Malicious Code
Lesson Introduction

- Reasons attackers use malware: automation, scalability, and deniability.
- Attackers release malicious programs on the **Internet** and let them **spread**
- Overview of **malware**

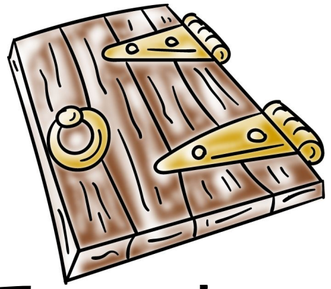
Types of Malicious Software (Malware)

- Needs host program

- Independent

Types of Malicious Software (Malware)

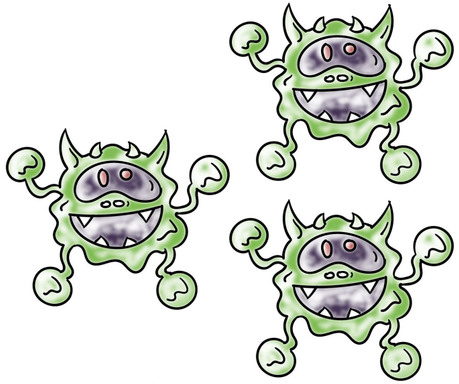
- Needs host program:



Trap doors

Logic bombs

Trojan horses

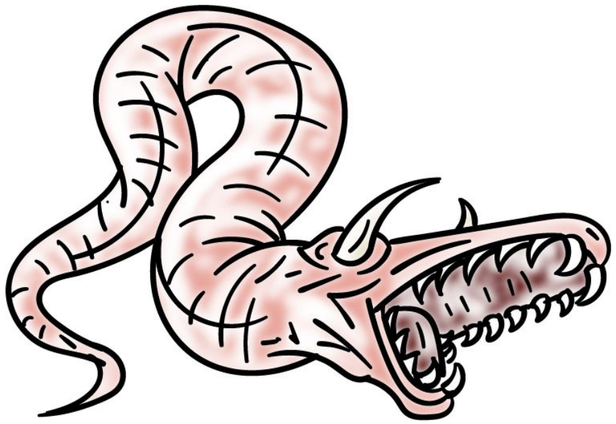


Viruses

Browser plug-ins,
extensions, scripts

Types of Malicious Software (Malware)

- Independent:



Worms

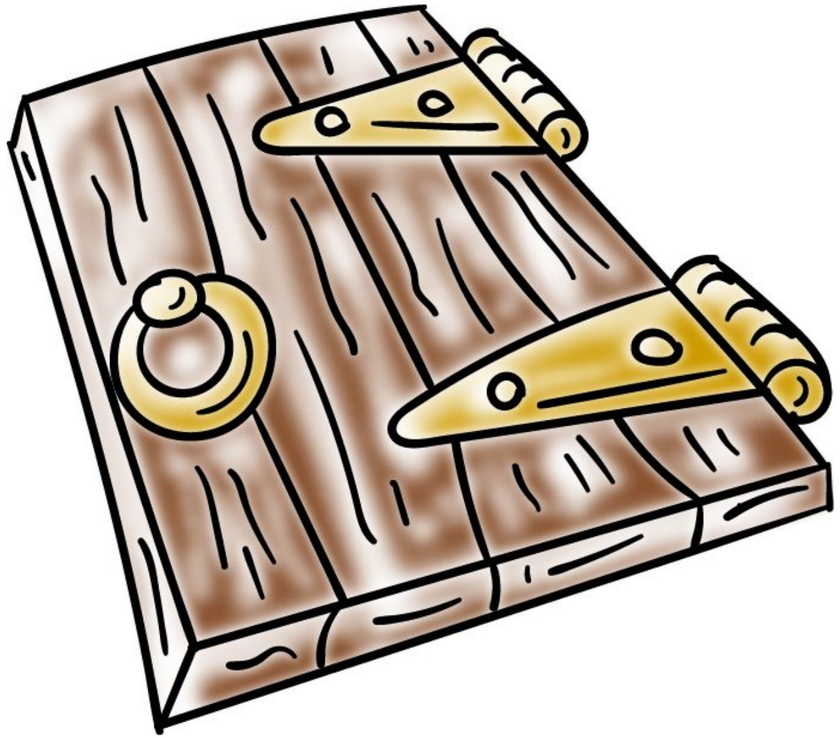
Botnet

APTs

Trap Doors

- A secret entry point to a program or system.

- Typically works by recognizing some special sequence of inputs or special user ID.



Logic Bombs

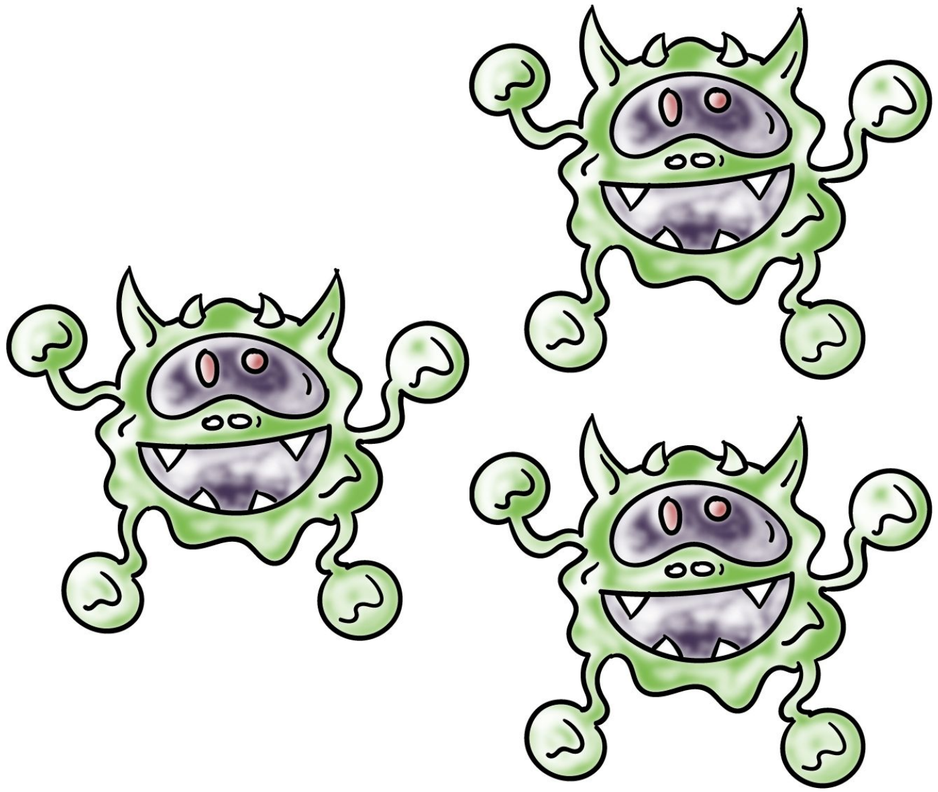
- Embedded in some legitimate program
- "Explode" or perform malicious activities when certain conditions are met.

Trojan Horses

- Hidden in an apparently useful host program

- Performs some unwanted/harmful function when the host program is executed

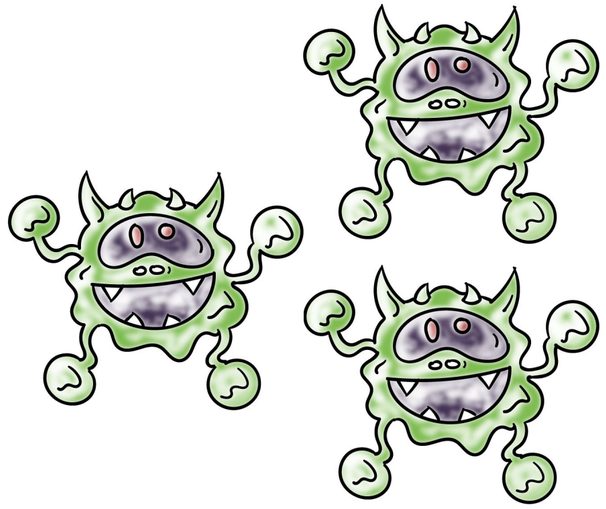
Viruses



- Infect a program by modifying it
- Self-copy into the program to spread

Viruses

Four Stages of Viruses



Dormant Phase

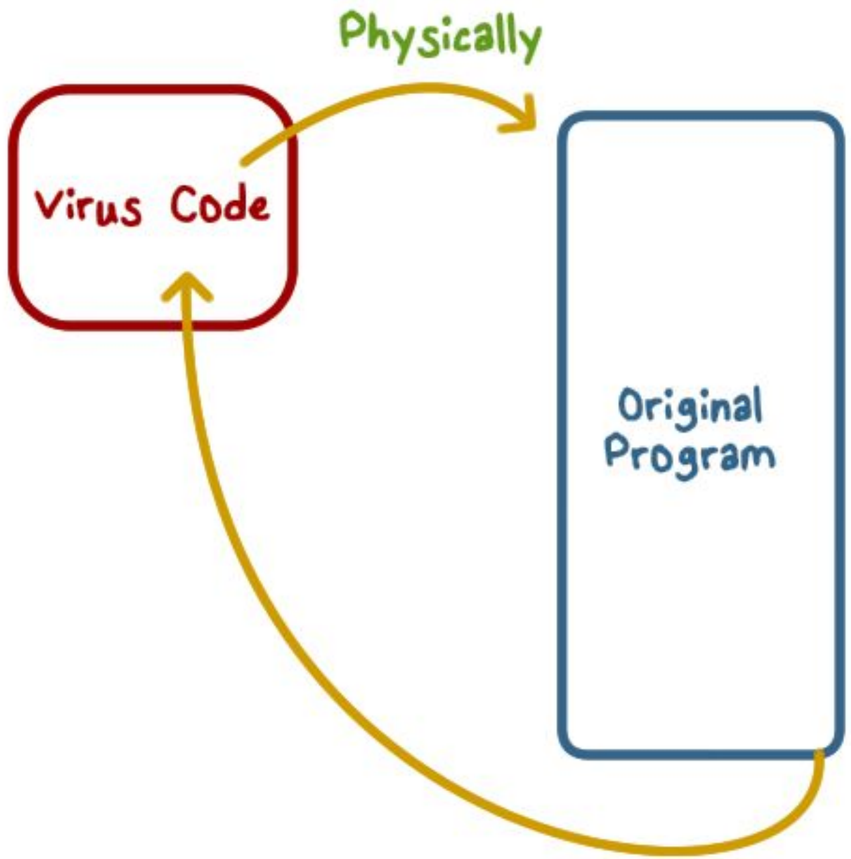
Propagation Phase

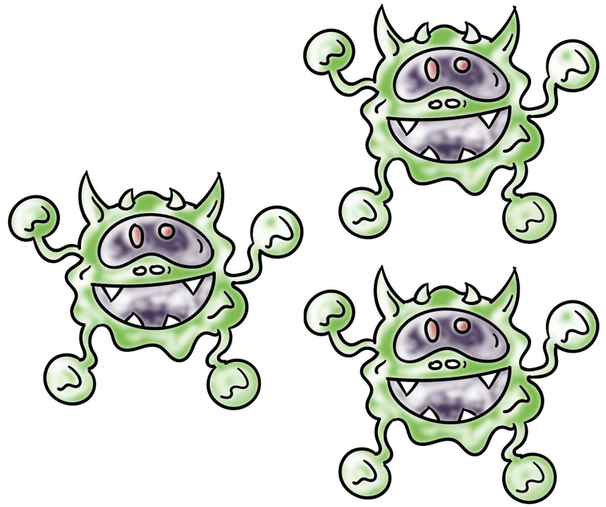
- E.g. attachment to email

Triggering Phase

Execution Phase

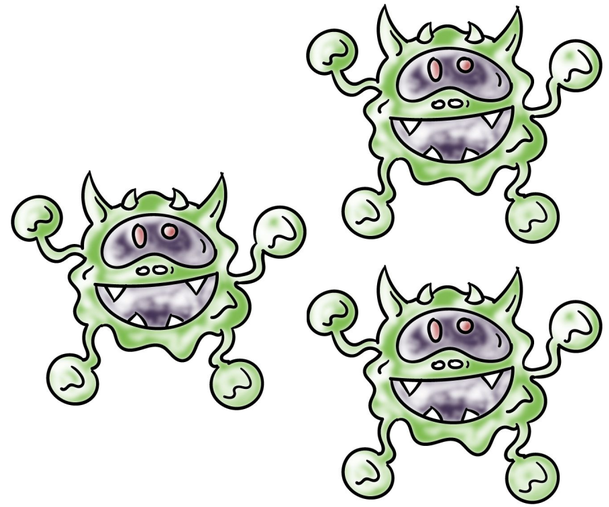
Virus Structure





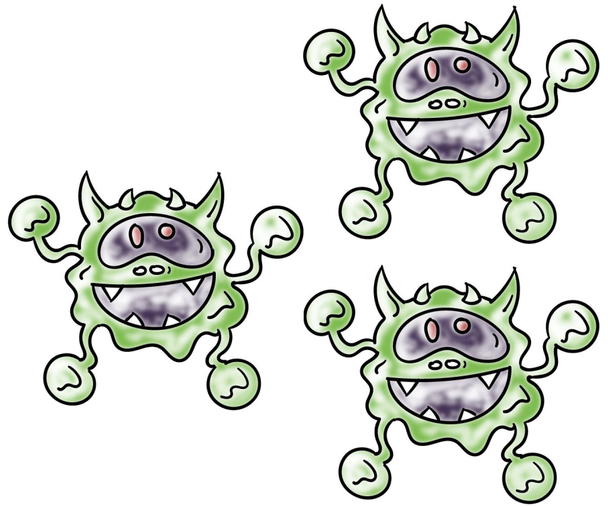
You run the malicious code when you enter a program or exit it.

Virus Structure



- **First line:** go to "main" of virus program
- **Second line:** a special flag (infected or not)
- **Main:**
  - Find uninfected programs - infect them
  - Do something damaging to the system
  - "Go to" first line of the host program - do normal work
- **Avoid detection** by looking at size of program
  - Compress/decompress the host program

Types of Viruses



- Parasitic virus: scan/infect programs
- Memory-resident virus: infect running programs
- Macro virus: embedded in documents, run/spread when opened
- Boot sector virus: run/spread whenever the system is booted

Infect the system code. Whenever you boost your system, the virus works.

- Polymorphic virus: encrypt part of the virus program using a randomly generated key

Rootkit

- Resides in operating systems
  - Modifies OS code and data structure
- Helps user-level malware
  - E.g., hide it from user (not listed in "ls" or "ps" command)

Rootkit

Inspect all files

FindFirstFile()

{ checkfile
FindNextFile()
repeat

Windows API

NTQueryDirectoryObject

kernel Native Interface

Rootkit filters call and results

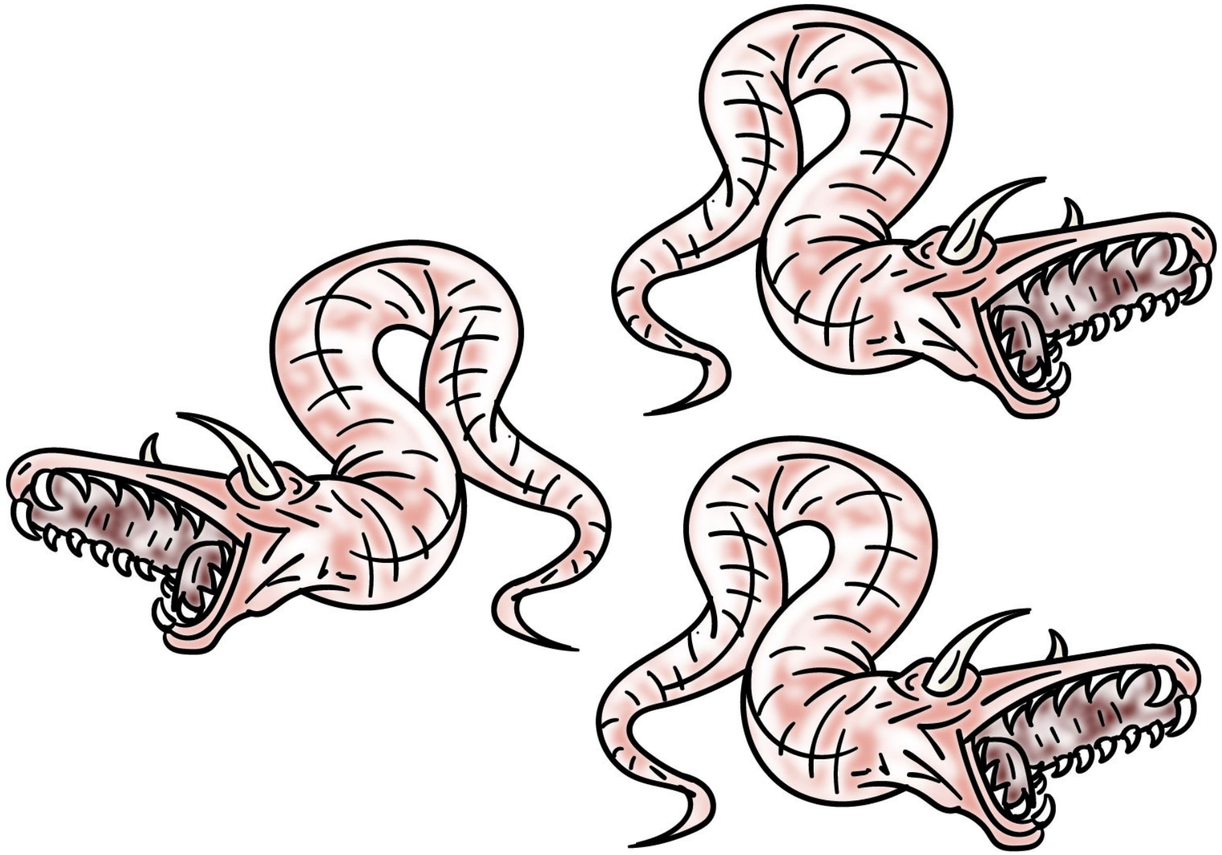
Device driver functions

Drivers

Worms

Also sometimes be called "Internet worms"

- Use network connections to spread from system to system

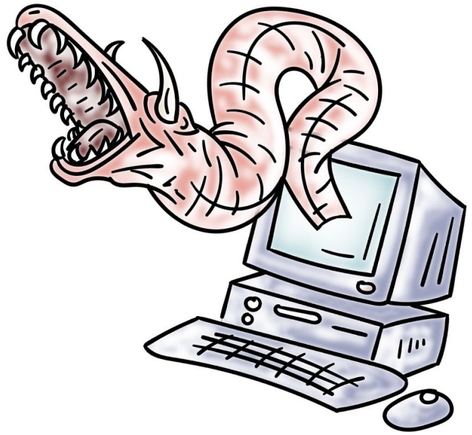


The Internet Worm

What it did:

- Determine where it could spread
- Spread its infection
- Remain undiscovered and undiscoverable

The Internet Worm

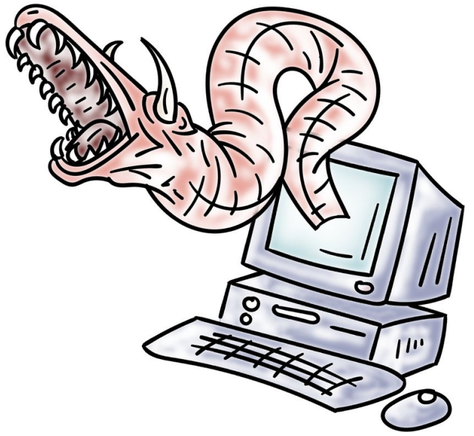


Effect:

Resource exhaustion - repeated infection due to programming bug

- Servers are disconnected from the Internet by system admin to stop the infection

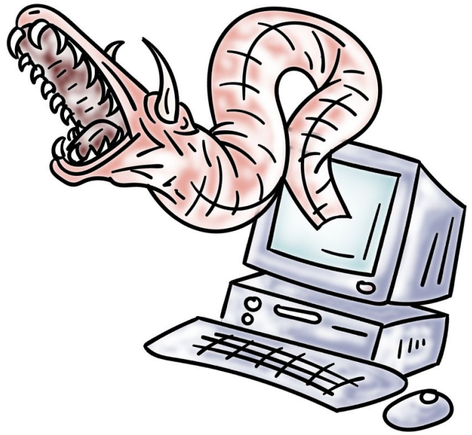
The Internet Worm



- Exploit security flaws
  - Guess password (encrypted passwd file readable)
  - fingerd: buffer overflow
  - sendmail: trapdoor (accepts shell commands)

- Spread
  - Bootstrap loader to target machine, then fetch
  - rest of code (password authenticated)

The Internet Worm



- Remain un-discoverable
  - Load code in memory, encrypt, remove file
  - Periodically changed name and process ID

- What we learned:
  - Security scanning and patching
  - Computer Emergency Response Team

Malware Prevention & Detection Approaches

- **Prevention:** Limit contact to outside world
- Detection and Identification
- Removal

Malware Prevention & Detection Approaches

4 Generations of antivirus software:

- Simple scanners: Use "signatures" of known viruses
- Heuristic scanners: Integrity checking: checksum, encrypted has
- Activity traps
- Full-featured analysis: Host-based, network-based, sandboxing-based

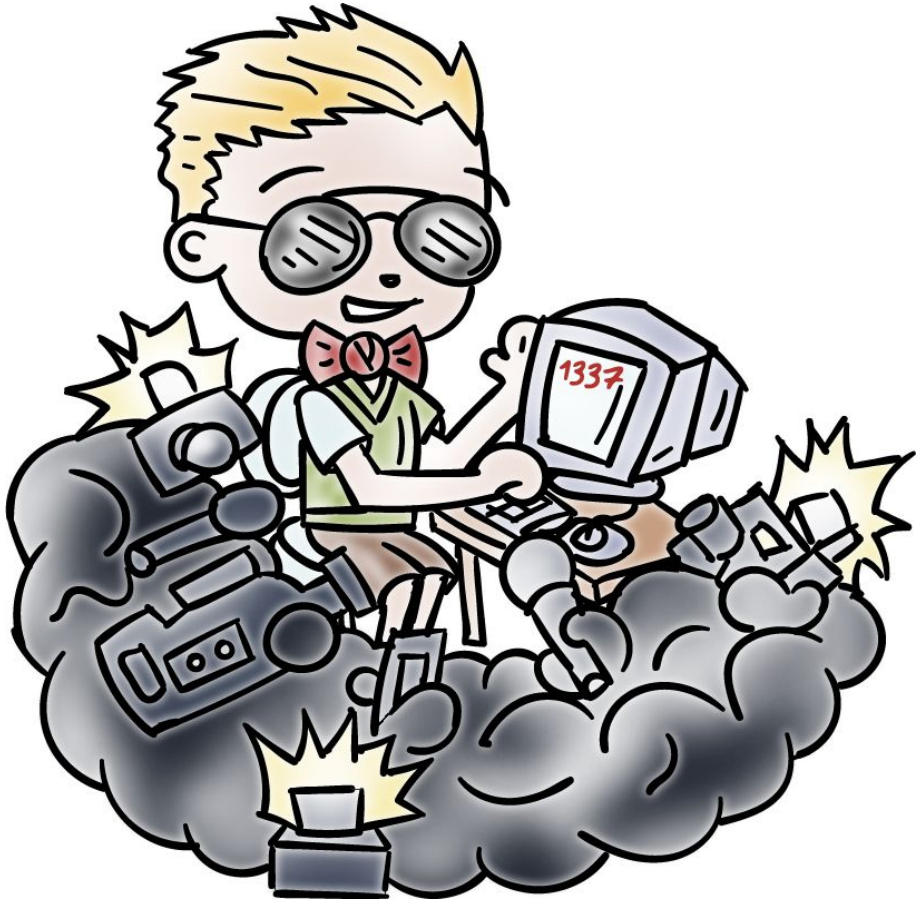
Treats the program as a blackbox, if it senses that a program tries to read you document and then send the document to somewhere outside, it considers this behavior as stealth.

A sandbox tries to entrap the damage caused by the malware. (everything done by a walware is restricted in the sandbox.

Modern Malware
Lesson Introduction

- What is “modern”
- Botnets and APTs
- Basic malware analysis and detection techniques

Past Malware



- In the past, often for "fame" and/or "fun"
  - E.g., defacing web pages
  - Fast and large-scale spreading

Modern Malware

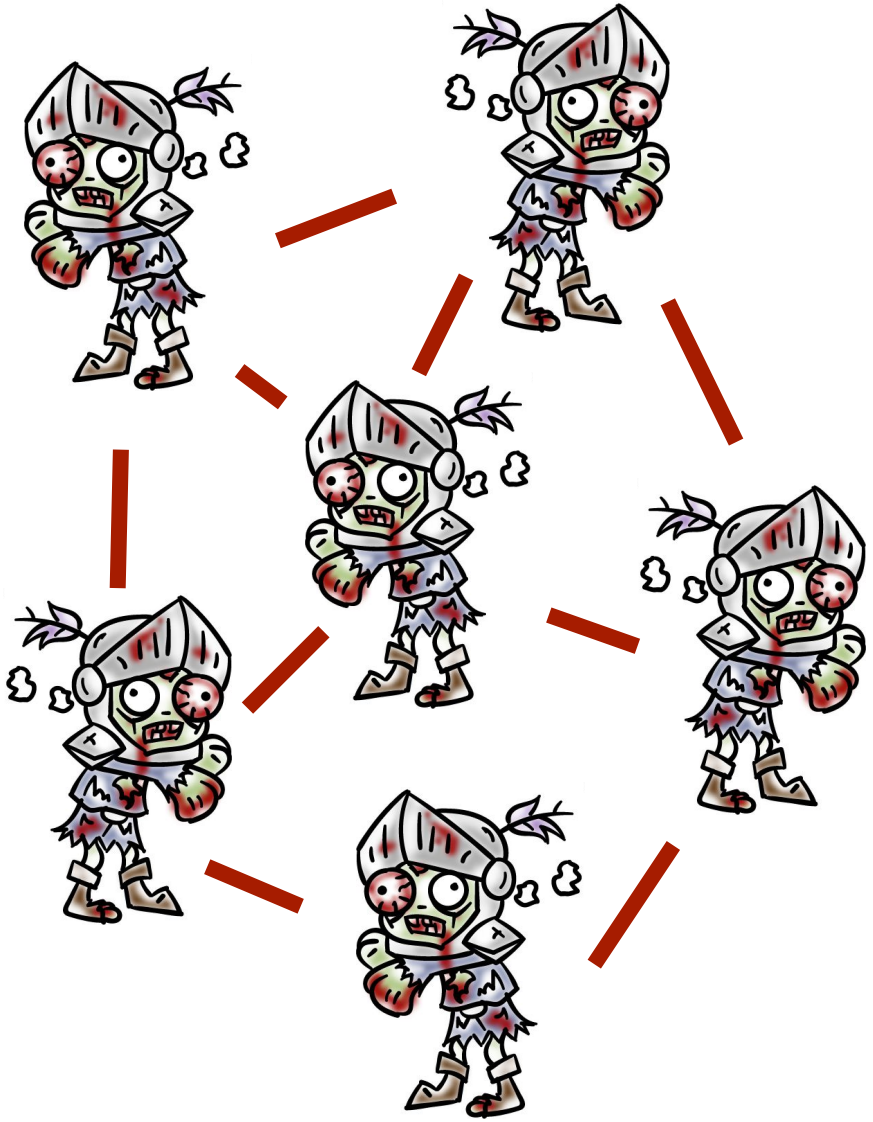
- Now, often for profit and political gains
- Technical sophistications based on the latest technologies
  - Efficiency, robustness, and evasiveness

Botnet

- **Bot (Zombie)**
  - A compromised computer under the control of an attacker
  - Bot code (malware) on the computer communicates with the attacker's server and carries out malicious activities per attacker's instructions

Botnet

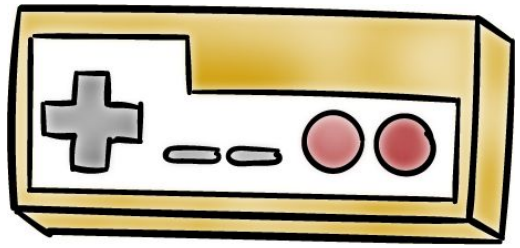
- Botnet
  - A network of bots controlled by an attacker to perform coordinated malicious activities
  - Key platform for most Internet-based attacks and frauds



Attacks and Frauds by Botnets

- Spam
- Distributed Denial of Service (DDOS) Attacks
- Clickfraud
- Phishing & Pharming

- Key Logging & Data/Identity Theft
- Key/Password Cracking
- ...



- Cheating in online games/polls
- Anonymized Terrorist & Criminal Communication

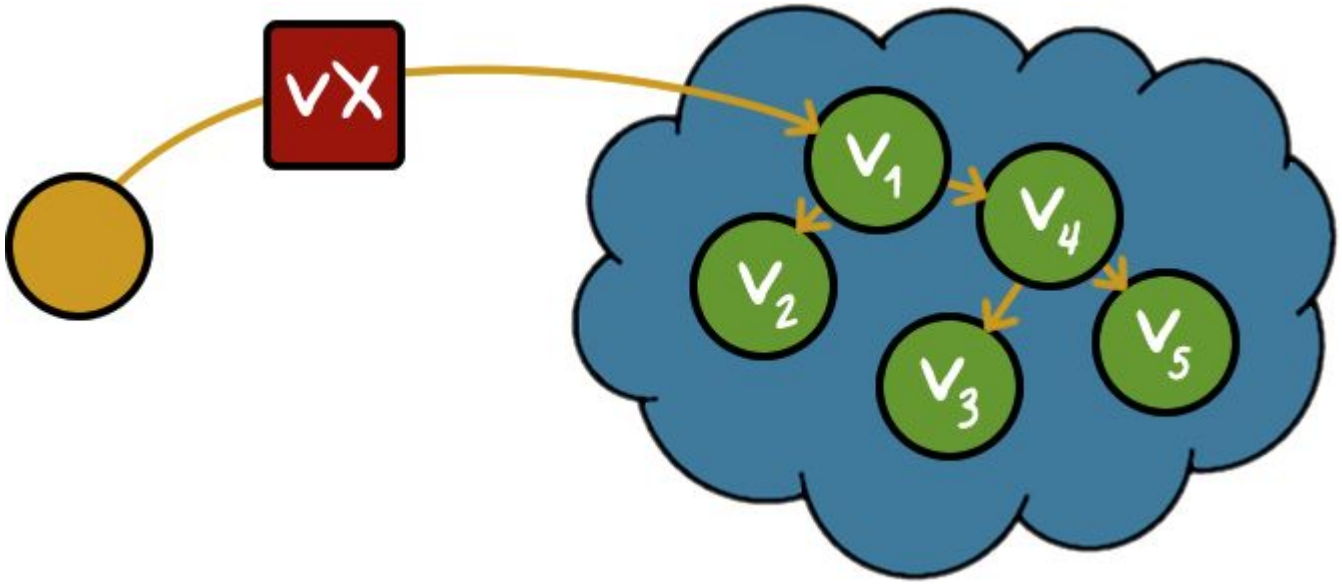
Botnet Command and Control

- Botnet is a network of compromised computers that the "botmaster" uses for malicious purposes

- There needs to be command & control (C&C) from the botmaster to the bots
  - Example: a bot reports to the botmaster its status, is directed to a site to download a malware (botcode) update, and/or receives instructions to spam/phish/DDoS, etc.

Botnet C&C Problem

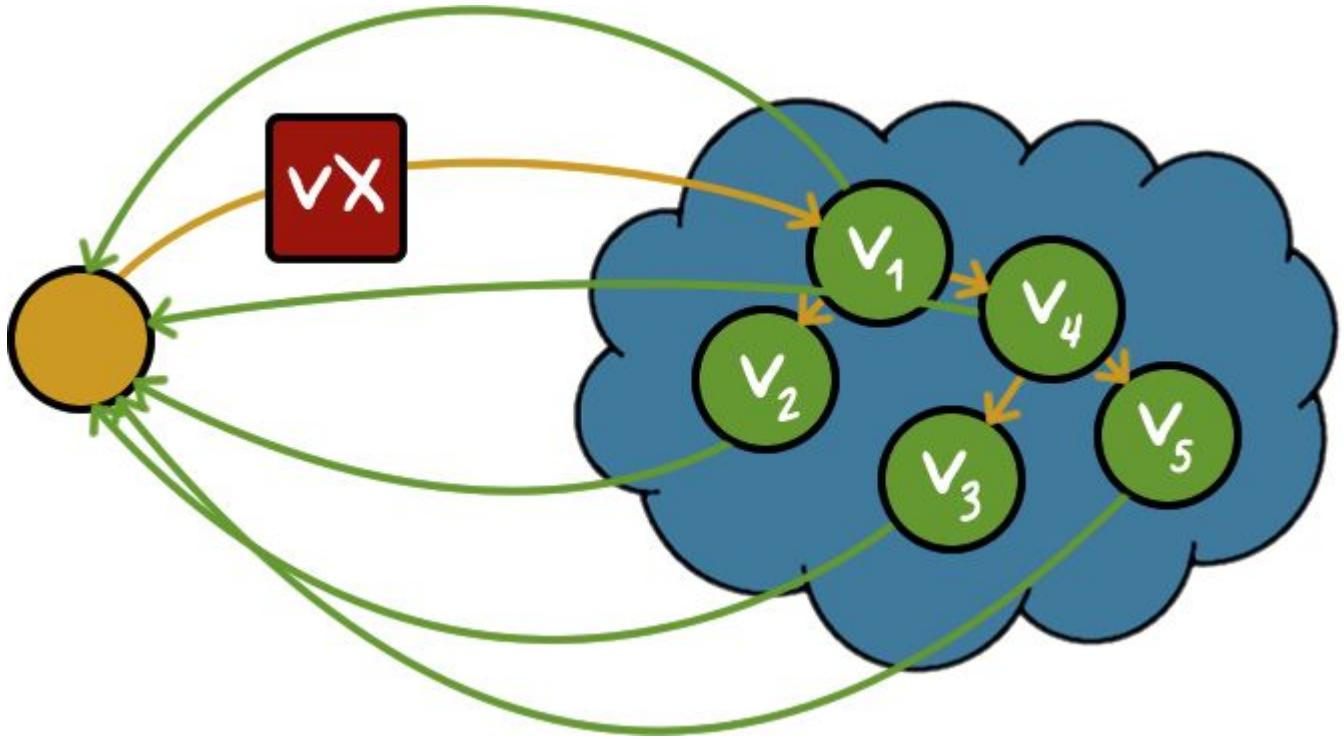
- Naively, we could have victims contact us...
  - Suppose we create malware (vx)
    - Download vx code; fiddle; compile
    - Uses email propagation/social engineering
  - We mail it...



Spreading is easy, but what if we want to use the compromised computers (victims)?

Botnet C&C- Naive Approach

- Naively, we could have victims contact us...



- Problems:
  - VX must include author's address (not stealthy)
  - Single rallying point (not robust)
  - VX has hard-coded address (not mobile)

Botnet C&C Design

- How can bots contact their master safely?
- Simple, naïve approach:
  - Victims contact single IP, website, ping a server, etc.
  - Easily defeated (ISP intervention, blackhole routing, etc.)
  - Still used by script-kiddies, first-time malware authors

Botnet C&C Design

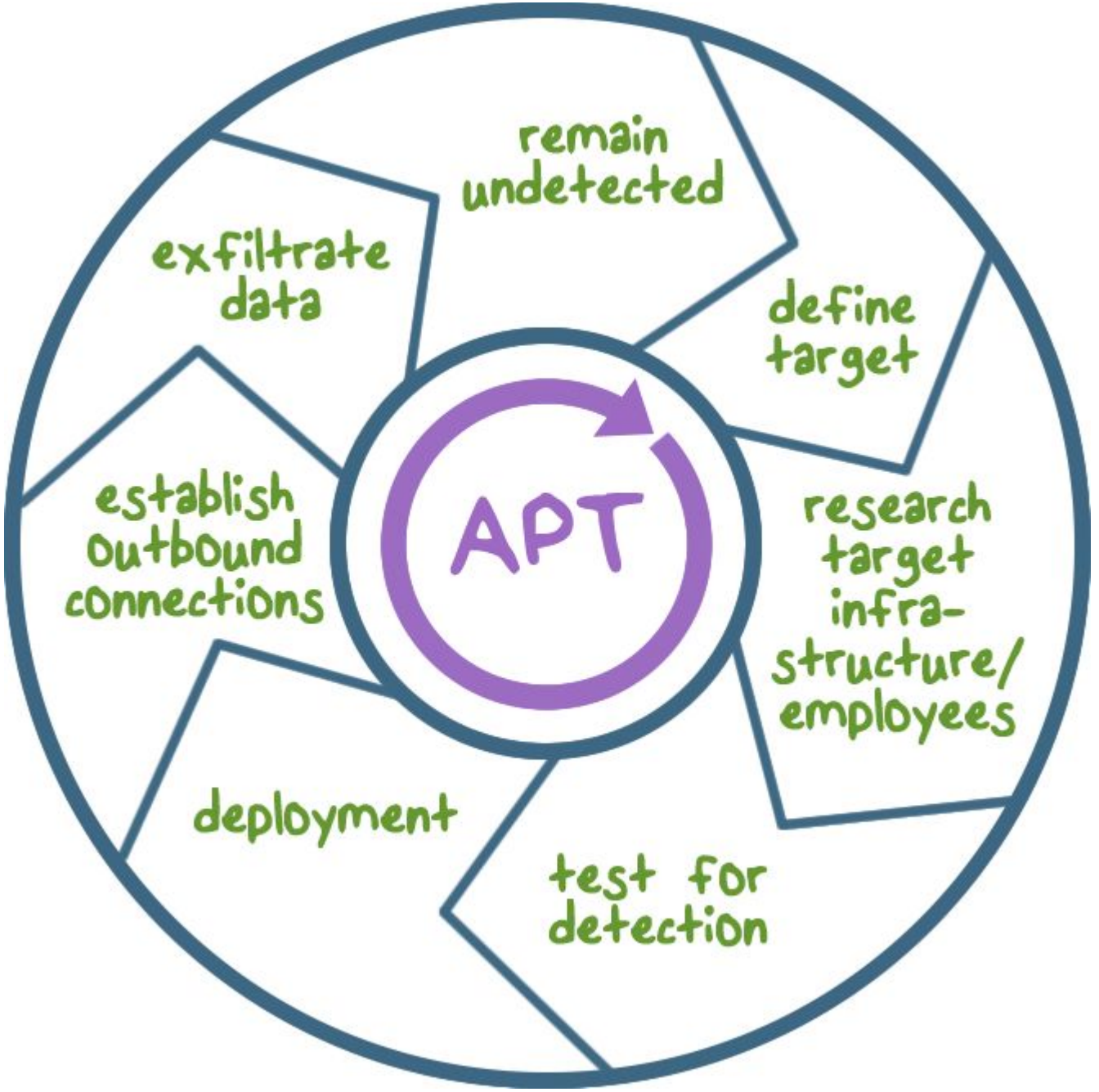
Design considerations:

- Efficient and reliable
  - Able to reach to a sizable set of bots within a time limit
- Stealthy
  - Hard to detect (i.e., blended with normal/regular traffic)
- Resilient
  - Hard to disable or block

Advanced Persistent Threat (APT)

- **Advanced:**
  - (Special) malware
  - Special operation and operators
- **Persistent:**
  - Long-term presence, multi-step, "low-and-slow"
- **Threat:**
  - Targeted at high-value organization and information

APT Lifecycle



APT Characteristics

- **Zero-day exploit** or a specially crafted malware
  - No readily available signature for its detection

APT Characteristics

- Social-engineering to trick even the most sophisticated users. Example:
  - First compromise core internal network control elements such as routers and web servers to learn about the valuable targets;
  - Then play man-in-the-middle (MITM) on the compromised routers/server to make social-engineering attacks very convincing, e.g., to even forge answers to challenge or inquiry by suspecting users

APT Characteristics

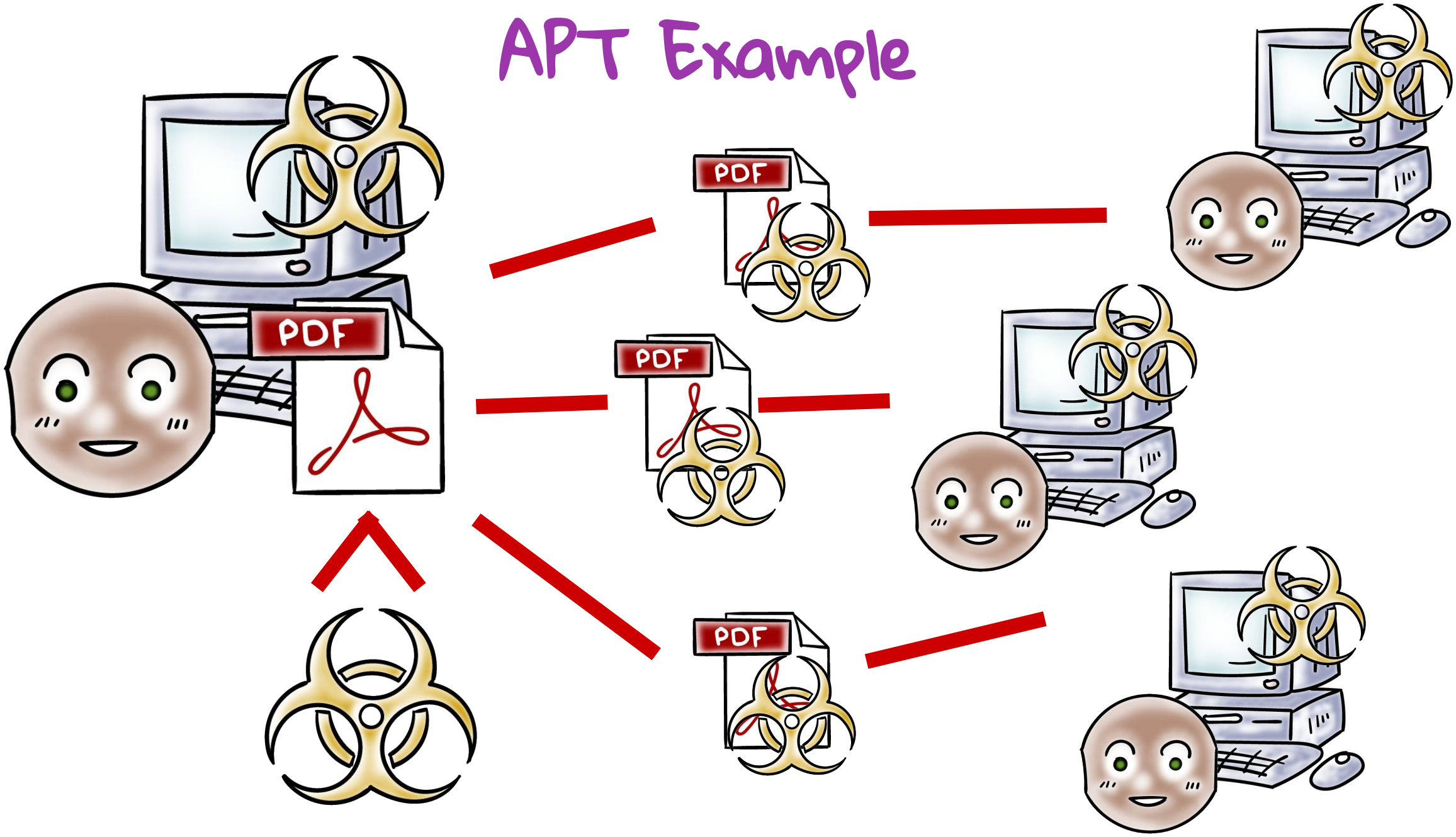
- Carry out its intended mission, such as data tampering and exfiltration, in a low-and-slow fashion to completely blend in with normal activities
- Example:
  - Acts only when the targeted user is authoring and sending a document
  - Make repeated, small incremental change to data to accomplish the eventual attack goal
  - Not detectable anomaly by existing approaches

APT Characteristics

- APT is a persistent operation that involves multiple deliberate steps over time, rather than a single attack act

- Example:
  - Employ a combination of steps to move laterally through the network and target only the necessary systems and users at each attack step

APT Example



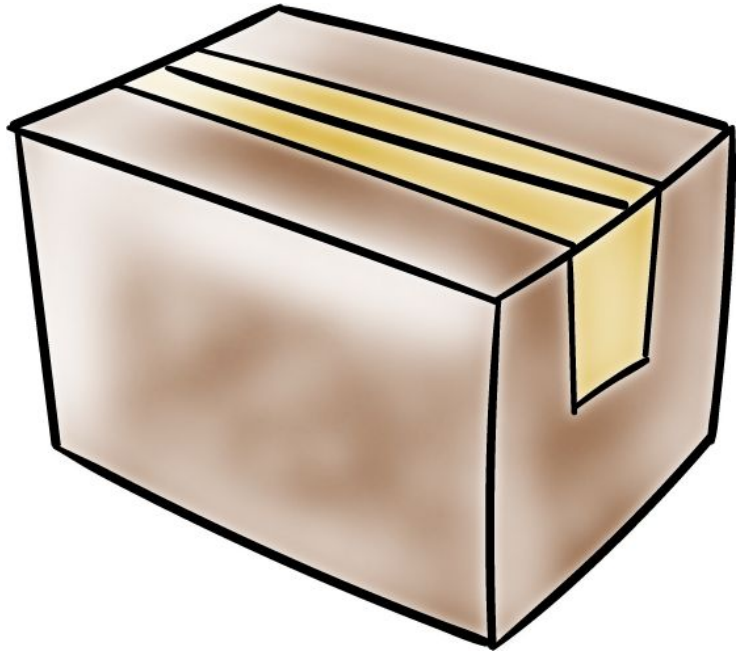
Malware Analysis

- Produce info for detection/response
- **Static Analysis:** Attempts to understand what a malware instance would do if executed Do not run the program.
- **Dynamic Analysis:** Attempts to Run the program and examine the behavior understand what a program does when executed
  - Different granularities
    - Fine-grained (e.g., automated unpacking)
    - Coarse-grained (e.g., system call tracing)

Malware Obfuscation

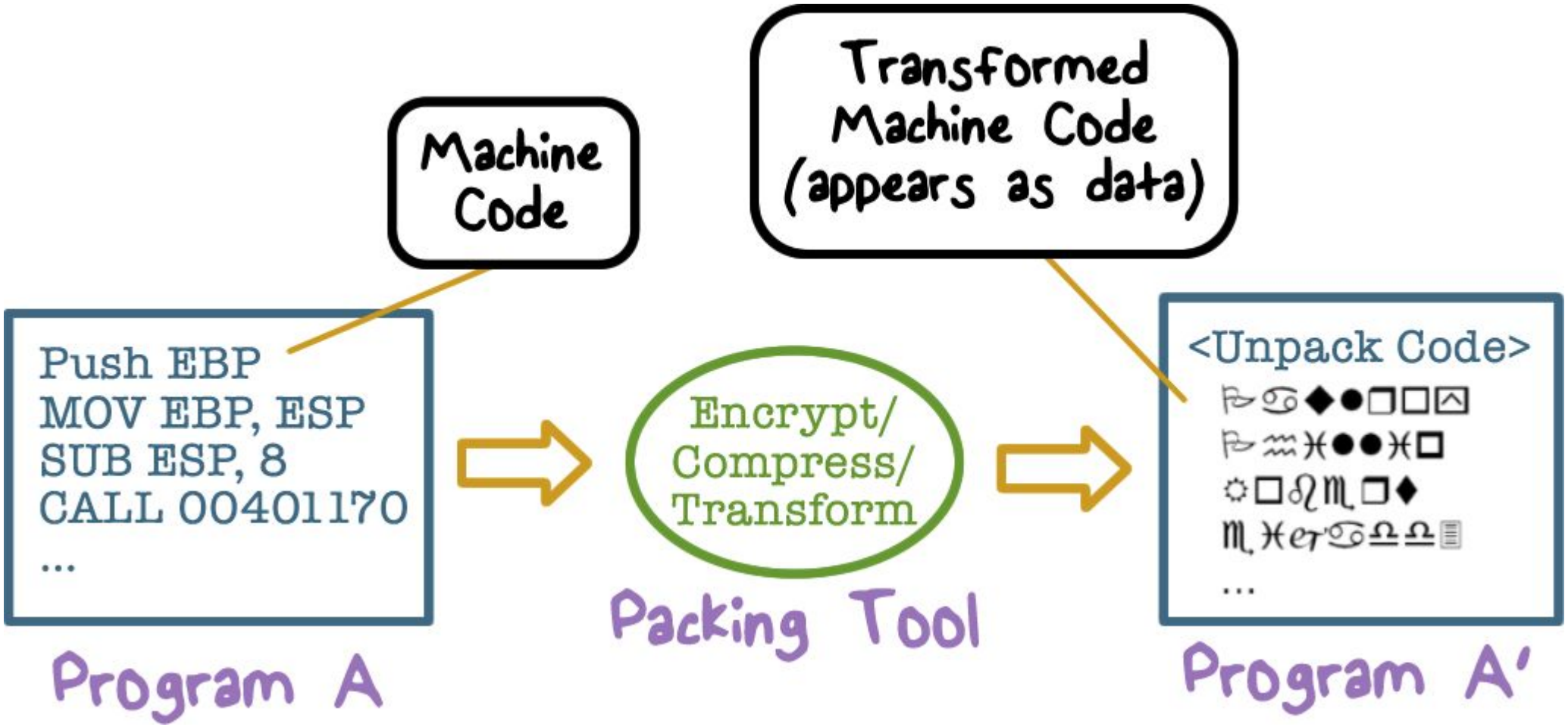
How to defeat static analysis: try to transform you self into something hard to identify

- **Packing:** a technique whereby parts or all of an executable file are compressed, encrypted, or transformed in some fashion

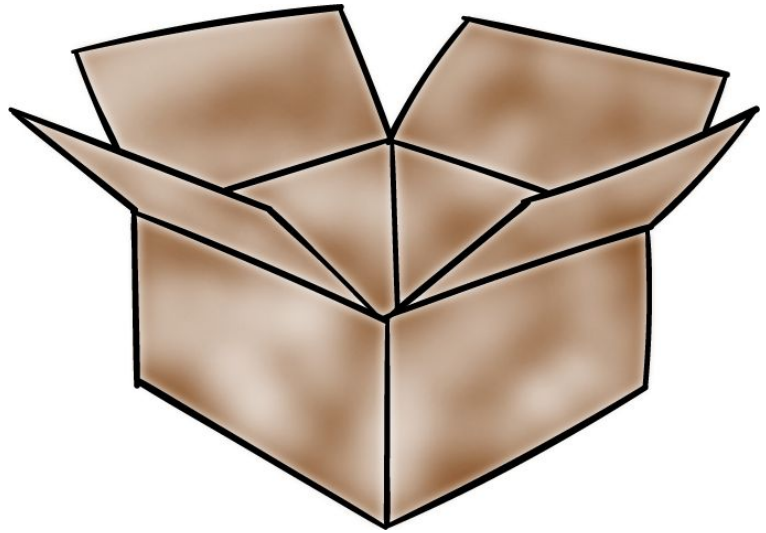


- Code that reverses the pre-runtime transformation is included in the executable

Malware Obfuscation



Unpacking



- Most modern malware comes packed
  - Thousands of packers, countless ways to obfuscate code
  - Volume of malware samples makes manual unpacking untenable
- Need for automated unpacking that does not require a priori knowledge
- Fine-grained tracing-based universal automated unpacking algorithms
  - =Detect the execution of code not in the static code model (i.e., the model of the packed program)

Modern Malware
Lesson Summary

- Botnets use command-and-control mechanisms
- APTs can hide tracks, and lay “low and slow”
- Need network monitoring, and static and dynamic analysis of malware

Malicious Code
Lesson Summary

Host-dependent malware:

- trap doors
- logic bombs
- trojan horses and
- viruses

Host-independent malware:

- Worms

PlatPal: Detecting Malicious Documents with Platform Diversity

Meng Xu and Taesoo Kim

Georgia Tech

Presented at *the 2017 USENIX Security Symposium (Security'17)*

Adobe Components Exploited

137 CVEs in 2015

227 CVEs in 2016

Element parser

JavaScript engine

Font manager

Customized font. To some extent a font is executable as well.

System dependencies

Maldoc Formula

Flexibility of doc spec

+

A large attack surface

+

Less caution from users

=

More opportunities to profit

Battle against Maldoc - A Survey

Category	Focus	Work	Year	Detection
Static	JavaScript	PJScan	2011	Lexical analysis
	JavaScript	Vatamanu et al.	2012	Token clustering
	JavaScript	Lux0r	2014	API reference classification
	JavaScript	MPScan	2013	Shellcode and opcode sig
	Metadata	PDF Malware Slayer	2012	Linearized object path
	Metadata	Srndic et al.	2013	Hierarchical structure
	Metadata	PDFrate	2012	Content meta-features
	Both	Maiorca et al.	2016	Many heuristics combined
Dynamic	JavaScript	MDScan	2011	Shellcode and opcode sig
	JavaScript	PDF Scrutinizer	2012	Known attack patterns
	JavaScript	ShellOS	2011	Memory access patterns
	JavaScript	Liu et al.	2014	Common attack behaviors
	Memory	CWXDetector	2012	Violation of invariants

Highlights of the Survey

Prior works rely on

- External PDF parsers → *Parser-confusion attacks*
- Machine learning → *Automatic classifier evasion*
- Known attack signatures → *Zero-day attacks*
- Detectable discrepancy → *Mimicry and reverse mimicry*

Motivations for PlatPal

Prior works rely on

- ~~External PDF parsers~~
- ~~Machine learning~~
- ~~Known attack signatures~~
- ~~Detectable discrepancy~~

What PlatPal aims to achieve

- Using Adobe's parser
- Using only simple heuristics
- Capable to detect zero-days
- Do not assume discrepancy
- Complementary to prior works

A Motivating Example

- A CVE-2013-2729 PoC against Adobe Reader 10.1.4

SHA-1: 74543610d9908698cb0b4bfcc73fc007bfeb6d84


Adobe Reader X
PDF
sample-cve-2013-2729-10.1.4


Macintosh HD
PDF
sample-
cve-201...0.1.4.pdf

Platform Diversity as A Heuristic

When the same document is opened across different platforms:

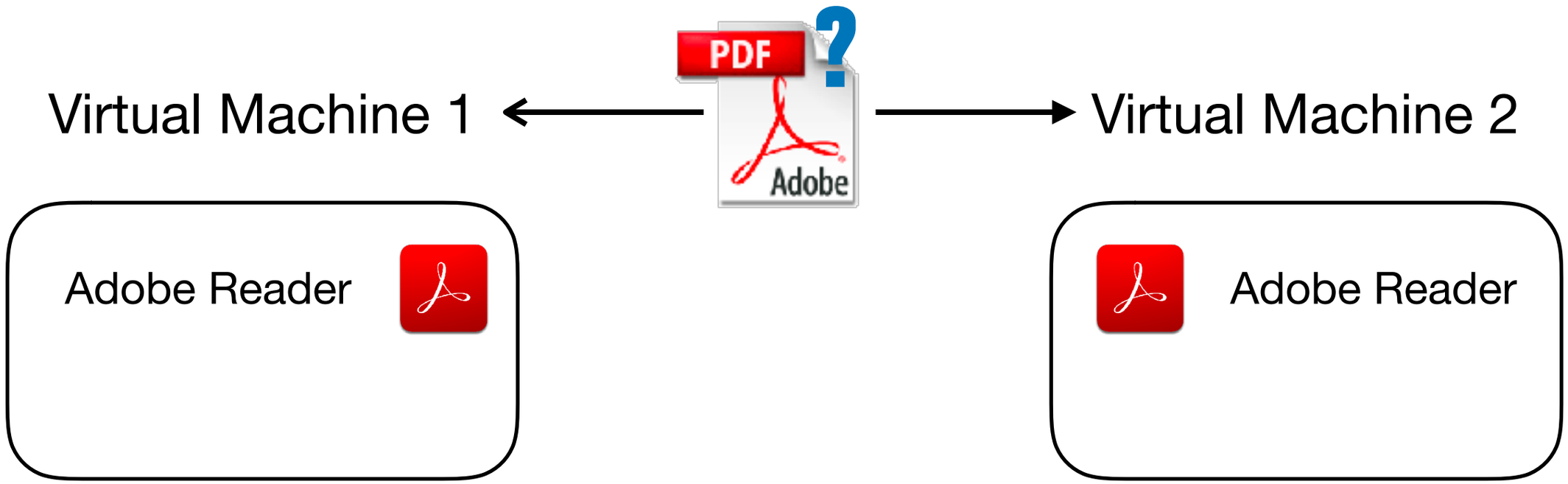
- A benign document “behaves” the same
- A malicious document “behaves” differently

Use machine learning to detect peculiarity.

Questions for PlatPal

- What is a “behavior” ?
- What is a divergence ?
- How to trace them ?
- How to compare them ?

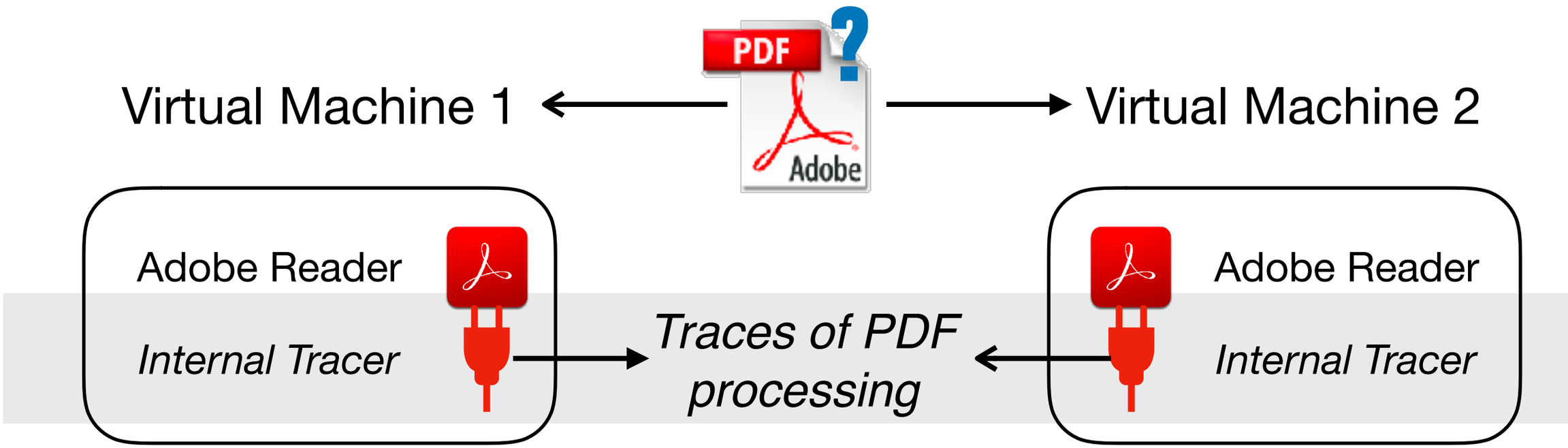
PlatPal Basic Setup



Windows Host

MacOS Host

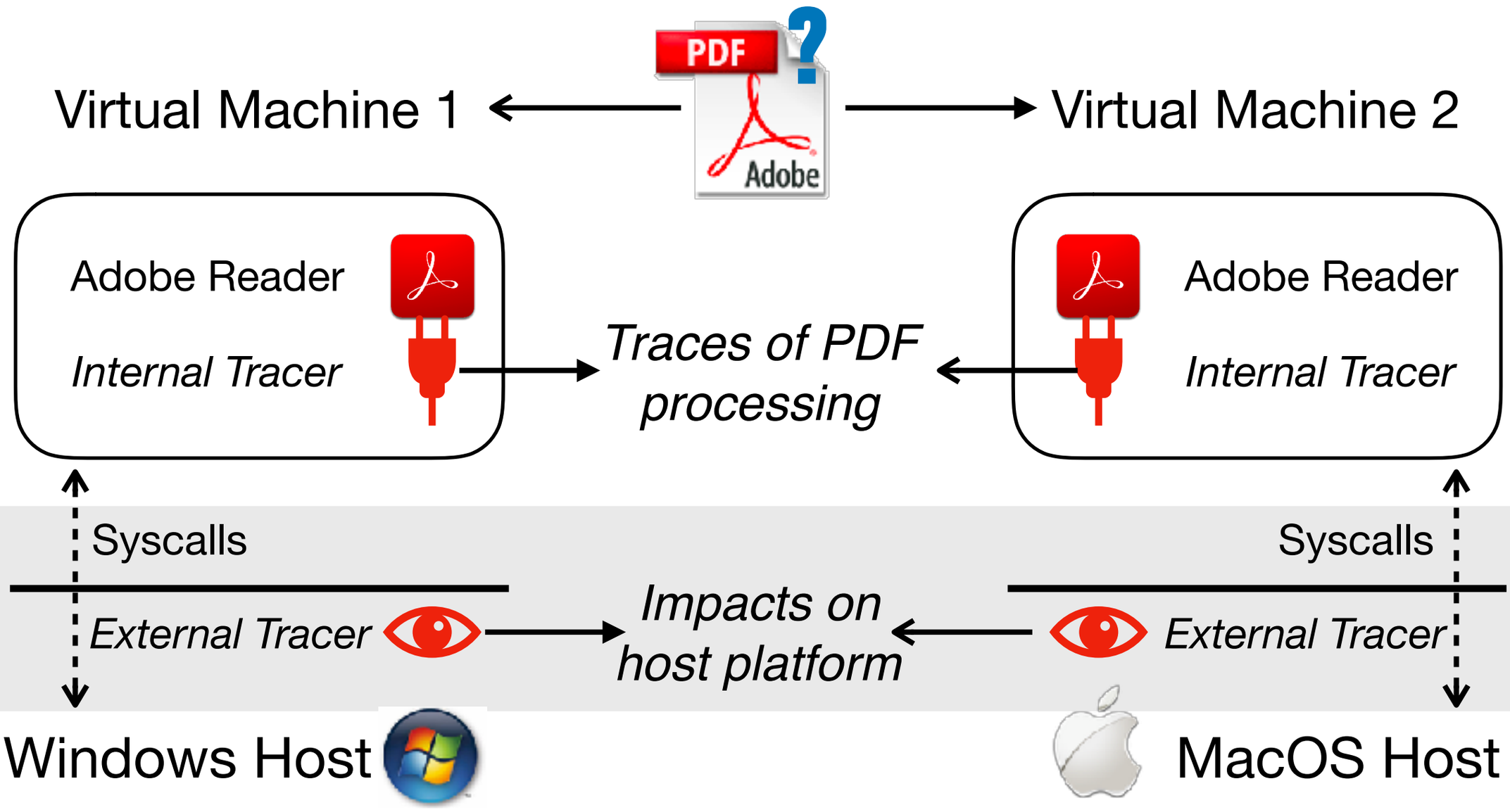
PlatPal Dual-Level Tracing



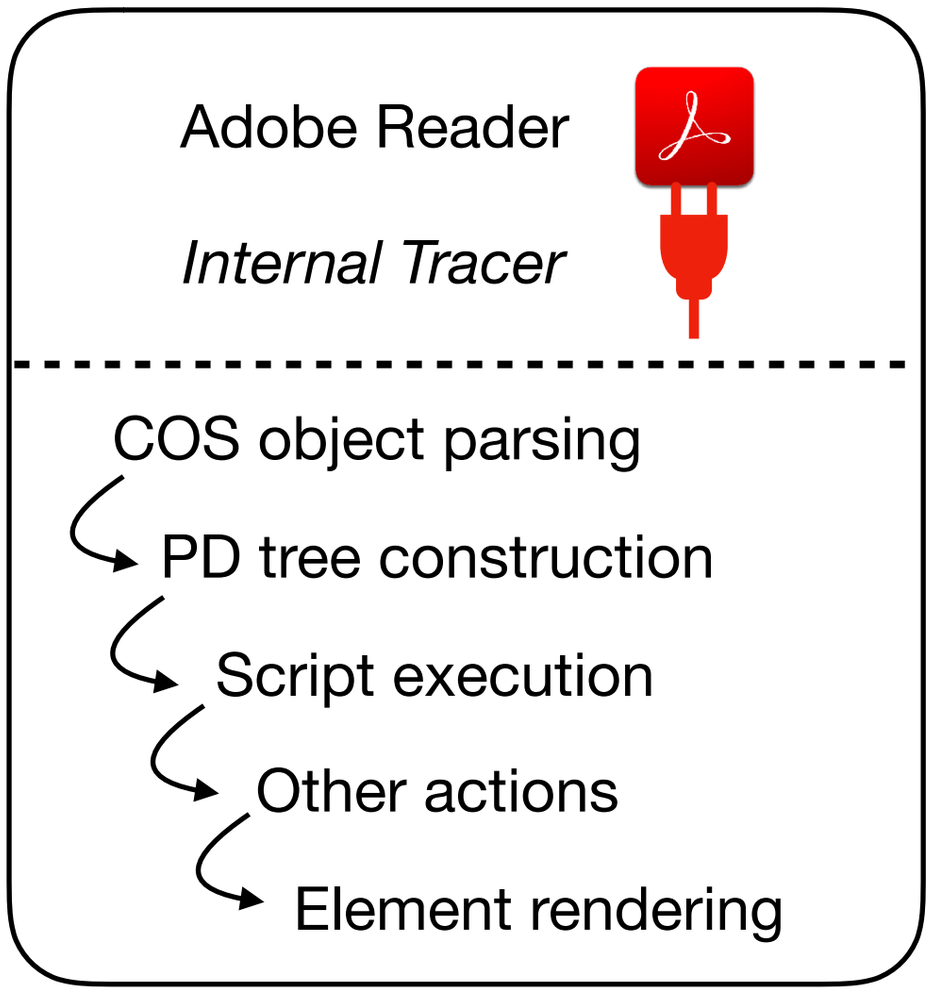
Windows Host

MacOS Host

PlatPal Dual-Level Tracing

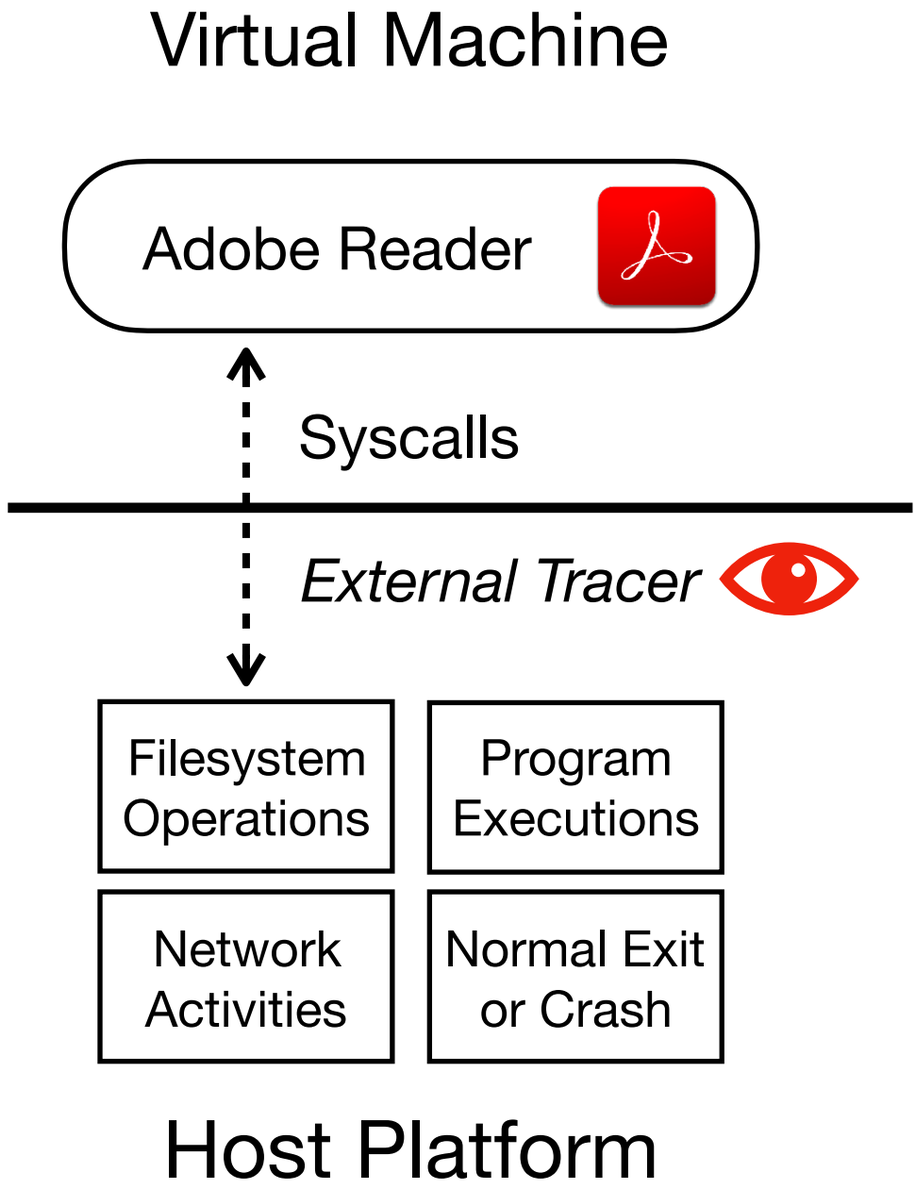


PlatPal Internal Tracer



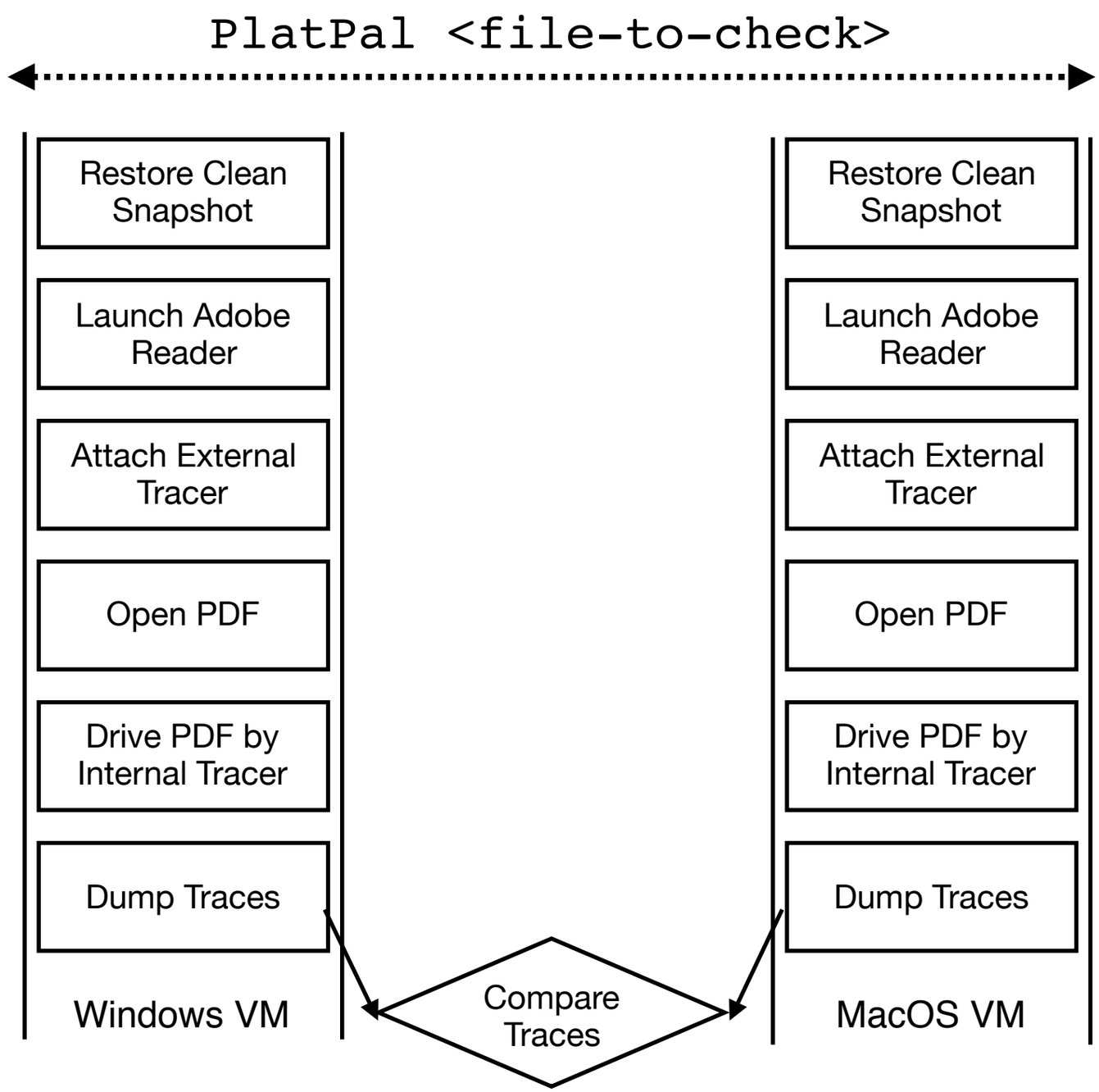
- Implemented as an Adobe Reader plugin.
- Hooks critical functions and callbacks during the PDF processing lifecycle.
- Very fast and stable across Adobe Reader versions.

PlatPal External Tracer



- Implemented based on *NtTrace* (for Windows) and *Dtrace* (for MacOS).
- Resembles high-level system impacts in the same manner as Cuckoo guest agent.
- Starts tracing only after the document is loaded into Adobe Reader.

PlatPal Automated Workflow



Evaluate PlatPal

- Robustness against benign samples
  - A benign document “behaves” the same ?
- Effectiveness against malicious samples
  - A malicious document “behaves” differently ?
- Speed and resource usages

Robustness

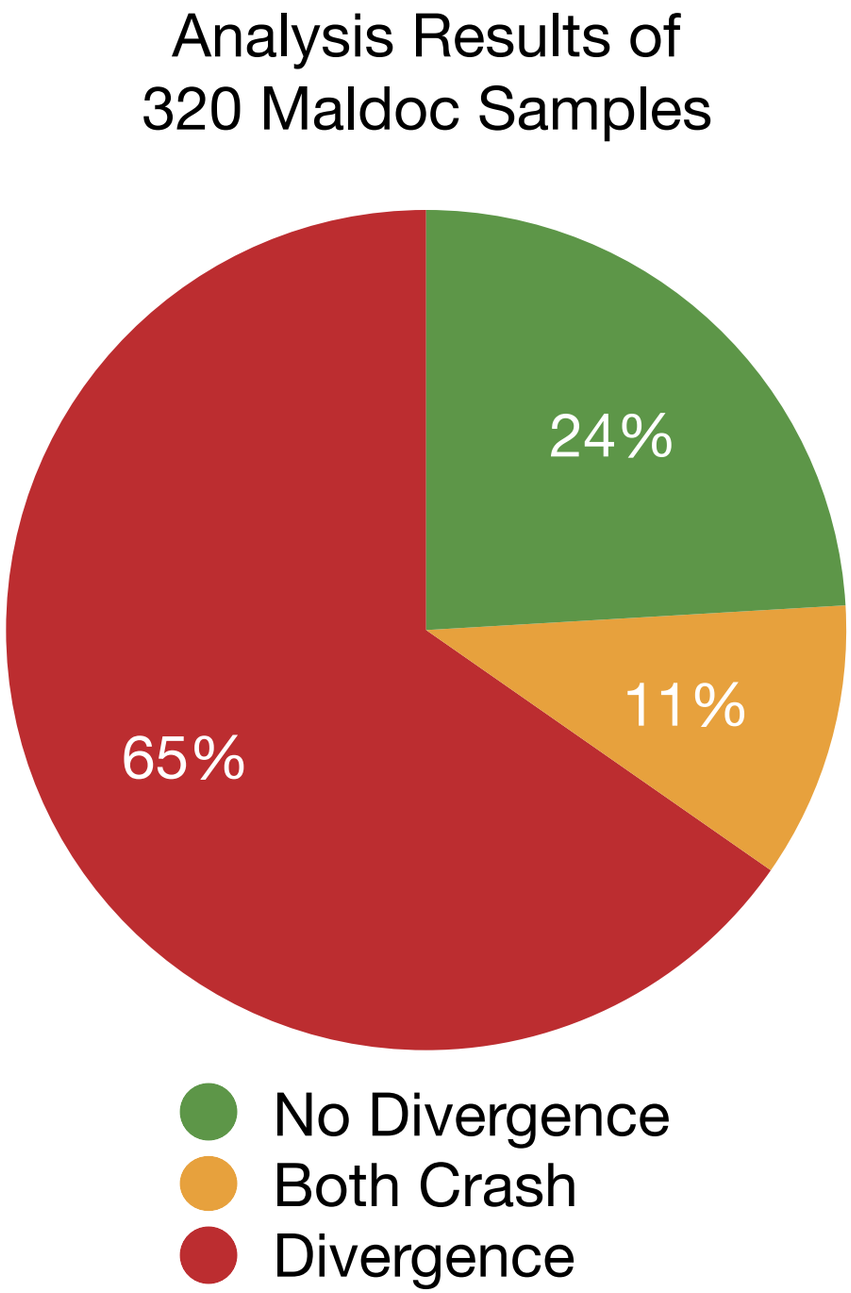
- 1000 samples from Google search.
- 30 samples that use advanced features in PDF standards from PDF learning sites.

Sample Type	Number of Samples	Divergence Detected ? (i.e., False Positive)
Plain PDF	966	No
Embedded fonts	34	No
JavaScript code	32	No
AcroForm	17	No
3D objects	2	No

Effectiveness

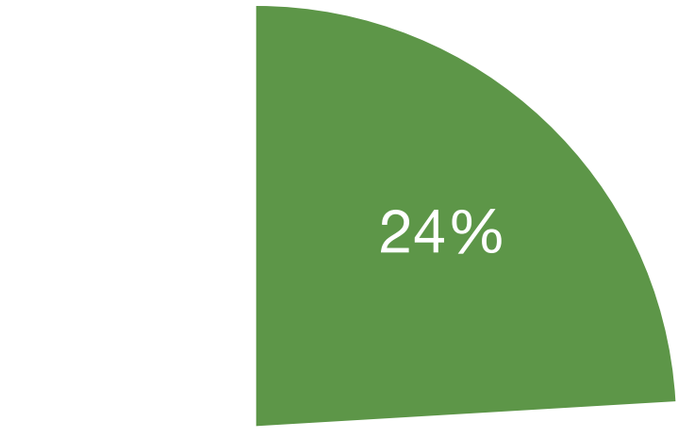
- 320 malicious samples from VirusTotal with CVE labels.
- Restricted to analyze CVEs published after 2013.
- Use the most recent version of Adobe Reader when the CVE is published.

Effectiveness

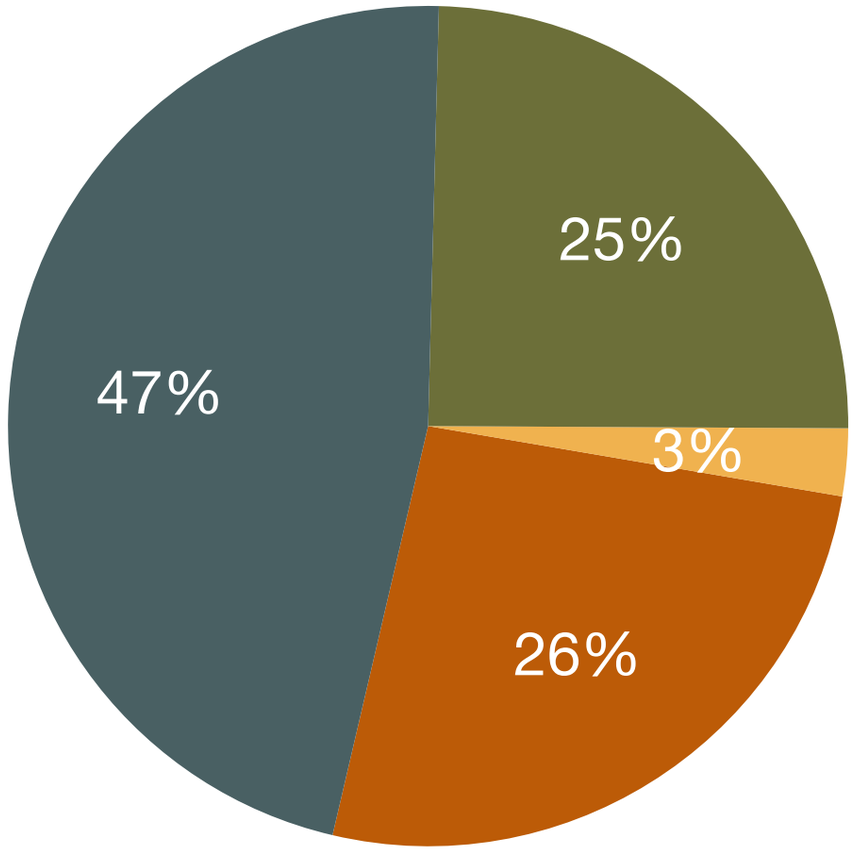


Effectiveness

Analysis Results of
320 Maldoc Samples



Breakdown of 77
potentially false positives



Time and Resource Usages

Average Analysis Time Breakdown
(unit. Seconds)

Item	Windows	MacOS
Snapshot restore	9.7	12.6
Document parsing	0.5	0.6
Script execution	10.5	5.1
Element rendering	7.3	6.2
Total	**23.7**	**22.1**

Resource Usages

- 2GB memory per running virtual machine.
- 60GB disk space for Windows and MacOS snapshots that each corresponds to one of the 6 Adobe Readers versions.

Evaluation Highlights

- Confirms our fundamental assumption in general:
  - benign document “behaves” the same
  - malicious document “behaves” differently
- PlatPal is subject to the pitfalls of dynamic analysis
  - i.e., prepare the environment to lure the malicious behaviors
- Incurs reasonable analysis time to make PlatPal practical

Further Analysis

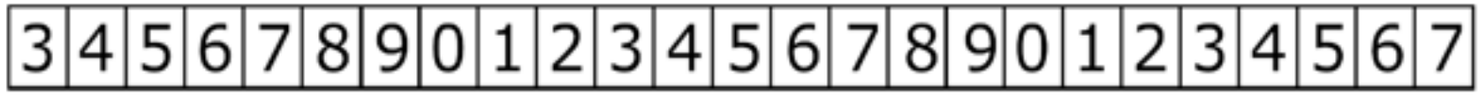
- What could be the root causes of these divergences?

Diversified Factors across Platforms

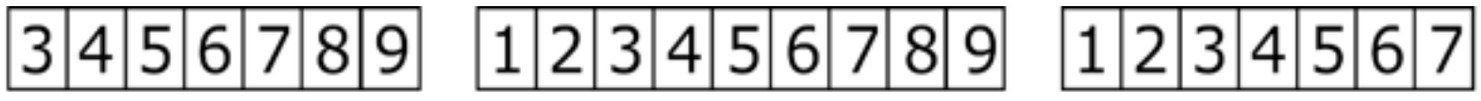
Category	Factor	Windows	MacOS
Shellcode Creation	Syscall semantics	Both the syscall number and the register set used to hold syscall arguments are different	
	Calling convention	*rcx, rdx, r8* for first 3 args	*rdi, rsi, rdx* for first 3 args
	Library dependencies	e.g., *LoadLibraryA*	e.g. *dlopen*
Memory Management	Memory layout	Offset from attack point (e.g., overflowed buffer) to target address (e.g., vtable entries) are different	
	Heap management	Segment heap	Magazine malloc
Platform Features	Executable format	COM, PE, NE	Mach-O
	Filesystem semantics	\ as separator, prefixed drive letter C:\	/ as separator, no prefixed drive letter
	Config and info hub	registry	proc
	Expected programs	MS Office, IE, etc	Safari, etc

Back to The Motivating Example

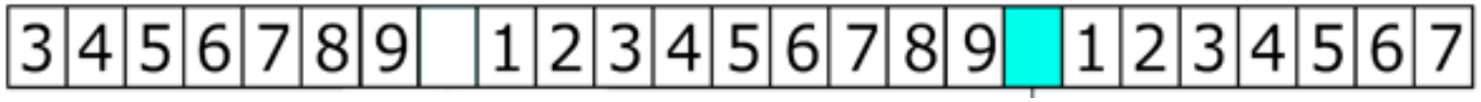
1. Allocate 1000 300-bytes chunks



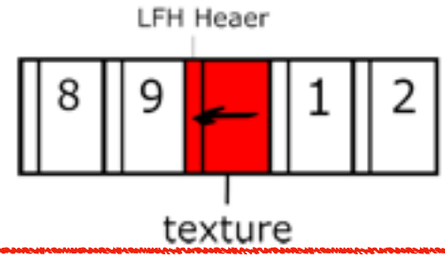
2. Free 1 in every 10



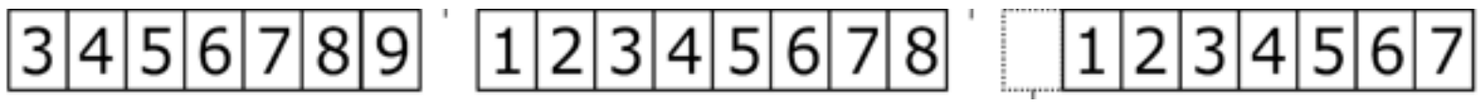
3. Load a 300-byte malicious BMP image



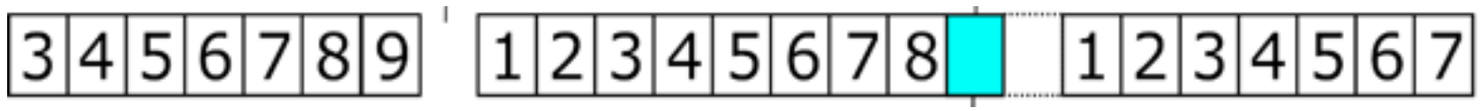
4. Corrupt heap metadata due to a buffer overflow



5. Free BMP image, but what is actually freed is slot 9



6. A *vtable* of 300-byte is allocated on slot 9, which is attacker controlled



Another Case Study

```
1  var t = {};
2  t.__defineSetter__('doc', app.beginPriv);
3  t.__defineSetter__('user', app.trustedFunction);
4  t.__defineSetter__('settings', function() { throw 1; });
5  t.__proto__ = app;
6  try {
7    DynamicAnnotStore.call(t, null, f);
8  } catch(e) {}
9
10 f();
11 function f() {
12   app.beginPriv();
13   var file = '/c/notes/passwords.txt';
14   var secret = util.stringFromStream(
15       util.readFileIntoStream(file, 0)
16       );
17   app.alert(secret);
18   app.endPriv();
19 }
```

CVE-2014-0521 PoC Example

Bypass PlatPal ?

*An attacker has to **<u>simultaneously</u>** compromise all platforms in order to bypass PlatPal.*

Limitations of PlatPal

- User-interaction driven attacks
- Social engineering attacks
  - e.g., fake password prompt
- Other none-determinism to cause divergences
  - e.g., JavaScript *gettime* or RNG functions

Potential Deployment of PlatPal

- Not suitable for on-device analysis.
- Best suited for cloud storage providers which can scan for maldocs among existing files or new uploads.
- Also fits the model of online malware scanning services like VirusTotal.
- As a complementary scheme, PlatPal can be integrated with prior works to provide better prediction accuracy.

Conclusion

- It is feasible to harvest platform diversity for malicious document detection.
- PlatPal raises no false alarms in benign samples and detects a variety of behavioral discrepancies in malicious samples.
- PlatPal is scalable with various ways to deploy and integrate.