MALWARE – PART II

Prof. Dr. Bernhard Tellenbach

Content

- Malware Defense
- Detection techniques
  - Signatures
  - Heuristics
  - Behavior
  - Reputation
- Evasion
- How good is Anti-Virus?

Goals

- You have a basic understanding of why our defenses against malware are still quite weak
- You can explain why anti-virus software is still an effective tool against (some) malware
- You know how anti-virus software works (signatures, fuzzy-hashes,…) and you can discuss its limitations

Malware Defense

Quiz on Detecting Malware

- Can a tool detect 100% of all viruses that enter your system in form of a file?

- What's the problem with this approach?

Detecting Malware - Are we Good at it?

Why Aren't we Better at (Malware) Defence?

- For one single offensive LOC defenders write 100'000 LOC
  - 120:1 Stuxnet to average malware
  - 500:1 Simple text editor to average malware
  - **100,000:1** Defensive tool to average malware
  - 1,000,000:1 Target operating system to average malware

Positive aspect: Few LOCs make the analysis of (small) malware samples practical.

- Prospect theory "we are risk adverse when it comes to gains and risk taking when it comes to losses" (=>we don't act until it is too late!)
- Detecting malware collected in the wild is often «easy» as long as the characteristics used for the detection do not change.
- Chicken and egg problem - To know how to detect a malware, it must be analyzed. But to analyze it, we must find a sample of it.

LOCs of offensive and defensive tools

According to an analysis of 9'000 malware samples by Mudge [1], malware has 125 lines of code (LOC) on average.

Today, this number seems a bit small since «advanced» malware like stuxnet (~15'000 LOC) and the Zeus Trojan (~250'000 LOC) have many more lines of code.

According to Mudge, modern defensive products have at least 10 million LOCs resulting in a very asymmetric business even when compared to modern malware suites.

[1] Mudge, "How a Hacker Has Helped Influence the Government --- and Vice Versa" Blackhat 2011

Detecting Malware - Overview

- Anti-Malware software uses multiple detection engines
- Detection engines differ in the detection technique employed
- Detection engines differ in how they obtain the required input data
- One way to classify engines is by combining these two features
- Note: Hybrid forms are quite common, especially with regard to input data

Detection technique	**Signature-based**
	Heuristic-based
	Behavior-based
	Reputation-based
Input data	**Static**
	Dynamic

Detecting Malware - Input Data Collection

- Static analysis: Data obtained without executing the sample
- Dynamic analysis: Data obtained while the sample is being executed
- The sample might be run natively, in an emulator, or a VM
- Data collection might be done at the same or at a different layer
  - E.g., run a sample in a VM and collect and analyze data from the outer layer (host)
- If at a different layer, malware has a harder time to detect being analyzed

Static

- File metadata (e.g., PE header)
- Binary data/code
- Assembly code
- Assembly code characteristics
- API / System calls
- …

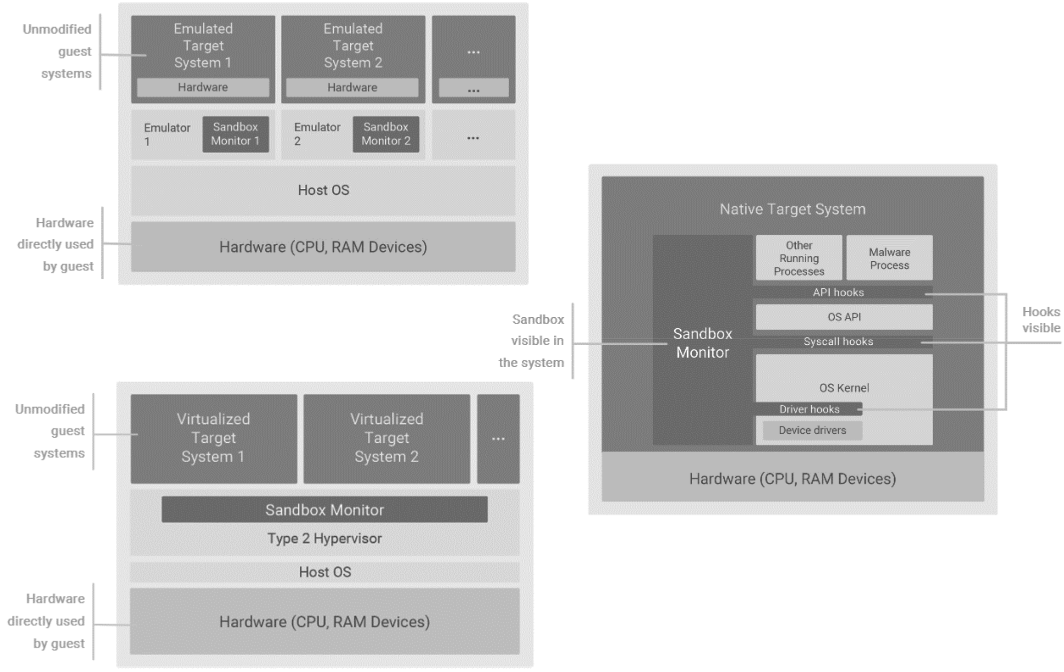
Dynamic

- Memory
- Network access/traffic
- Filesystem
- API / System calls
- …

For many, **emulation** and **virtualization** go hand in hand, but there are some key differences. When a device (ev. including software that runs on it) is being emulated, a software-based construct has replaced a hardware component (and eventually also its software components). Its possible to run a complete virtual machine on an emulated server. However, virtualization makes it possible for that virtual machine to run directly on the underlying hardware, without needing to impose an emulation tax (the processing cycles needed to emulate the hardware). With virtualization, the virtual machine uses hardware directly, although there is an overarching scheduler. As such, no emulation is taking place, but this limits what can be run inside virtual machines to operating systems that could otherwise run atop the underlying hardware.

With emulation, since an entire machine can be created as a virtual construct, there are a wider variety of opportunities, but with the emulation penalty. But emulation makes it possible to, for example, run programs designed for a completely different architecture on an x86 PC. This approach is common, for example, when it comes to running old games designed for obsolete platforms on today's modern systems. Because everything is emulated in software, there is a performance hit in this method, although todays massively powered processors often cover for this.

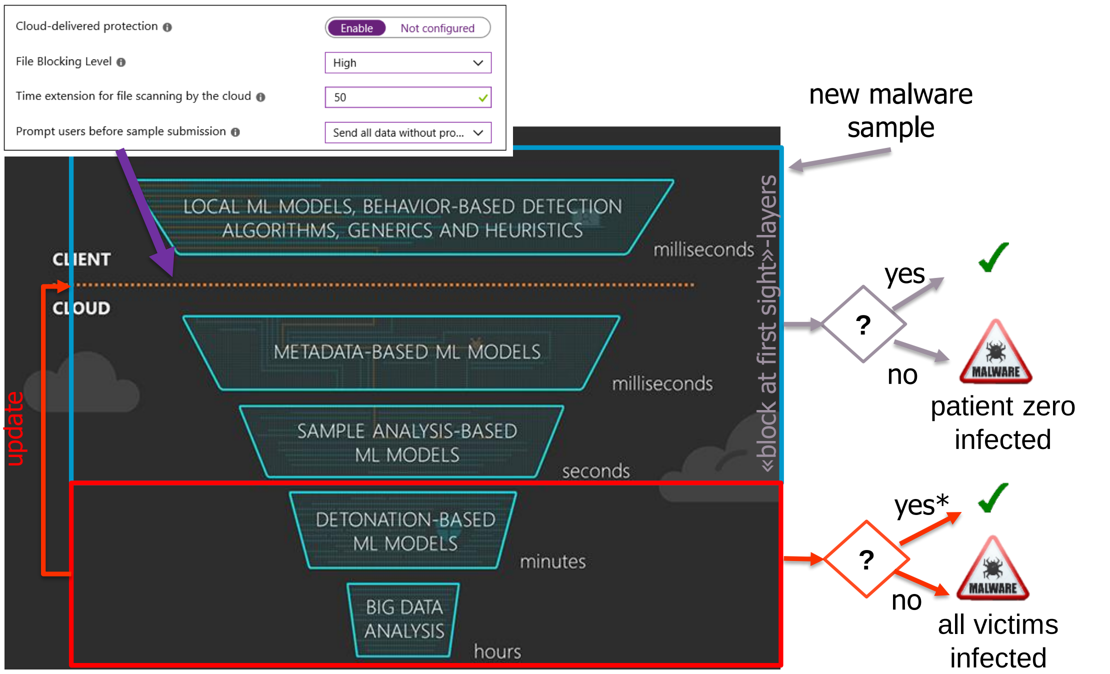
Natively (with hooking), Emulation and Virtualization



AV-System Architecture

Anti-Virus Technology

- Host-based and network-based anti-virus solutions exist
- Most of them have client- (local) and cloud-based components
- Cloud-based component provides «instant-update» since signatures are hosted in the cloud
  - Usually, there is a local cache in case connectivity is lost
  - Cache often contains "most relevant" signatures only
- Bandwidth limitation – At first, only meta data is submitted
  - the file's unique identifier: (fuzzy) hash(es)/fingerprints
  - data about how the file came to be in the system
  - how the file behaved (behavioral blocker only)
- Unknown files might be temporarily quarantined on the client machine and the file sent to the cloud for analysis (automated and eventually also manual)



Most anti-virus solutions today use a multi-layered approach where each layer consists of detective measures that operate at a different time scale.

The diagram on the slide shows the basic layers for the Microsoft Defender Advanced Threat Protection (ATP) solution [1].

Defeating one layer does not mean that the attacker evaded detection, as there are still opportunities to detect the attack at the next layer, albeit with an increase in time to detect it.

However, that requires that a potentially dangerous activity such as opening or running a file that has never been analyzed yet (“first sight”), is blocked until a decision can be made.

This results in a security-usability trade-off. Most solutions offer a way to configure the maximum wait time (or similar) if other layers than the client-layer (local) are enabled.

In the case of Microsoft Defender ATP, the first three layers are considered as being part of the **“block at first sight”** protection [2]. The use of the cloud layers and “block at first sight” is enabled by default in most enterprise deployments. **Custom configurations** are possible too. The **dialog box** on the slide (from Microsoft Intune) illustrates this. Alternative options to configuring the settings is to directly modify the Microsoft Group Policy Objects [3].

The fact that the other two layers are not part of the “block at first sight” protection might be due to the following two reasons: The first one is that they take quite a lot of time for their analysis. The second one is, that if these layers find something, the finding must be **fed back to the first three layers**. Only after if it is fed back to these layers, the problematic action will be blocked in the future.

If we take on an attacker’s perspective, an attacker that can **evade the “block at first sight”** layers, would not be stopped. At least not on “patient zero” – at first sight. If the malware is later found by the two remaining layers**, further victims are prevented** from the point where the **first three layers are updated** to identify this threat.

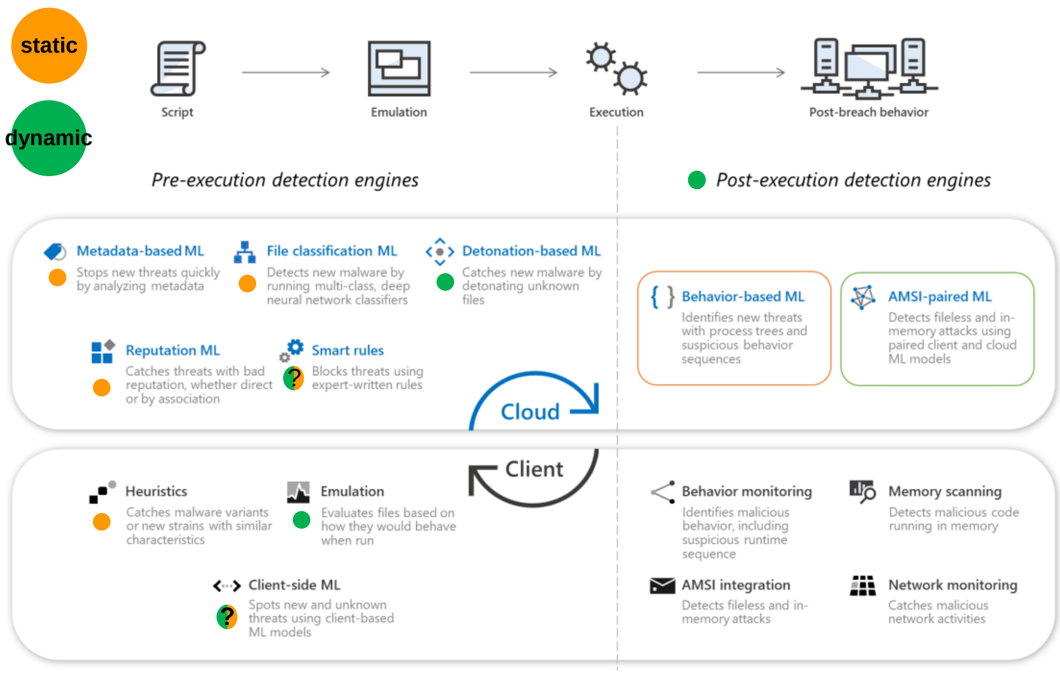
Sources:

1. Protecting the protector: Hardening machine learning defenses against adversarial attacks, https://www.microsoft.com/security/blog/2018/08/09/protecting-the-protector-hardening-

machine-learning-defenses-against-adversarial-attacks/

2. Inside out: Get to know the advanced technologies at the core of Microsoft Defender ATP next generation protection
https://www.microsoft.com/security/blog/2019/06/24/inside-out-get-to-know-the-advanced-technologies-at-the-core-of-microsoft-defender-atp-next-generation-protection/
3. Enable block at first sight
https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-antivirus/configure-block-at-first-sight-windows-defender-antivirus

AV System Architecture – Client + Cloud



Anti-virus solutions usually consist of **client-** and **cloud-based** components. This diagram shows an outline of the components (engines) used by the **Microsoft Windows Defender Advanced Threat Protection (ATP)** solution.

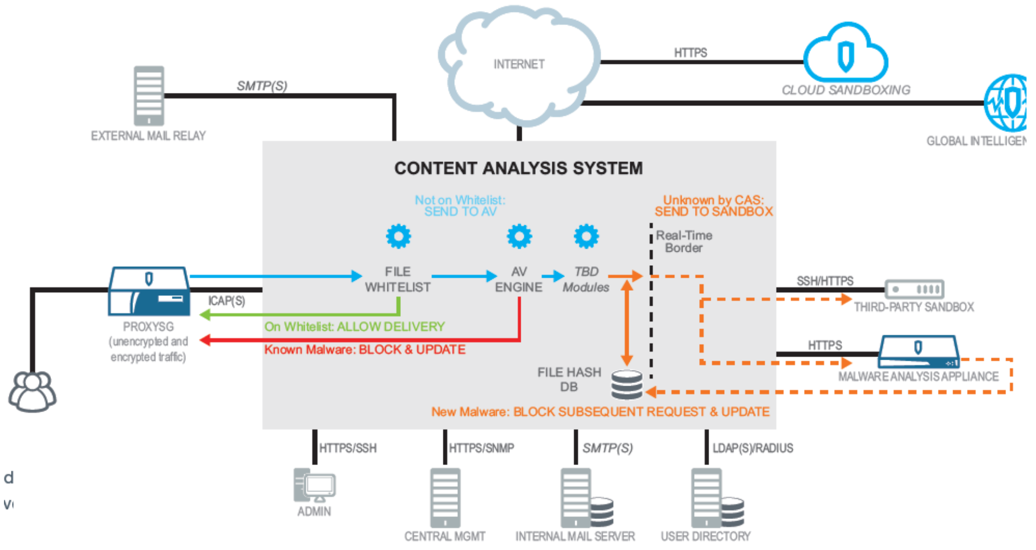
On the left, it shows the engines that do their work mainly based on files (executables, scripts, documents,…) and before they are executed/used/opened. Hence, this involves mostly static analysis approaches but also two dynamic ones: (1) analysis of the file in an emulator on the client and (2) opening/execution (=detonation) of the file in the cloud. Note that in the case of the client-side emulator in Defender, the emulator has many limitations. For example, the emulation of the Windows API is very limited. Many functions do not do anything and return (if at all) some fixed return values.

When the pre-execution detection engines did their work and nothing was detected, the execution is permitted. In this case, the post execution detection engines displayed on the right come into play. On the client side, there are engines for monitoring the network activities (e.g., connection to a blacklisted endpoint), the memory and suspicious process trees and sequences of activities on the system. Such sequences might for example be that a winword.exe process writes to the memory of another process and then starts a remote threat in that process or that it drops an executable file to disk. And on the cloud-side, more complex machine learning models are fed with data from the client, if some triggers are hit.

There are some interesting articles/blog posts from Microsoft and other parties detailing various aspects of the Windows Defender APT solution.

https://www.microsoft.com/security/blog/2019/06/24/inside-out-get-to-know-the-advanced-technologies-at-the-core-of-microsoft-defender-atp-next-generation-protection

AV System Architecture – Network



The sample architecture stems from Blue Coat but looks similar in other products.

1. A user downloads content from the web through the ProxySG secure web gateway, which sends it to the Content Analysis System for malware scanning via ICAP or ICAPS.
2. The Content Analysis System checks the file in real time against the known-good-file whitelist database, which is hosted in the Global Intelligence Network. If it's listed there, the file is delivered and Content Analysis System processing is finished. A temporary local cache is maintained for performance reasons.
3. If the file is not whitelisted, it's scanned by one or two anti-virus (AV) engines. If the file is known bad (rated 0) it is blocked and its URL is added to the Global Intelligence Network.
4. If the file is neither known good or known bad (rated 1), it can be sent to one or more sandboxing appliances, including the Malware Analysis Appliance or any third party sandbox. The file will only be sent if the file has not yet been analyzed.
5. When sandboxing is complete, the result goes to the Content Analysis System. If the file is malicious, the Content Analysis System updates the local cache – the file hash database – and tells the ProxySG to block all subsequent requests to the same object. It also updates the Global Intelligence Network with the object's URL, file hash, timestamp and filename.

ICAP(S) - The Internet Content Adaptation Protocol (**ICAP**) is a lightweight HTTP-like protocol specified in RFC 3507 which is used to extend transparent proxy servers, thereby freeing up resources and standardizing the way in which new features are implemented.

Signatures

Anti-Virus Technology – Signatures (Traditional)

- Traditionally, a signature captures the unique characteristics of a malware at the byte-level
- Most common types of traditional signatures that are still in use are hashes, fuzzy-hashes, and characteristic byte sequence(s)
- Signature matching depends on the signature
  - Does the file match a signature? (byte sequences)
  - Does the signature of the sample match a hash in the database? (hash)
- Up to tens of thousands of signatures added every day
  - Clever data structures and storage formats enable anti-virus solutions to check a sample vs. all signatures in <30 ms
- Signatures often supported by whitelisting of known-good files that would trigger an alert otherwise and/or context (file size, origin,…)

Note that **the term signature is not well-defined** in the malware domain. Kaspersky explains it as follows:

From the very beginning, in the 1980s, signatures as a concept were not clearly defined. Even now, they don't have a devoted Wikipedia page, and the entry on malware uses the term without defining signatures, as if were such common knowledge as to go without explanation.

So: Let's define signatures at last! A virus signature is a continuous sequence of bytes that is common for a certain malware sample. That means it's contained within the malware or the infected file and not in unaffected files.

Nowadays, signatures are far from sufficient to detect malicious files. Malware creators obfuscate, using a variety of techniques to cover their tracks. That's why modern antivirus products must use more advanced detection methods. Antivirus databases still contain signatures (they account for more than half of all database entries), but they include more sophisticated entries as well.

As a matter of habit, everyone still calls such entries "signatures." There's no harm in that, as long as we remember that the term is shorthand for a gamut of techniques that make up a much more robust arsenal.

Source: https://www.kaspersky.com/blog/signature-virus-disinfection/13233/

Signatures - Hashes

Signatures - Hashes

- MD5, SHA-1 or other cryptographic hash functions can be used to check whether file (or binary blob) is known to be malicious or benign
- Problem: Polymorphic/mutating malware may change its code
  - Changing one bit of the input results in a different signature
- Today, it is mainly used to prevent the analysis of samples that are already known (benign or malicious) => pre-filter / speedup
  - For cloud-based solutions, known files are not sent to the cloud
  - For solutions with sandboxes, unknown samples might be sent to the sandbox
- A signature that is more robust to modifications of the malware would be great

Signatures – Fuzzy Hashes

Finding Similar Files – Fuzzy Hashes

- Most accurate approach to compare the byte content of two files is a side-by-side comparison looking for (byte-wise) differences
  - E.g., using a mix of xxd and diff
- Does not scale once we move from comparing two files to comparing one file to many (known malicious) files
- More efficient: Generate a fuzzy hash (=signature) for the file and compare it to the stored signatures
- Different approaches to compute fuzzy hashes exist
  - Context-triggered piecewise hashes (CTPH) (e.g., ssdeep, MRS, mrsh-v2, bbHash)
  - Other similarity digest approaches exist, for example sdhash and tlsh

Context Triggered Piecewise Hashes (CPTH)

- Basic idea:
  1. Segment a file into pieces (e.g., blocks of size n-bytes)
  2. "Hash" each piece
  3. Compile a hash from the "hashes" (e.g., concatenate them)

- What needs to be defined?
  - What alphabet to be used for the signature (optional)
    - Alphabets like base64 simplify inspection of the signatures (human readable)
  - How we want to segment the file into "summarizeable" pieces
  - Technique for summarizing a piece and form the hash composed of alphabet characters

First Attempt – Blockwise, 4-byte Blocks

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	48	49
394				410				426				442				458				474				340			
K				a				q				6				K				a				U			
Kaq6KaU																											

a	b	c	d	e	f	g	h	i	h	e	l	l	o	o	p	q	r	s	t	u	v	w	x	y	z	0	1
97	98	99	100	101	102	103	104	105	104	101	108	108	111	111	112	113	114	115	116	117	118	119	120	121	122	48	49
394				410				418				442				458				474				340			
K				a				i				6				K				a				U			
Kai6KaU																											

Different block and
different «summary»
=> GOOD!

Compare: Kaq6KaU with Kai6KaU **=> similar!**

Different block but same «summary»
=> **more collision resistant** «summary function»

file content
byte values (ASCII)
sum of ASCII values
base64 char of ascii sum modulo 64
signature

First Attempt – Blockwise, 4-byte Blocks

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	48	49
394				410				426				442				458				474				340			
K				a				q				6				K				a				U			
Kaq6KaU																											

a	b	c	d	e	f	g	h	i	j	k	l	m	n	h	o	p	q	r	s	t	u	v	w	x	y	z	0	1	N/A	N/A	N/A
97	98	99	100	101	102	103	104	105	106	107	108	109	110	104	111	112	113	114	115	116	117	118	119	120	121	122	48	49	0	0	0
394				410				426				434				454				470				411				49			
K				a				q				y				G				W				b				x			
KaqyGWb																															

Compare: Kaq**6KaU** with Kaq**yGWbx** **=> NOT similar!**

- There is similar content but not at the 4-byte block level
- Using fixed blocks is not the best approach…

file content
byte values (ASCII)
sum of ASCII values
base64 char of ascii sum modulo 64
signature

2nd Attempt: Context Triggered Piecewise Hashes (CPTH)

- Instead of fixed block boundaries, the content (context) determines the bounds
- One way to do this is by using a rolling hash (e.g., as in ssdeep and spamsum)

- Example of a simplistic rolling hash:

Fixed window size: 3

Calculation: ∑ mod 8

Trigger value: **7**

a	b	c	d	e	f	g	h	i
97	98	99	100	101	102	103	104	105

97 mod 8 = 1

195 mod 8 = 3

294 mod 8 = 6

297 mod 8 = 1

300 mod 8 = 4

303 mod 8 = **7**

2nd Attempt: Context Triggered Piecewise Hashes (CPTH)

97	195	294	297	300	303	306	309	312	315	318	321	324	327	330	333	336	339	342	345	348	351	354	357	360	363	291	219
1	3	6	1	4	7	2	5	0	3	6	1	4	7	2	5	0	3	6	1	4	7	2	5	0	3	3	3
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	48	49
597						852								916								579					
V						U								U								D					
VUUD																											

insert a 'h'

97	195	294	297	300	303	306	309	312	315	318	321	324	327	323	325	327	336	339	342	345	348	351	354	357	360	363	291	219
1	3	6	1	4	7	2	5	0	3	6	1	4	7	3	5	7	0	3	6	1	4	7	2	5	0	3	3	3
a	b	c	d	e	f	g	h	i	j	k	l	m	n	h	o	p	q	r	s	t	u	v	w	x	y	z	0	1
97	98	99	100	101	102	103	104	105	106	107	108	109	110	104	111	112	113	114	115	116	117	118	119	120	121	122	48	49
597						852								327			693						579					
V						U								H			1						D					
VUH1D																												

Compare: VU**U**D with VU**H1**D **=> similar!**

- More resilient to insertions – recovers if similar content is seen after a difference
- Window size and other factors determine minimal file size to produce meaningful results

Fuzzy Hashes – CTPH Summary

- Context Triggered Piecewise Hashes (CPTH)
  - First proposed and implemented (ssdeep) by Kornblum
  - Rolling hash - r_p at position p produces a pseudo-random value based only on the current context (the last s bytes) of the input

$$r_p = F(b_p, b_{p-1}, b_{p-2}, \ldots, b_{p-s})$$

  - r_p is used to decide whether p is a reset point (block boundary)
  - The blocks are then hashed and summarized in an overall hash
- How different are the files (hashes)?
  - ssdeep uses the edit distance between the hashes as a distance measure
- Effective in finding samples where many blocks are similar and have not been moved around too much

Piecewise hashing - The technique was originally developed to mitigate errors during forensic imaging. If an error occurred, only one of the piecewise hashes would be invalidated. The remainder of the piecewise hashes, and thus the integrity of the remainder of the data, was still assured. Piecewise hashing can use either cryptographic hashing algorithms, such as MD5 in dcfldd or more traditional hashing algorithms such as a Fowler/Noll/Vo (FNV) hash

Context triggered piecewise hashing (CTPH)

This was first proposed by Kornblum [1] and implemented in the ssdeep tool. It originates from the spam detegtion algorithm of Tridgell implemented in spamsum. The ssdeep tool divides a byte sequence (file) into chunks and hashes each chunk separately using the Fowler-Noll-Vo (FNV algorithm. CTPH then encodes the six leas significant bites of each FNV hash as a Base64 character. All the characters are concatenated to create the file fingerprint. The trigger points for splitting a file into chunks are determined by a rolling hash function. This function, which is a variation of the Adler-32 algorithm, is computed over a seven-byte sliding window to generate a sequence of pseudorandom numbers. A number r in the sequence trigger a chunk bundary if r= -1 mod b. The modulus b, called the block size, correlates with the file size. Konrblum suggests dividing a file into approximately S=64 chunks and using the same modulus b for similar sized files. The modulus b is a saltus function: b=bmin*2^floor(log2(N/S/bmin)) where bmin=3 and N is the input length in bytes. Since two fingerprints can only be compared if they were generated using the same block size, ssdeep calculates two fingerprints for each file using the block sizes b and 2b and stores both fingerprints in one ssdeep hash.

[1] Jesse Kornblum. 2006. Identifying almost identical files using context triggered piecewise hashing. Digit. Investig. 3 (September 2006), 91-97

Active Attacks on CPTH

If the content dependent sequence of pseudorandom numbers for creating the chunks is not «random» enough, an active adversary might find attack vectors to bypass CPTH based signature matching. The following paper presents some attack vectors to bypass Konrblum's approach and presents a PRNG that is more efficient and «random» than Kornblum's approach

Harald Baier, Frank Breitinger, Security Aspects of Piecewise Hashing in Computer Forensics, Conference on IT Security Incident Management and IT Forensics (IMF), 2011

https://www.researchgate.net/publication/224243607_Security_Aspects_of_Piecewise_Hashing_in_Computer_Forensics

Real-World Example With ssdeep

```
> ssdeep EVA-Hacking-Lab-ssdeep1.docx
ssdeep,1.1--blocksize:hash:hash
768:fjU7sORx39kaUCi0ZBn7ThAwdQUWLjJ+pl20O1RTOl/OxMfj3DruNaSeHyBKha:o7sOb9kaUChNBvdQU7lMTU/OxMDHgmSZ
```

```
> ssdeep EVA-Hacking-Lab-ssdeep2.docx
ssdeep,1.1--blocksize:hash:hash
768:y7JRFx39kaUCi0ZBn7ThAwdQUWLjJ+pl20O1RTOl/OxMfj3Dru6eHyBvW:GRn9kaUChNBvdQU7lMTU/OxMDHISU
```

```
> ssdeep –m sig.txt EVA-Hacking-Lab-ssdeep2.docx
EVA-Hacking-Lab-ssdeep2.docx matches sig.txt:EVA-Hacking-Lab-ssdeep1.docx (86)
```

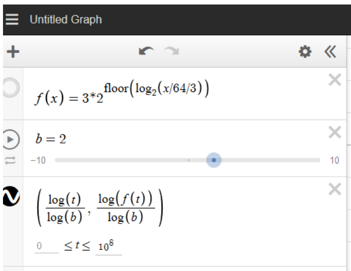

Despite the following differences

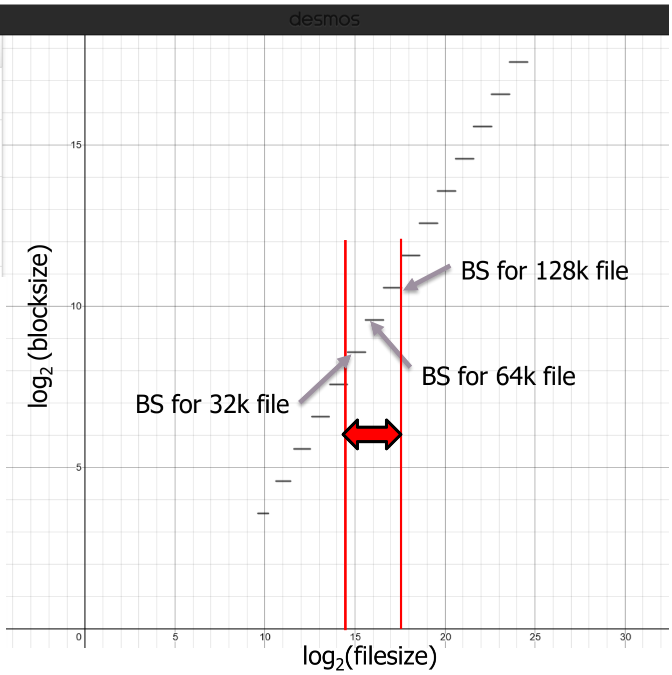
- Paragraph moved from the end to the beginning
- Parts of the table lines reordered
- Content added «… and the defender's …»
- Font and colour modifications

the content triggered piecewise hash calculated using ssdeep for the two word documents are at least partially the same.

According to ssdeep, the documents match with a similarity score of 86.

ssdeep and Blocksizes





- A file with size x has a blocksize b=f(x)
- For files to be comparable, they must share a hash for a specific «blocksize»
- Since hashes for b and 2*b are produced for each file, files whose size differ by more than ~3 orders of magnitude (log2-scale) cannot be compared

Blocksize: This is not really a blocksize but rather a parameter for the rolling hash. For a given filesize, the parameter is chosen so that it produces approximately 64 junks.

For details about the algorithm, see:

- Jesse Kornblum. 2006. Identifying almost identical files using context triggered piecewise hashing. Digit. Investig. 3 (September 2006), 91–97. https://www.dfrws.org/sites/default/files/session-files/paper-identifying_almost_identical_files_using_context_triggered_piecewise_hashing.pdf

Fuzzy Hashes – Other Approaches

- Similarity digest hashing (sdhash)
  - Looks for statistically improbable sequences of 64 bytes (features)
    - A file's feature set file is represented in Bloom filters
    - Size of the hash depends on the size of the input data
  - Effective in finding samples where some parts are copied from the same source (partial matching)
- Trendmicro locality sensitive hashing (TLSH)
  - Based on the frequency distribution of n-grams (substrings of n bytes)
  - Effective in finding samples that make use of the same building blocks
    - For example, text documents written with the same language (same words)
    - For example, binaries where code blocks have been re-ordered

Fuzzy Hashes – Usage, Security and Performance

- Usage:
  - Anti-virus products make use of this but there is little to no information about it (see notes)
- Security:
  - It should be hard to make a file "not similar" anymore with minimal changes only
  - Most fuzzy hashing schemes have weaknesses (see notes)
- Performance:
  - Originally, ssdeep was considered as de-facto standard
  - Recent research showed that it might perform badly in many scenarios relevant to malware
    - Recognizing object files embedded inside a statically linked binary
    - Compilation of the malware with a different compiler (and flags)
    - Binaries with few small changes applied at the assembly level
    - Binaries compiled from different versions of the same software
    - …

Usage

Usage is documented for example for McAfee and Microsoft Defender, but no details are given.

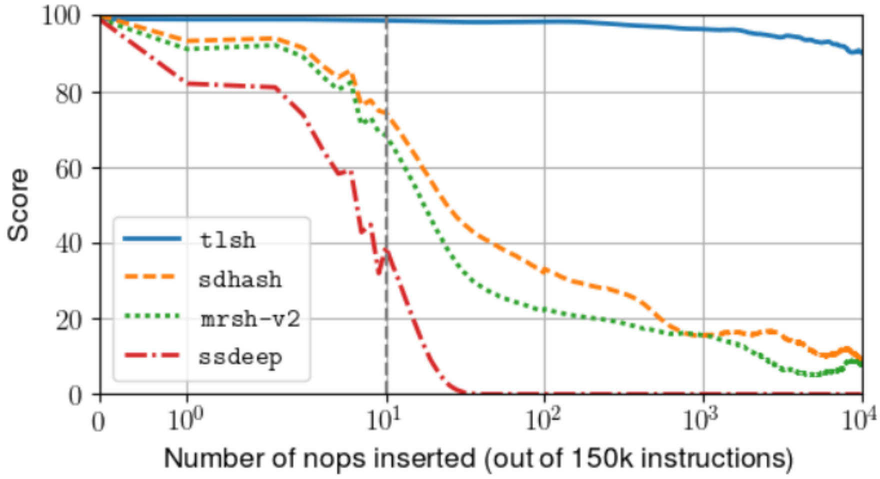
- In 2010, Christoph Alme and Declan Eardly from McAfee Labs published a report on **their Anti-Malware Engines** saying that McAfee makes use of several proprietary fuzzy fingerprinting techniques.
  https://kc.mcafee.com/resources/sites/MCAFEE/content/live/PRODUCT_DOCUMENTATION/25000/PD25219/en_US/mcafee_engine_technologies.pdf+&cd=2&hl=en&ct=clnk&gl=ch&lr=lang_de|lang_en
- Description of the Microsoft Defender Advanced Threat Protection (ATP) platform mentions the use of fuzzy hashes:
  "Feature selection is very important when training models that detect malware. There are two types of features that the researchers and machines look for: static file properties and behavioral components. Static file properties include things like if a file is signed or not, who signed the file, and various **fuzzy hashes**."
  https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet/customers/microsoft-defender

Security - Some attacks on fuzzy hashes:

- CTPH - Trigger points' attack [1]
  - H. Baier and F. Breitinger, "Security Aspects of Piecewise Hashing in Computer Forensics," *2011 Sixth International Conference on IT Security Incident Management and IT Forensics*, Stuttgart, 2011
- sdhash - Bloom filter shifting [2]
  - F. Breitinger, H. Baier, and J. Beckingham. "Security and implementation analysis of the similarity digest sdhash", In *First International Baltic Conference on Network Security & Forensics (NeSeFo)*, 2012

tlsh seems to be more robust but as of 2020, there is no publication that analyzes it in-depth.

Similarity Detection Performance - Example



Source: Fabio Pagani et al., 2018

Impact of small-scale modifications at the assembly level

- The test was applied to the ***stripped version of ssh-client***, a medium-size program containing over 150K assembly instructions.
- The test consisted of ***randomly inserting NOP instructions*** in the binary.
- The results, obtained by repeating the experiment ***100 times*** and ***averaging the similarity***, are shown in the above figure.

Conclusion from [1]:

"This study sheds light on how fuzzy hashing algorithms behave in program analysis tasks, to help practitioners understand if fuzzy hashing can be used in their particular context and, if so, which algorithm is the best choice for the task: an important problem that is not answered conclusively by the existing literature. Unfortunately, we found that the CTPH approach adopted by ssdeep—the most widely used fuzzy hashing algorithm—falls short in most tasks. We have found that other approaches (sdhash's statistically improbable features and tlsh' sn-gram frequency distribution) perform way better; more in particular, we have found that sdhash performs well when recognizing the same program compiled in different ways, and that tlsh is instead very reliable in recognizing variants of the same software when the code changes. Instead of blindly applying algorithms to recognize malware families and collecting difficult to interpret results, our evaluation looked at the details of both hashing algorithms and the compilation process: this allowed us to discover why fuzzy hashing algorithms can fail, sometimes surprisingly, in recognizing the similarity between programs that have undergone only very minor changes. In conclusion, we show that tlsh and sdhash consistently out-perform ssdeep and should be recommended in its place (tlsh is preferable when dealing with source code changes, and sdhash works better when changes involve the compilation toolchain); our analysis on where and why hashing algorithms are or are not effective sheds light on the impact of implementation choices, and can be used as a guideline towards the design of new algorithms."

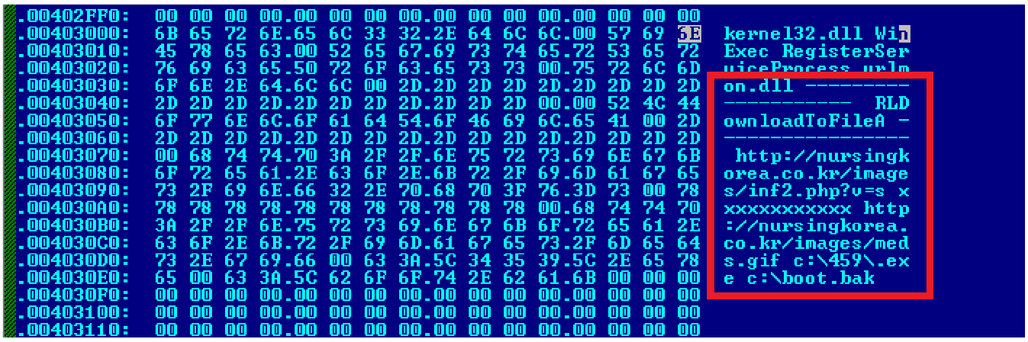
Source:

[1] Fabio Pagani, Matteo Dell'Amico, and Davide Balzarotti. 2018. 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* (*CODASPY '18*). Association for Computing Machinery, New York, NY, USA, 354–365. DOI:https://doi.org/10.1145/3176258.3176306

Signatures – Byte Sequences

Characteristic Byte Sequence - Example



Source: https://www.kaspersky.com/blog/signature-virus-disinfection/13233/

Virus Name	*String Pattern (Signature)*
Accom.1280	89C3 B440 8A2E 2004 8A0E 2104 BA00 05CD 21E8 D500 BF50 04CD
Die.448	B440 B9E8 0133 D2CD 2172 1126 8955 15B4 40B9 0500 BA5A 01CD
Xany.979	8B96 0906 B000 E85C FF8B D5B9 D303 E864 FFC6 8602 0401 F8C3

Source: B. Rad et al., *Evolution of Computer Virus Concealment and Anti-Virus Techniques: A Short Survey*

Anti-Virus Technology – Signature Rule Languages

- Different forms of signatures based on byte-sequences exist
- One signature format is YARA
- YARA - YARA helps malware researchers to identify and classify malware samples
- Open-source tool
- Multi-platform scan engine
- Many more use cases:
  - Microsoft Office document analysis (olevba.py)
  - Forensics (yaraPCAP)
  - Intrusion detection (Hipara)

```
global private rule zip_malware_size {
    meta:
        description = "Size of all samples is lower than 1MB –
                       setting limit to 3MB"
    condition:
        uint16(0) == 0x4B50 and filesize < 3MB
}

rule apt_equation_exploitlib_mutexes {
 meta:
    copyright = “Kaspersky Lab”
    description = “Rule to detect Equation
                  group’s Exploitation library”
    version = “1.0”
    last_modified = “2015-02-16”
    reference = “https://securelist.com/blog/”
 strings:
    $mz=“MZ”
    $a1=“prkMtx” wide
    $a2=“cnFormSyncExFBC” wide
    $a3=“cnFormVoidFBC” wide
    $a4=“cnFormSyncExFBC”
    $a5=“cnFormVoidFBC”
 condition:
    (($mz at 0) and any of ($a*))
}
```


YARA

The above YARA rule is just a simple example, more complex and powerful rules can be created by using wild-cards, case-insensitive strings, regular expressions, special operators and many other features that you'll find in the following documentation:

- Main documentation: http://yara.readthedocs.org/
- YARA Performance Guidelines: https://gist.github.com/Neo23x0/e3d4e316d7441d9143c7

YARA can be used for much more than just scanning files to identify malware.

Another use case worth mentioning is forensics. Volatility, a popular memory forensics tool, supports YARA scanning in order to pinpoint suspicious artefacts like processes, files, registry keys or mutexes. Traditionally YARA rules created to parse memory file objects benefit from a wider range of observables when compared to a static file rules, which need to deal with packers and cryptors. On the network forensics counterpart, **yaraPcap**, uses YARA for scan network captures (PCAP) files. Like in the SPAM analysis use case, forensic analysts will be in advantage when using YARA rules to leverage the analysis

Source: http://countuponsecurity.com/2016/02/10/unleashing-yara-part-1/

More details including more YARA rules and other indicators of compromise for the #EquationAPT group of cyber attacks can be found here:

https://securelist.com/blog/research/68750/equation-the-death-star-of-malware-galaxy/

Heuristics

Heuristics - Example

```
view plain  print  ?

Rule A
An API call to RtlMoveMemory with a string of "SOFTWARE\Classes\http\shell\open\commandV"

Rule B
An API call to CreateMutexA with a string of ")!VoqA.I4"

Rule C
An API call to GetSystemDirectory

if ( Rule A then Rule B then Rule C )
then
Process  = PoisonIvy

Keribos Output
Rule A
simple.exe   | 00401447 | RtlMoveMemory(0012F458, 0040162F: "SOFTWARE\Classes\http\shell\open\commandV", 00000028) ret
...........
Rule B
simple.exe   | 0040155D | CreateMutexA(00000000, 00000000, 0012F43B: ")!VoqA.I4") returns: 0000003C
...........
Rule C
simple.exe   | 004018BF | GetSystemDirectoryA(0012F6F1, 000000FF) returns: 00000013
```

Rule A

Rule B

Rule C

IF the rules A, B and C match, then it is Poison Ivy

Source: http://hooked-on-mnemonics.blogspot.com/2011/01/intro-to-creating-anti-virus-signatures.html

Heuristics-based Detection

- Heuristic-based engines make use of a set of rules and weighing methods written by domain experts
- Static heuristic analysis is usually founded on the analysis of file structure (metadata) and code organization
  - E.g., suspicious section characteristics (e.g., writable code section), uncommon section names, import from KERNEL32.DLL by ordinal, …
- Dynamic heuristic analysis performs emulation of virus code to extract the features required to evaluate the rules
- Each AV engine uses different algorithms and different proprietary techniques
- Behavioral-based and heuristics-based are often used interchangeably
  - Heuristics is more general as it is not limited to behavioral features
  - Heuristic engines are usually run prior to the runtime-behavioral analysis

Heuristics vs. Signatures

In contrast to signatures, heuristics are more universal in the sense that they can deal with modifications to the code and structure of a malware, if the characteristics tracked by a heuristic "signature" does not change. This way, heuristics can sometimes even deal with the evolution of malware variants.

Behaviour

The approaches are not based on the malware-binary itself but on what the malware does

Behavioral (or sematic) detection approaches are unaffected by changing the form of the malware-binary

Behavioral Detection - Example

A suspicious file was observed

Manage

Severity: Medium
Category: Malware
Detection source: Windows Defender ATP

Description

This file exhibits behaviors or traits of malware. It might do one or more of the following:

1. Give a remote attacker access to your PC.
2. Download and install other malware.
3. Record your keystrokes and the sites you visit.
4. Send information about your PC, including user names, passwords and browsing history, to a remote malicious hacker.
5. Use your computer for click-fraud, bitcoin mining, DDoS attacks and spamming.

Our algorithms found this file as malicious due to the following factors:

suspicious behaviors observed on this or other machines
suspicious memory activity observed on this or other machines
combination of structural and behavioral signals observed during file scan.

Source: https://www.microsoft.com/security/blog/2017/08/03/windows-defender-atp-machine-learning-detecting-new-and-unusual-breach-activity/

Behavioral Detection

- Behavioral (sematic) detection is unaffected by changing a malware's form – it is based on what the malware does
- Behavior on the host and on the network is sometimes considered separately by different engines
- Examples of monitored behavior:
  - HOST: Files created or modified by the malware
  - HOST: Specific changes made to the registry
  - HOST: Processes spawned
  - NET: Network sockets created
  - NET: Network connections to specific hosts, domains (or naming schemes)
  - NET: Structure of the communication protocol used
    - message length, encoding, header information etc.

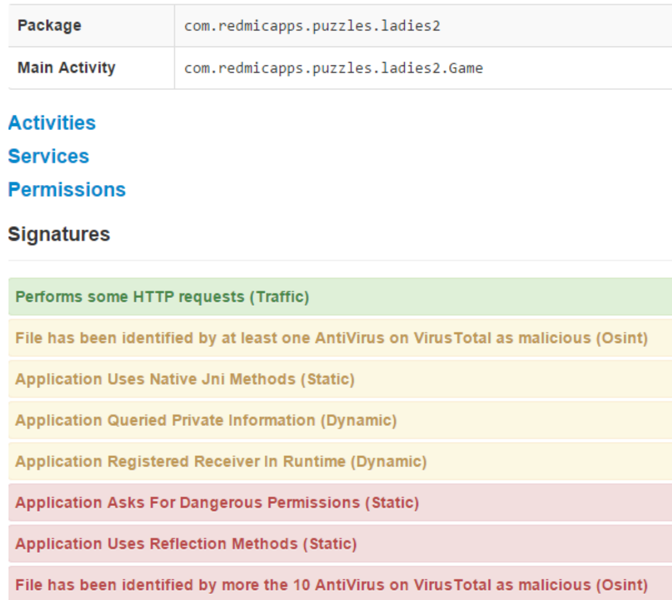
Analysis in a «Sandbox»

Behavioral Detection - Sandboxes

- Malware Sandbox - Captures the malicious program sample in a controlled testing environment (sandbox)
- Behavior can be studied and analyzed without affecting other systems
  - Monitor all operating system calls
    - file operations, process creation, network access, registry access etc.
  - Monitor network activity
    - Contacted end points, protocols used, traffic content, use of encryption,…
- Assumption: Malware behaves as it would in the wild

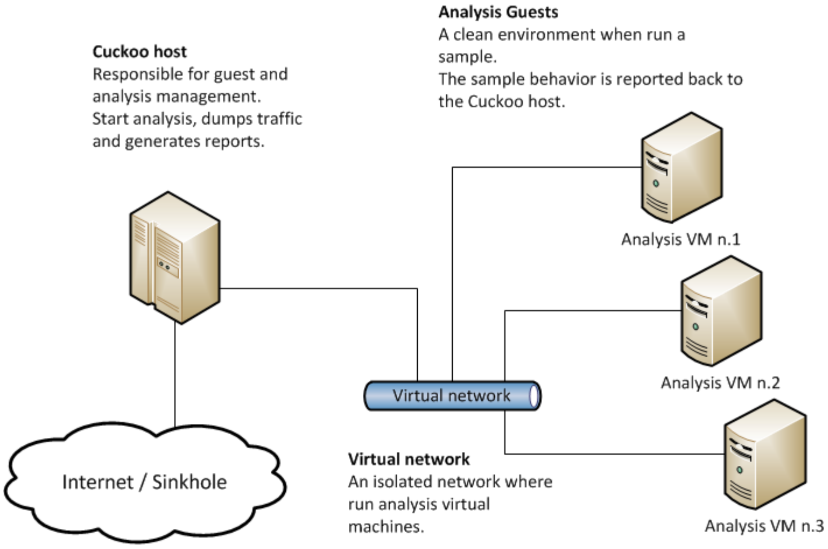
Cuckoo Sandbox

- Cuckoo Sandbox is a malware analysis system
  - Analysis of files for Windows, OS X, Linux, and Android
- By default, it can:
  - Analyze many different malicious file types and websites
  - Trace API calls and general behavior
  - Dump and analyze network traffic, even when encrypted
  - Perform advanced memory analysis with integrated support for Volatility



The **Volatility Framework** is a completely open collection of tools, implemented in Python under the GNU General Public License (GPL v2), for the extraction of digital artifacts from volatile memory (RAM) samples. The extraction techniques are performed completely independent of the system being investigated but offer unprecedented visibility into the runtime state of the system. The framework is intended to introduce people to the techniques and complexities associated with extracting digital artifacts from volatile memory samples and provide a platform for further work into this exciting area of research.

Cuckoo Sandbox



Analysis in a «Sandbox»

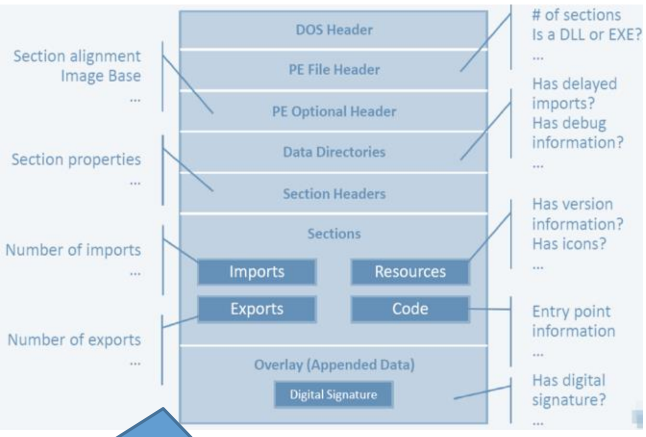
Reputation-based Analysis

- Based on what we know about a resource (e.g., file, URL, IP)
- Reputation based on things like prevalence, age, origin
  - Prevalence: A file is used on many computers around the world
    - => contributes to its reputation in a slightly positive way
  - Age: A domain was recently registered, and little is known about it at the moment
    - neutral or negative for its reputation
  - Origin: A file downloaded from the Internet is to be executed
    - negative for its reputation
  - Prevalence & Age: A file is used on many hosts and has been present for over a year now
    - contributes to its reputation in a positive way
- For malware files: Reputation systems cause a dilemma
  - Mutate more -> bad reputation/bad prevalence ->suspicious
  - Mutate less -> easy detection by signatures

Anti-Virus and ML

Next-Generation «Signatures» - Machine Learning

- Signatures, heuristics, or behavior are learned instead of crafted by experts
- Trains a model with millions and millions of data points
- Data points are from static (data, files) or dynamic (behavior) analysis
- Is a very active research field - some useful lessons learned:
  - https://www.microsoft.com/security/blog/2018/08/09/protecting-the-protector-hardening-machine-learning-defenses-against-adversarial-attacks/



- Static analysis examines "attributes" of the file without running the file
- Characteristic attribute combinations are learned Cylance was pioneering this

Evasion

Generic Evasion - File Formats

- Renaming of the Malware, e.g., from .exe to .hex
  - Requires social engineering to rename it back and execute it
  - Renaming is quite «normal» since email filters often block .exe or other «problematic» files => Mainly to circumvent blacklisting of file extensions
- To scan a file and to know what detection mechanisms to apply, an anti-virus system must «understand» many different file formats
  - Analysis of file formats of executables and data files (e.g., PDF, GIF)
- Vulnerabilities in those decoders has been an attack vector in the past
  - E.g., 03/2016: McAfee Enterprise antivirus could be disabled with a specifically crafted file
  - Code for less common file formats is likely to be less «mature» and tested
- Many executable file formats in Windows (e.g., *lnk, pif, wsh)*
  - lnk: Windows shortcut file simply links to a program, may contain parameters allowing for the execution of potential malicious code, e.g., by linking to cmd.exe /C <command>.

So how much can we trust our antivirus software?

Security researcher Joxean Koret believes that they're riddled with bugs. He highlighted some of them at a presentation during the SysScan 360 security conference in 2014, and said that he'd found security flaws in 14 antivirus products.

https://www.syscan360.org/slides/2014_EN_BreakingAVSoftware_JoxeanKoret.pdf

Executable file formats in Windows

- Executable binaries (com, exe, jar, msi, msp, shs)
- Executable scripts (bat, cmd, js, jse, vbe, vbs, wsf)
- Execution by built-in OS apps (chm, cpl, css, hlp, hta, msc, reg, scr)
- Executable reference files (lnk, pif, wsh)

File type Description Executed by

File type	Description	Executed by
bat	DOS batch file	shell32.dll
chm	HTML Help Compiled Help File	hh.exe
cmd	Command File	shell32.dll
com	DOS command file	shell32.dll
cpl	Windows Control Panel Extension	rundll32.exe
css	Hypertext Cascading Style Sheet	shell32.dll
exe	Executable file	(direct)
hlp	Windows Help File	shell32.dll
hta	Hypertext Application	mshta.exe
jar	Java Archive	JRE
js	JavaScript Source Code	WScript.exe
jse	JScript Encoded Script File	WScript.exe
lnk	Windows Shortcut File	rundll32.exe
msc	Mgmt Console Snap-in Control File	mmcbase.dll
msi	Windows Installer File	msiexec.exe
msi	Windows Installer File	msiexec.exe
pif	Windows Program Information File	(direct)
reg	Registry Data File	regedit.exe
scr	Windows Screen Saver	rundll32.exe
shs	Shell Scrap Object File	shscrap.dll
vbe	VBScript Encoded Script File	WScript.exe
vbs	VBScript Script File	WScript.exe
wsf	Windows Scripting File	WScript.exe
wsh	Windows Script Host Settings File	WScript.exe

Generic Evasion - Compression

- Compressed files are «normal» and can hardly be blocked
- Compression is time consuming => AV must decompress to analyse the file
- AV gateways in the network might skip decompression and let such files either pass or block them (setting)
  - for large files
  - for nested files (ZIP Bomb: DoS potential)
  - in times of high load

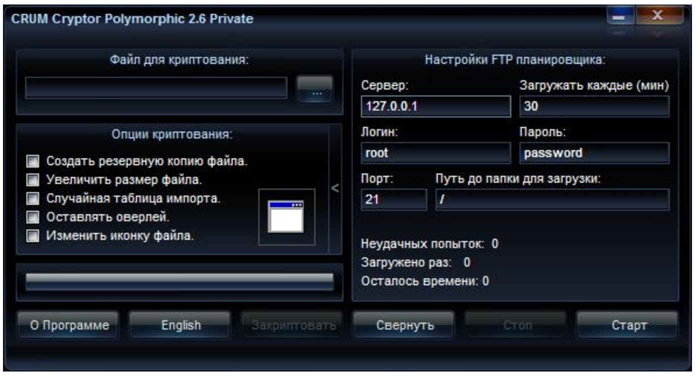
  => default should be detect and block in all three cases
- Password protected (and encrypted) compressed files
  - AV can't inspect the content of the file
  - Denying such files is often not an option since this is used as an easy way to protect content sent over the Internet (e.g., email or http)

A **zip bomb** is usually a small file for ease of transport and to avoid suspicion. However, when the file is unpacked, its contents are more than the system can handle. An example of a zip bomb is the file *42.zip*, which is a zip file consisting of 42 kB of compressed data, containing five layers of nested zip files in sets of 16, each bottom layer archive containing a 4.3 GB file for a total of 4.5 PB (peta bytes) of uncompressed data. This file is still available for download on various websites across the Internet. In many anti-virus scanners, only a few layers of recursion are performed on archives to help prevent attacks that would cause a buffer overflow or an out-of-memory condition, or exceed an acceptable amount of program execution time. Zip bombs often (if not always) rely on repetition of identical files to achieve their extreme compression ratios. Dynamic programming methods can be employed to limit traversal of such files, so that only one file is followed recursively at each level, effectively converting their exponential growth to linear.

File type	**Description**		
7z	7-Zip Compressed File	rar	RAR Compressed Archive
ace	Ace Compressed File	rev	RAR Recovery Volume File
arj	ARJ Compressed Archive	tar	Tape Archive File
bz	Bzip UNIX Compressed File	taz	.TAR.Z Compressed File
bz2	Bzip 2 UNIX Compressed File	tbz	BZIP2 Compressed TAR
cab	Cabinet File	tbz2	BZIP2 Compressed TAR
gz	Gzip Compressed Archive	tgz	UNIX Tar File Gzipped
img	Disk Image	uu	Uuencoded File
iso	ISO-9660 CD Disc Image	uue	Uuencoded File
lha	LHA Compressed Archive File	xxe	Xxencoded File
lzh	LZH Compressed Archive File	z	UNIX Compressed Archive File
r00-r29	RAR Split Compressed Archive	z00-	ZIP Split Compressed Archive
		zip	ZIP Compressed Archive

Signature Evasion - Polymorphism (1)

- Polymorphism - polymorphic code is code that uses a polymorphic engine to mutate while keeping the original function of the code (its semantics) the same
- For self-propagating malware, mutation engine is bundled with it, for other malware, the samples are mutated before their distribution
- Common Methods include (custom-made) cryptors and packers
- Available as tools and as-a-services



Source: https://blog.malwarebytexs.org/threat-analysis/2014/03/malware-with-packer-deception-techniques/

How to hide Meterpreter shellcode in executables and why AV nowadays can still detect it:

http://www.sevagas.com/?Hide-meterpreter-shellcode-in

How to really evade AV (might need modification after some time…):

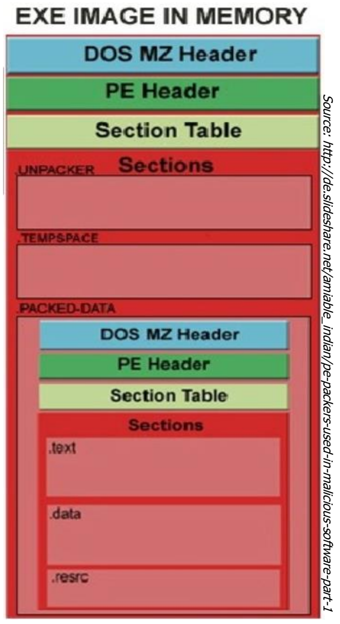
Emeric Nasi, Bypass Antivirus Dynamic Analysis - Limitations of the AV model and how to exploit them , 2014
http://www.sevagas.com/IMG/pdf/BypassAVDynamics.pdf

To learn about encrypting code segments and having decryptors for it:

http://www.sevagas.com/Code-segment-encryption

Signature Evasion - Polymorphism (2)

- Packers/cryptors contain a packed/encrypted part (payload) and a unpacking/decryption routine (stub) to extract+execute the payload
  - Antiviruses will attempt to generate signatures to match the stub's
  - Mutation of stub to evade signatures
    - Junk instructions that don't alter the semantics
    - Replacing instructions with others that do the same
- Example of a cryptor:
  - https://www.stringencrypt.com/



Memory protection and self-modifying code

One of the features enabled by the use of paging is the ability to enforce memory protection. Each page can specify restrictions to which operations are allowed to be performed on memory of the respective page.

In the context of runtime code modification, memory protection is of special importance as memory containing code usually does not permit write access, but rather read and execute access only. A prospective solution thus has to provide a means to either circumvent such write protection or to temporarily grant write access to the required memory areas.

As other parts of the image are write-protected as well, memory protection equally applies to approaches that modify non-code parts of the image such as the Import Address Table. That's why the call to VirtualProtect is neccessary when patching the IAT Programs using runtime code modification often do not restrict themselves to *changing* existing code but rather generate *additional* code. Assuming Data Execution Prevention has been enabled, it is thus vital for such approaches to work properly that any code generated is placed into memory regions that grant execute access. While user mode implementations can rely on a feature of the RTL heap (i.e. using the HEAP_CREATE_ENABLE_EXECUTE when calling RtlCreateHeap) for allocating executable memory, no comparable facility for kernel mode exist — a potential instrumentation solution thus has to come up with a custom allocation strategy.

Source: http://jpassing.com/2015/01/12/runtime-code-modification-explained-part-1-dealing-with-memory/

Signature (~Heuristics/Behavior) Evasion - Metamorphism

- Metamorphism: metamorphic code is similar to polymorphic code but the whole binary including the metamorphic engine itself undergoes changes
- Idea: Translate their own binary code into an intermediate language, edit it and translate it back to machine code
- Example: A simple decryptor and two «realizations» of it:

1. mov R_1, len
2. mov R_2, beg
3. xor [R_2], key
4. add R_2, 4
5. sub R_1, 4
6. jnz step_3

⇨

```
00 PUSH 44554433
01 POP ESI
02 SUB EBX,EBX
03 ADD EBX, 124
04 XOR [ESI],d20b9a65
05 ADD ESI,4
06 SUB EBX,4
07 JZ $ + 2
08 JMP $ + f0
```

```
00 XOR EDI,EDI
01 LEA EDI,[EDI+124]
02 PUSH 44554433
03 POP ESI
04 MOV EDX,[ESI]
05 NOT EDX
06 AND EDX,d75d40bc
07 AND [ESI],28a2bf43
08 OR [ESI],EDX
09 ADD ESI,4
10 SUB EDI,4
11 JZ $ + 2
12 JMP $ + e4
```

Source: www.cs.sjsu.edu/faculty/stamp/papers/topics/topic1/teja.pdf

Details of an example of such a metamorphic engine making use of the LLVM IR Bytecode from research can be found here:

Teja Tamboli, Metamorphic Code Generation from LLVM IR Bytecode, 2013

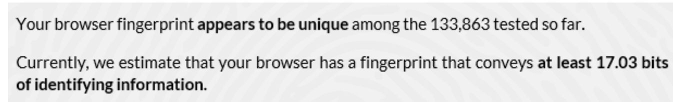
www.cs.sjsu.edu/faculty/stamp/papers/topics/topic1/teja.pdf

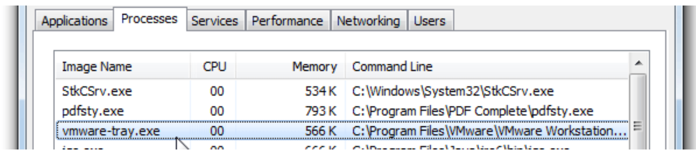
Evasion - Heuristics/Behavioral Detection

- The easiest way to bypass static heuristic analysis is to ensure that all the malicious code is hidden (e.g., by packing/encrypting it)
- The easiest way to bypass dynamic heuristic analysis is to hide the code in a way that the emulator is not able to unhide it (see demo)
  - If known packers/cryptors are used, it can usually unpack/decrypt the it
- For behavioral detection, the same applies as for the heuristics based detection
- In addition, for approaches monitoring the behavior at runtime (natively or in a sandbox), the following might be used to evade detection:
  - Detecting the behavioral monitoring or sandbox and behave differently
  - Outwait the behavioral monitoring by the AV solution
    - Difficult in case of endpoint security solution with always-on detection

- Fingerprint the environment like e.g., done in the browser fingerprinting project https://panopticlick.eff.org/
- Behave no-malicious if run in an analysis environment (or when specific security controls are in place)
- Virtualization-/Sandbox-specific techniques: Scan for registry entries, hardware, software (e.g., VMware Tools) or specific processes

=>Use «custom» virt. approaches or alter the footprint of existing ones





Evasion - Sandboxes

- Human interaction specific – Implement some form of «CAPTCHA»
  - Wait for a certain amount of mouse clicks before continuing decryption and execution of «malicious» part (e.g., Trojan.APT.BaneChan)
  - Show dialog boxes and ask the user for a decision
  - Assess user-activities in general => is this the machine of a real user?

  => Arms race: «detecting a human» vs. «emulating a human»

- Time-specific techniques
  - Time triggers: Malicious code executes when a time condition is met (e.g., on April 1st ☺)
  - Extended sleep – Outwait the analysis sandbox

  => Sandboxes may try to modify the sleep duration, or increase the analysis period

Extended Sleep – "To achieve scalability, most sandbox systems only run a limited and finite range of virtual machines which implies that if malware does not have any reaction inside the sandbox, no malicious activity will be detected. Malware authors employ sleep calls to bypass the automated malware analysis sandbox systems" (Lakhani, 2015).

More details and information on Sandbox evasion:

- *Emeric Nasi, Bypass Antivirus Dynamic Analysis - Limitations of the AV model and how to exploit them, 2014*
  http://www.sevagas.com/IMG/pdf/BypassAVDynamics.pdf

Evasion Example – Combining Multiple Free Packers

Table 4.9: Scan results of the meterpreter reverse shell sample protected with multiple packers.

Packer	Bypasses (Suspicious)	Engines bypassed	Engines suspicious[1]
Msf Evasion	4(2)	ClamAV, EScan, FProt, McAfee	Avast, Comodo
PeCloak	6(3)	ClamAV, Comodo, FProt, Ikarus, McAfee, Sophos	Avast, EScan, FSecure
Petite	3(2)	ClamAV, EScan, FProt	Avast, FSecure,
Themida	8(3)	BitDefender, ClamAV, Comodo, EScan, FProt, Ikarus, McAfee, Sophos	Avast, Avg, FSecure
VMProtect	7(3)	ClamAV, Comodo, EScan, FProt, FSecure, Ikarus, McAfee, Sophos	Avast, BitDefender, FSecure
UPack	1(4)	Sophos	Avg, ClamAV, Comodo, Ikarus
UPX	1(2)	FProt	Avast, Avg
Obsidium	7(3)	ClamAV, Comodo, EScan, FProt, Ikarus, McAfee, Sophos	Avast, BitDefender, FSecure
PeLock	7(1)	ClamAV, Comodo, EScan, FProt, Ikarus, McAfee, Sophos	Avast
Petite	3(1)	ClamAV, EScan, FProt	Avast
Smart Packer Pro X	10(1)	Avast, Avg, BitDefender, ClamAV, Comodo, EScan, FProt, Ikarus, McAfee, Sophos	FSecure
Enigma	7(3)	BitDefender, ClamAV, Comodo, EScan, FProt, McAfee, Sophos	Avast, Avg, Ikarus

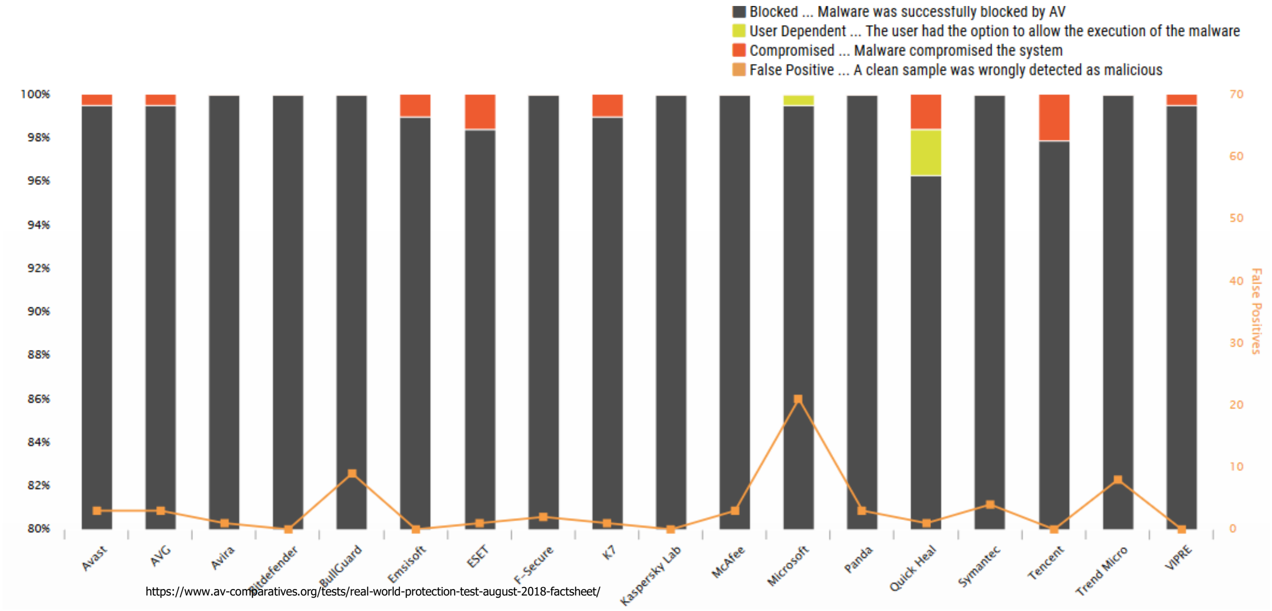
Source: Daniel Jampen, Analysis of Anti-Virus Evasion Techniques and their Use in APTs, ZHAW MSE Project Thesis II, 2019

How Good is Anti-Virus?

Next Generation «Signatures» - How Good Are They?

- Difficult to say because…
  - Next generation vendors favor presenting or creating their own «tests»
  - Third party testing of next generation products are rare
    - Getting licenses for such tests seems to be difficult according to AV-Comparatives
- AV testing is a difficult business:
  - Result heavily depend on the methodology and sample set used
  - Test labs tend to follow the Anti-Malware Testing Standards Organization (AMTSO) recommendations
  - AMTSO has several documents related to testing a given protection product, from standard anti-virus to newer endpoint defense (http://www.amtso.org/)
  - Tests are done by independent labs like AV-Test, AV-Comparatives or MRG Effitas but are often sponsored by AV product vendors
    - http://www.csoonline.com/article/3167236/security/cylance-accuses-av-comparatives-and-mrg-effitas-of-fraud-and-software-piracy.html

Real-World Protection Test Results, August 2018



 62

The results are based on the test set of **193** live test cases (malicious URLs found in the field), consisting of working exploits (i.e. drive-by downloads) and URLs pointing directly to malware. Thus exactly the same infection vectors are used as a typical user would experience in everyday life. The test-cases used cover a wide range of current malicious sites and provide insights into the protection given by the various products (using **all** their protection features) while surfing the web. Every month (from February to June and from July to November) we update the charts on our website showing the protection rates of the various tested products over the various months. The interactive charts can be found on our website.

Disclaimer for the test: *We would like to point out that while some products may sometimes be able to reach 100% protection rates in a test, it does not mean that these products will always protect against all threats on the web. It just means that they were able to block 100% of the widespread malicious samples used in a test.*

Source: *https://www.av-comparatives.org/tests/real-world-protection-test-august-2018-factsheet/*

Anti-Virus Performance – Retrospective Test

AV comparatives	Blocked	User dependent[3]	Compromised	Proactive Protection Rate	False Alarms	Cluster
Bitdefender	1448	-	15	**99%**	few	1
F-Secure	1358	3	102	**93%**	many	1
eScan	1354	-	109	**93%**	many	1
Kaspersky Lab	1343	-	120	**92%**	Few	1
BullGuard	1259	129	75	**90%**	many	1
ESET	1253	-	210	**86%**	very few	1
Emsisoft	777	667	19	**76%**	many	2
Avast	985	-	478	**67%**	very many	2
Lavasoft	781	-	682	**53%**	many	3
Microsoft	772	-	691	**53%**	very few	3
Fortinet	742	-	721	**51%**	few	3
ThreatTrack	682	-	781	**47%**	many	-

Why are retrospective tests with modern anti-virus products usually not possible anymore?

- Heuristics and behavioral protection (offline) only
- 1463 malware samples appearing for the first time shortly after the freezing date (3rd March 2015)

Tests ONLY heuristics, generic detection and behavioural protection (offline). Additional stuff like cloud-technology sending and analysing unknown files in a sandbox in the cloud are not included.

Anti-Virus Performance - Summary

- No anti-virus software offers full protection from malware
- Some do «significantly» better than others but results must be considered with care - these are snapshots and depend on the test setup (malicious and benign)
- It is quite likely that you get infected by malware at some point if anti-virus is your only line of defense