inherited functions



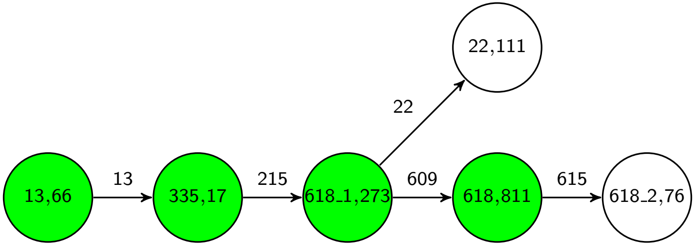
Lineage Algorithm→ Adding Cross-Edges

- Multiple parents, i.e., branching and merging
- Iterate over each node in topological order
- Ignore successor and predessors
- Non-inherited functions



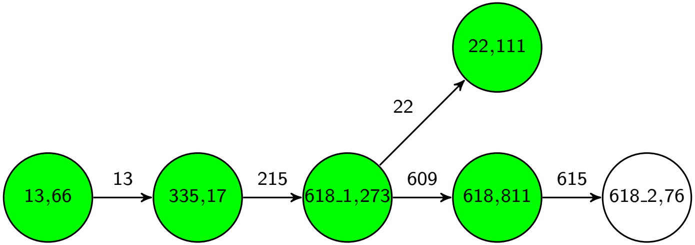
Lineage Algorithm→ Adding Cross-Edges

- Multiple parents, i.e., branching and merging
- Iterate over each node in topological order
- Ignore successor and predessors
- Non-inherited functions



Lineage Algorithm→ Adding Cross-Edges

- Multiple parents, i.e., branching and merging
- Iterate over each node in topological order
- Ignore successor and predessors
- Non-inherited functions



Evaluation → Goals

- Benign Lineage
- Malware Lineage

Evaluation → Dataset

Benign (631 Versions, 13 Programs)

– FileZilla → **10 years**
– Fzputtygen → **10 years**
– Fzsftp → **10 years**
– Notepad++ → **14 years**
– Pageant → **16 years**
– Plink → **16 years**
– ProcessHacker → **6 years**
– PSCS → **16 years**
– PSFTP → **14 years**
– PuTTY → **14 years**
– PuTTYgen → **16 years**
– PuTTYtel → **14 years**
– WinSCP → **10 years**

Malware (7,793 Samples, 10 Families)

– Allaple → **4,000**
– IRCBot → **365**
– Klez → **750**
– Loring → **216**
– Memery → **113**
– Picsys → **131**
– Simbot → **214**
– Sytro → **1,354**
– Urelas → **206**
– VtFlooder → **444**

Evaluation → Benign

Lineage Inference evaluation on open-source programs.

	Reference		**SPP**				**Raw**			
Program	**Type**	$\|V\|$	$\|V\|$	$\|V^r\|$	$\|X\|$	**PO**	$\|V\|$	$\|V^r\|$	$\|X\|$	**PO**
FileZilla	S	119	117	1	8	96%	119	1	20	97%
Fzputtygen	S	107	19	1	0	72%	30	1	0	96%
Fzsftp	S	116	50	1	0	71%	52	1	2	33%
Notepad++	D	70	70	1	6	65%	70	1	14	71%
Pageant	S	18	18	1	0	93%	18	1	0	97%
Plink	S	18	18	1	0	99%	18	1	1	90%
ProcessHacker	2-S	52	52	2	2	76%	52	2	7	79%
PSCP	S	20	20	1	1	99%	20	1	1	73%
PSFTP	S	16	16	1	0	99%	16	1	0	88%
PuTTY	S	22	22	1	2	100%	22	1	0	92%
PuTTYgen	S	17	16	1	0	87%	16	1	0	66%
PuTTYtel	S	4	4	1	1	100%	4	1	0	50%
WinSCP	S	47	47	1	1	100%	47	1	9	100%

Lineage → Benign

PuTTY

$$\begin{tikzcd}
 & & & & 0.50 \arrow[r] & 0.51 \arrow[dr, "*"] & & & & & & & & & & & & & & & & \\
0.46 \arrow[r] & 0.47 \arrow[r] & 0.48 \arrow[r] \arrow[urr] & 0.49 \arrow[ur, "*"] \arrow[rrr] & & & 0.52 \arrow[r] & 0.53 \arrow[r] & 0.54 \arrow[r] & 0.55 \arrow[r] & 0.56 \arrow[r] & 0.57 \arrow[r] & 0.58 \arrow[r] & 0.59 \arrow[r] & 0.60 \arrow[r] & 0.61 \arrow[r] & 0.62 \arrow[r] & 0.63 \arrow[r] & 0.64 \arrow[r] & 0.65 \arrow[r] & 0.66 \arrow[r] & 0.67
\end{tikzcd}$$

Fzputtygen

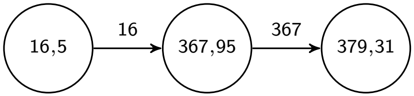
$$\begin{tikzcd}
3.0.10 \arrow[r] & 3.2.0 \arrow[r] & 3.3.5.1 \arrow[r] & 3.5.1 \arrow[r] & 3.7.2 \arrow[r] & 3.7.3 \arrow[r] & 3.7.4.1 \arrow[r] & 3.8.1 \arrow[r] & 3.9.0.1 \arrow[r] & 3.10.0.1 \arrow[r] & 3.10.2 \arrow[r] & 3.11.0.1 \arrow[r] & 3.13.0 \arrow[r] & 3.14.0 \arrow[r] & 3.15.0.1 \arrow[r] & 3.17.0.1 \arrow[r] & 3.20.0 \arrow[r] & 3.21.0 \arrow[r] & 3.24.0
\end{tikzcd}$$

Evaluation $\rightarrow$ Malware

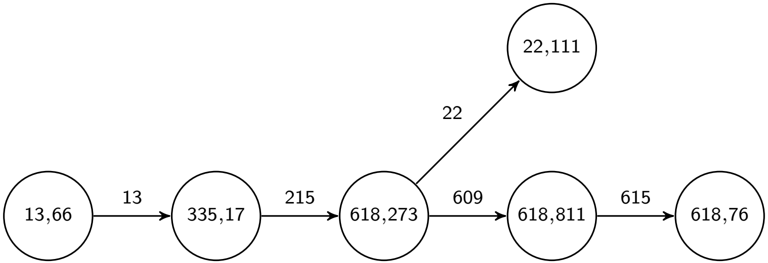
Lineage Inference evaluation on malware samples.

		$\lvert V \rvert$		$\lvert E \rvert$		$max(\lvert L(v_i) \rvert)$		$\lvert L(v_i) = 1 \rvert$		$max(\lvert F(v_i) \rvert)$		$min(\lvert F(v_i) \rvert)$		$\lvert \bigcup F(v_i) \rvert$	
Family	**EXE**	**spp**	**raw**	**spp**	**raw**	**spp**	**raw**	**spp**	**raw**	**spp**	**raw**	**spp**	**raw**	**spp**	**raw**
allaple	241	114	143	113	142	71	71	92	138	250	301	10	10	413	510
klez	118	13	13	12	12	93	93	8	8	618	667	244	251	666	927
memery	140	12	12	11	11	66	66	3	3	121	123	25	27	131	136
picsys	113	5	5	4	4	92	92	1	1	379	473	16	16	397	498
simbot	135	21	93	20	92	65	24	13	82	67	71	17	17	108	1,723
sytro	186	4	4	3	3	92	92	0	0	617	667	335	350	754	1,290
vtflooder	170	20	69	18	66	75	75	11	61	712	749	10	10	3,202	4,652

Lineage → Malware



Picsys



Sytro

Limitations

- Packers that modify the original code, e.g., VMProtect
- Evasion, anti-VM checks
- Code semantics require manual analysis
- Function identification, e.g., Nucleus

Conclusion

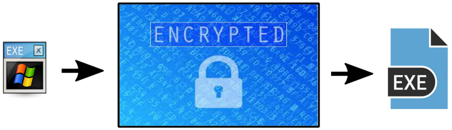
- An approach to perform malware lineage on samples collected in the wild.
- First study to identify number of versions in a malware family.
- Evaluated on 13 benign programs and 10 malware families.
- Our approach provides, on average, 26x reduction from input sample versions.

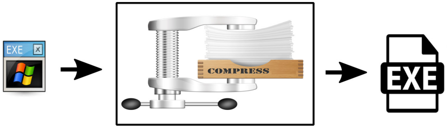
Thank You!

Questions?

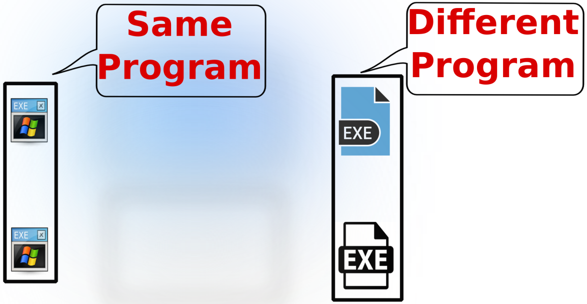
Packing/Unpacking

Packing/Unpacking





Packing/Unpacking



back