Bypassing Linux EDRs

for fun and profit

Adhokshaj Mishra

Who am I?

.Security researcher (Linux) @ Uptycs Inc.

- Malware (offensive / defensive)

.Linux and FOSS enthusiast

.Loves presenting in meetups and conferences

.Hobbyist programmer (C++)

.Get in touch:

- Email: me <at> adhokshajmishraonline <dot> in

Agenda

- What is an EDR
  - What does it do?
  - How does it do it?
- General capabilities
  - How are they implemented?
- Deep dive into EDR techniques
  - Visibility
  - Detection

Agenda

- Deep dive into EDR techniques
  - Prevention
- Limitation and blind spots
  - How to exploit them?
- Defense evasion
  - From detection
  - From prevention
- Closing notes

What is an EDR?

.Integrated endpoint security solution

-Real-time continuous monitoring

-Continuous data collection, and analysis

-Anomaly detection

-Semi-automated response

.Blocking

.Containment

.Notification

What is an EDR?

- Generally centrally monitored and controlled
  - Agent process on every endpoint
  - Optional: kernel level drivers
  - Optional: binary blobs at middleware level
- Central data aggregation and analysis
  - Rule based analysis
  - Statstical analysis
  - AI / ML models

EDR Capabilities

- Real-time system tracing
  - Process execution
- And process relationships
  - File access
  - Privilege manipulation
  - Network activity

EDR Capabilities

- Memory analysis
  - Does this process have <something> in its memory?
- Trivial example: yara based detection
- Anomaly detection
  - Does this process behave as expected?
- Example: apache/nginx modifying PAM configuration.

EDR Capabilities

- Use case creation
  - Aids in hypothesis based proactive threat hunting
- Integration with services like VirusTotal
  - Helps in finding related binaries, records of prior activities

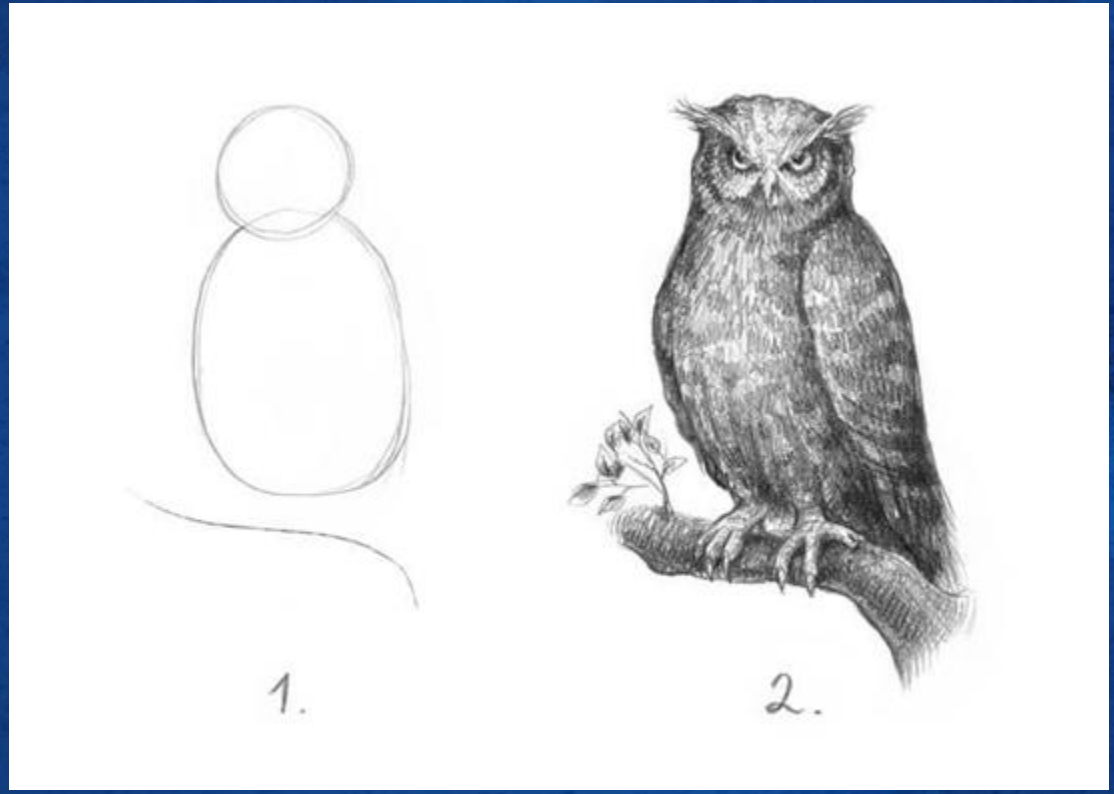
Thing you learn after building EDR software that runs on millions of endpoints across the world: “X should definitely not be doing Y” will almost always detect APT. It will also shut some company’s critical servers because they rely on that specific behaviour.

The detection pipeline

Red-teamers be like:

- Own everything
- Capture crown jewels
- Recommendation: Oh you just need to log X to detect Y.

The detection pipeline



The detection pipeleine

.The classic recommendation is no better than “draw rest of the owl” meme.

-Logging X at scale creates HUGE amount of data

.Most of it is noise

-Now you have to analyse even bigger data to find anomaly

.And chances of missing something malicious become higher

The detection pipeline

The detection pipeline

The detection pipeline

EDR Techniques

Common visibility sources

- Kernel level
- System-call tracepoints
- Linux security modules

EDR Techniques

Common visibility sources

- User level
- Linux Audit Subsystem
- Extended Berkeley Packet Filter (eBPF)
- Filesystem event notification system

EDR Techniques

Common visibility sources

- Middleware level
- Shared object injection
  - LD_PRELOAD
  - LD_LIBRARY_PATH
- Controlling the dynamic linker
  - LD_AUDIT

EDR Techniques

Common detection parameters

- Process execution
- Process tree
- Owner user/group
- Effective user/group
- Command line
- Hash of process binary

EDR Techniques

Common detection parameters

- Filesystem access
  - File path
  - Access type
  - Process trying to perform operation
  - Process tree
  - User and group information

EDR Techniques

Common detection parameters

- Network access
  - Local IP and port
  - Remote host and port
  - Operation type
  - Protocol
- Process trying to perform operation
- Process tree
- User and group information

EDR Techniques

Common detection parameters

- Privilege manipulation
- Current user information
- Current process information
- PAM parameters

EDR Techniques

Common prevention mechanisms

- Kernel level
  - Linux Security Modules
- Middleware level
  - Hooks using LD_AUDIT / LD_PRELOAD etc.
  - Seccomp profile via PAM
- User level
  - Filesystem event notification and permission subsystem

Limitations

Common visibility sources

.Kernel level

–System-call tracepoints

.Kernel driver compiled against one particular version (down to patch level), is not really going to work against another version.

.Need to build and test against every kernel version supported in every linux distro (and their versions) that we are going to support.

Limitations

Common visibility sources

.Kernel level

-Linux security modules

.Need to be compiled in the source tree of target kernel

.Cannot be loaded or unloaded dynamically

.Need to distribute compiled kernel for every distro (and their versions) that are going to be supported.

Limitations

Common visibility sources

.User level

-Linux Audit Subsystem

.Only one process can read data from audit subsystem socket at any time.

.If EDR agent does not forward event logs, these cannot be forwarded to some log aggregator/analysis tool.

Limitations

Common visibility sources

.User level

–Extended Berkeley Packet Filter (eBPF)

.Powerful enough only in recent kernels (4.18+)

.Tooling used to be dependent upon LLVM for “compilation on demand”

.eBPF helper libraries (and related infrastructure) are pretty recent

Limitations

Common visibility sources

.User level

-Filesystem event notification system

.Does not catch events occuring on network filesystems.

.File accesses and modifications via mmap(), msync(), and munmap() will be missed.

.Activity on children of a marked directory does not create events for the monitored directory itself.

Limitations

Common visibility sources

- Middleware level
- Shared object injection
- Does not work with statically linked binaries.
- Does not work with dynamically linked binaries using non-traditional dynamic linker.

Limitations

Common visibility sources

- Middleware level
- Controlling the dynamic linker
- Does not work with statically linked binaries.
- Does not work with dynamically linked binaries using non-traditional dynamic linker.

Other Blind Spots

- Command lines are brittle.
  - Command lines can be altered while preserving intended behaviour
- Process tree is brittle.
  - Ensuring process tree integrity is much harder than you think
  - Process trees can be altered.

Other Blind Spots

- File integrity monitoring is hard
  - Multiple paths can point to same file in a filesystem
  - Not all filesystems support event notification
  - A process can perform file I/O without touching filesystem stack.

Other Blind Spots

- System call events may appear with different process than monitored
  - Processes inherit a lot of things.
- Detecting shell environment variables is not a reliable way to detect almost anything.
  - Detecting LD_PRELOAD, anyone?

Defense Evasion

Let us start hacking already...

Defense Evasion

- Command line arguments
  - Way too much detection depends upon binary being executed, and its command line parameters.
  - Example: **crontab -** may be detected to trigger an alert.

Defense Evasion

.Sample detection

Event: Process updating cron job using crontab

pid: 120556

path: /usr/bin/crontab

cmdline: **crontab -**

current working directory: /home/adhokshajmishra

event time: Tue Jun 15 09:35:12 PM IST 2021

parent: /usr/bin/zsh

Defense Evasion

- Common techniques
  - Change executable name
- Copy with different name
- Symlink with different name

Defense Evasion

- Common techniques
  - Using utilities which can act like other ulities
- Busybox
- Toybox
- Gow

Defense Evasion

- Common techniques
  - Using utilities which can run other binaries
- ld.so
- systemd-run

Defense Evasion

- Process tree
  - Another significant chunk of detection depends upon process tree spawning the “offending” process.
  - Example: **bash** getting spawned from some terminal emulator is okay, but getting spawned from nginx is not.

Defense Evasion

.Sample detection

Event: Shell being spawned by nginx

pid: 120570

path: /usr/bin/bash

cmdline: /usr/bin/bash -c “wget https://...

current working directory: /var/www

event time: Tue Jun 15 09:36:12 PM IST 2021

parent: **/usr/bin/nginx**

Defense Evasion

.Common techniques

- Let some other process do the work

.Insert cron entry for some suitable user

.Add a system service, and let systemd do the job

Defense Evasion

- Common techniques
  - Or, code injection
- Add entry in /etc/ld.so.preload
- Add PAM module to inject shared object in user session
- Use ptrace to control another process, and spawn from there

Defense Evasion

- Common techniques
  - Poisoning dynamic linker cache
    - Modify /etc/ld.so.cache in-place, to overwride a common shared object (and load your bogus one)
  - Patching dynamic linker
    - Modify the ld.so in-place to read ld.so.preload from alternate (non-standard path)

Questions?