A Survey on Cross-Architectural IoT Malware Hunting

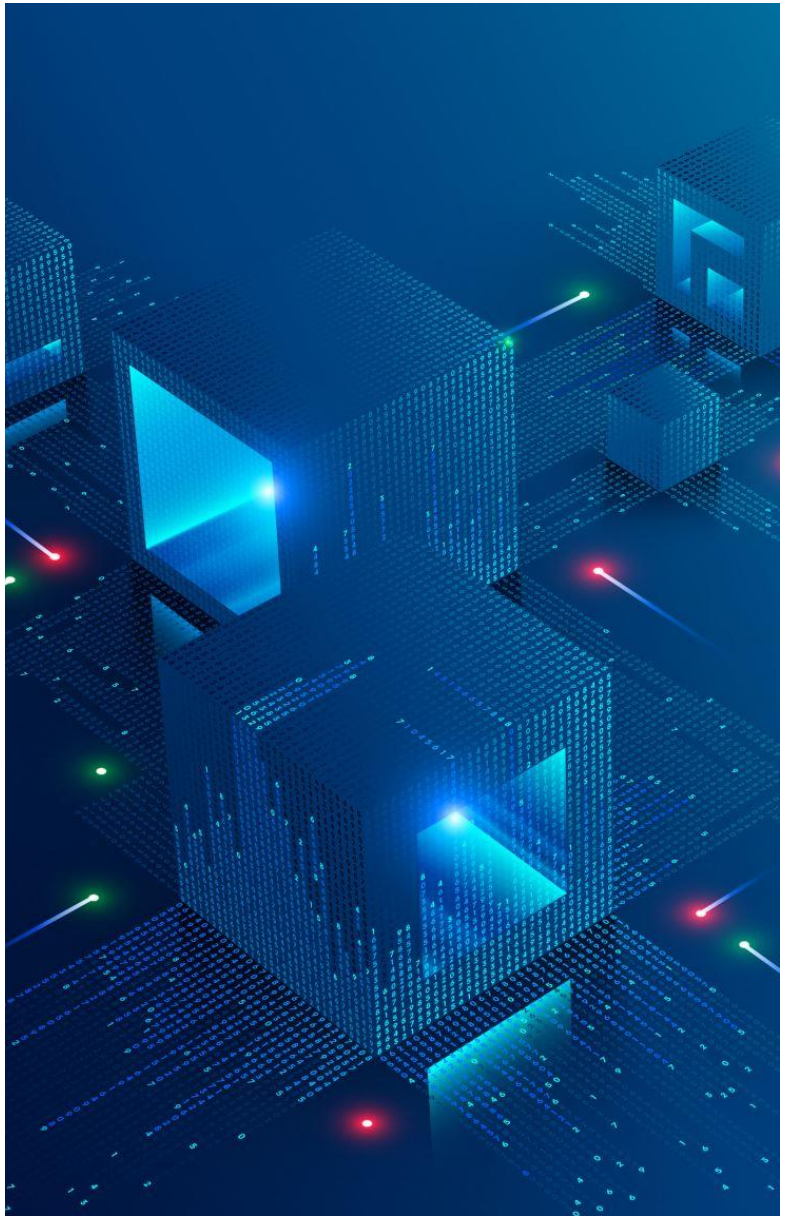
~ Part 01 ~

Written By: Raju et. al,.
(School of Computing Science , Simon Fraser University, Canada)

Presented By: 제레미아

Course: Advanced Security in Emerging ICT

Monday, October 31, 2022 (Week - 09)



PART I – INTRO, BACKGROUND & TAXONOMY

1. Introduction ~ Background
2. IoT Malware Hunting Background
   - ✓ A: ELF File Formats (Linux)
   - ✓ B: IoT CPU Architectures
   - ✓ C: IoT OS Platforms
   - ✓ D: Feature Extraction Tools for ELF Static Analysis
   - ✓ E: Malware Threat Hunting Approaches
3. Taxonomy
   - ✓ A: Metric Based
     1. High level Features: ELF Header, Strings, Symbol Table, System Call and APIs.
     2. Assembly Level Features: Opcodes & Mnemonics
     3. Machine Level Features: Static emulation,
   - ✓ B : Graph / Tree-Based Features
     1. Graph Based Features
     2. Tree Based Features
   - ✓ C: Sequence- Based Features
     1. 1-D Sequence: (Byte sequence, assembly instruction sequence, entropy sequence, short sequence,)
     2. 2-D Sequence: (Grayscale image, color image)
     3. 3-D Sequence (Latent projection,)
   - ✓ D: Interdependence

     (File to machine relation, file to file relation)
   - ✓ E: Dynamic Analysis

1.

INTRODUCTION

1. Introduction / Background:

- In recent years, the increase in non-Windows malware threats had turned the focus of the cybersecurity community.
- Research works on hunting Windows PE-based malwares are maturing, whereas the developments on Linux malware threat hunting are relatively scarce
- With the advent of the Internet of Things (IoT) era, smart devices that are getting integrated into human life have become a hackers' highway for their malicious activities

  - ✓ This study provides a comprehensive survey on the latest developments in cross-architectural IoT malware detection and classification approaches.
  - ✓ The study discuss the feature representations, feature extraction techniques, and machine learning models employed in the surveyed works

1.

INTRODUCTION

1. Introduction / Background:

- In the past two decades, the machine learning approaches adapted to the domain of malware detection/classification strove towards convergence at better handling of malware threats as hard as zero-day attacks
- Malware attacks are steadily on the rise where some financially motivated attacks target big industry players but some attacks (60%) are directed towards small and mid-sized businesses.
- Such attacks are estimated to cause a worldwide damage of approximately 6 Trillion in 2021 and expected to rise to 10.5 Trillion by 2025. Overall ransomware attacks grew by 150% in 2020.
- Reasons accounting for this rise includes:
  - ✓ Availability of Malware groups such as Egrogor and Netwalker which provides Ransomware-as-a-service (RaaS) accounting for 64% of the total ransomware attacks.
  - ✓ IoT devices are deemed to be the most targeted at present. Even wearables as FitBit devices are also vulnerable to getting hacked which puts in danger the PII (Personal Identifiable Information)

1.

INTRODUCTION

1. Introduction / Background:

- Surprisingly, smart connected devices such as security cameras, refrigerators, and toasters were part of the BotNets (roBot Networks) in the infamous massive DDoS (Distributed Denial of Service) cyber-attack against Dyn DNS provider by Mirai which caused parts of the world inaccessible to major sites like Airbnb, Twitter, PayPal, GitHub, Amazon, Netflix
- DDoS attacks via BotNets are now the extensively used distributed attack source targeting T devices, and their strains spread over 25 different malware families.
- DDoS-for-hire had become one of the trending hack-for-hire services, where botnets with GBps to TBps attack bandwidth are being sold in the underground forums of the dark web. In light of the above mentioned issues:
  - ✓ The problem of malware detection and/or classification continues to be a topic of much importance.
  - ✓ This paper address this problem of detecting and/or classifying the malware threats commonly with the term 'Malware Threat Hunting'.

2. IoT Malware Background

- **Internet-of-Things (IoT)** is a large set of devices connected via the private or public internet, and that is infused with the ability to talk to each other streaming real-time data with less or no intervention required from humans, thereby building a unified intelligence.
  - ✓ **What is considered an IoT device??** Nowadays, devices of any size with a chip installed for enabling centralized control, device-to-device control, wireless sensor networks, and embedded systems are considered IoT devices.
  - ✓ For example, security motion sensors, smartphones, voice assistant-controlled home automation devices like TVs, speakers, home lighting systems are considered IoT devices.
- **IoT Devices Features:**
  - ✓ IoT devices are generally equipped with less computing and storage compared to traditional laptops and PCs which impose tight constraints leading to the need of specialized OS and CPU architecture
  - ✓ Windows, Linux, Android, iOS dominates Laptops, PC, Servers and mobile devices but they are not suitable for embedded device in constrained IoT space.

2.

IoT MALWARE

BACKGROUND

2. IoT Malware Background

- An OS for IoT device should be lightweight to support the minimal hardware yet following security requirements. Linux flavors and distributions such as Ubuntu core, Raspian (Debian) supports such requirements and hence are widely used by the IoT developers.
- ELF-based malware gained attention from the cybersecurity community only in the mid of the past decade when a large number of samples started accumulating with VirusTotal before which it was generally believed that Linux was not as vulnerable as Windows
- OSes such as Windows and Android would follow a one-of-a-kind approach and usually make use of features tailored for that specific OS and would not be transferable to other OSes
  - ✓ Techniques proposed for Linux ELF threat hunting suffer from not being able to follow the many-of-the-same-kind approach to accommodate the multiple distributions and variants within the Linux landscape

2.

IoT MALWARE

BACKGROUND

RANKING OF OPERATING SYSTEMS FOR IOT

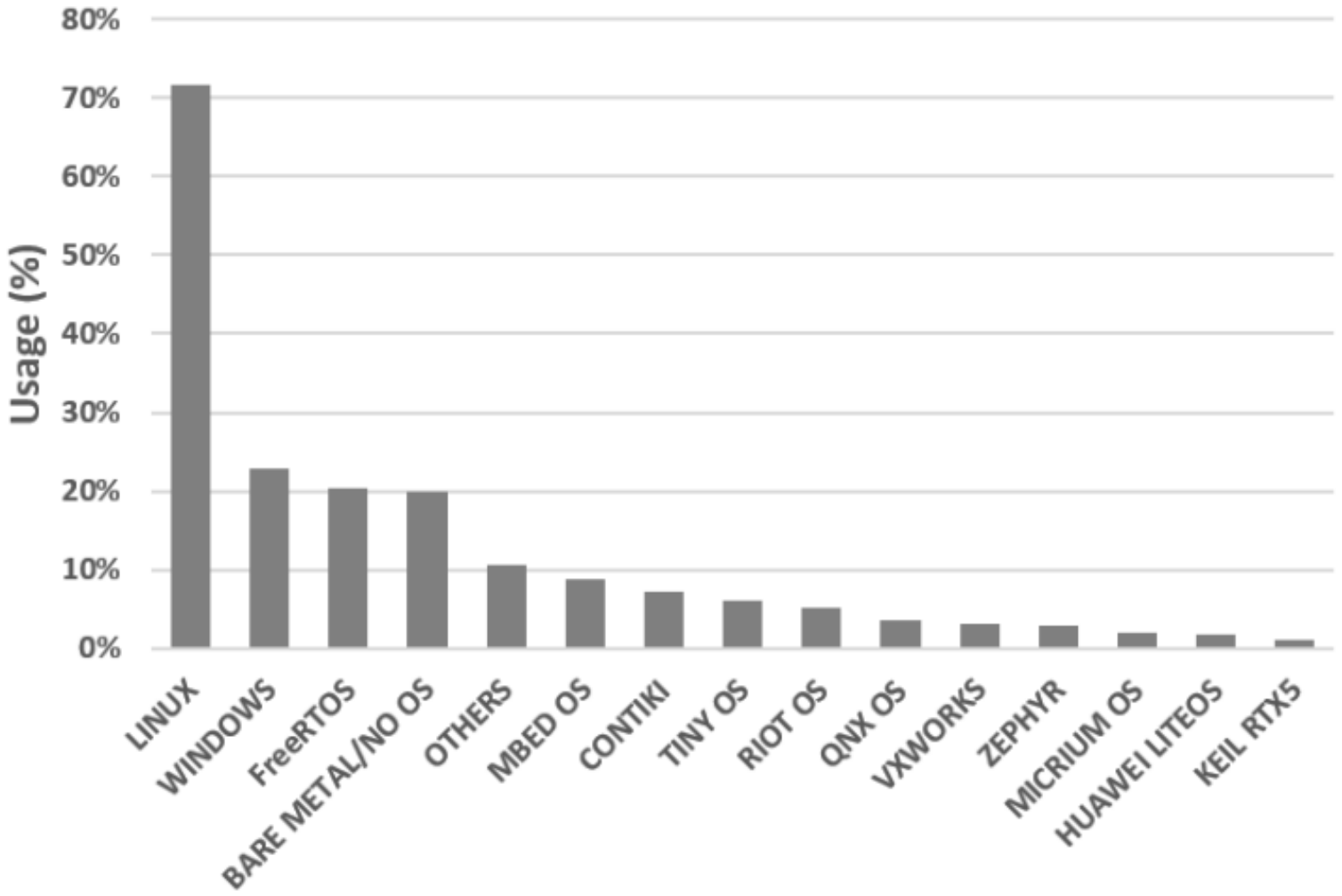


FIGURE 1. Ranking of operating systems for IoT [24].

Overview:

- ✓ **Overall:** From Figure 1, it can be seen that, Linux variants are the most utilized operating systems for IoT devices, according to a survey by Eclipse Foundation (2018)
- ✓ As of 2020, Linux and FreeRTOS were the top OSes IoT developers preferred with 43% and 35% ratings, respectively.

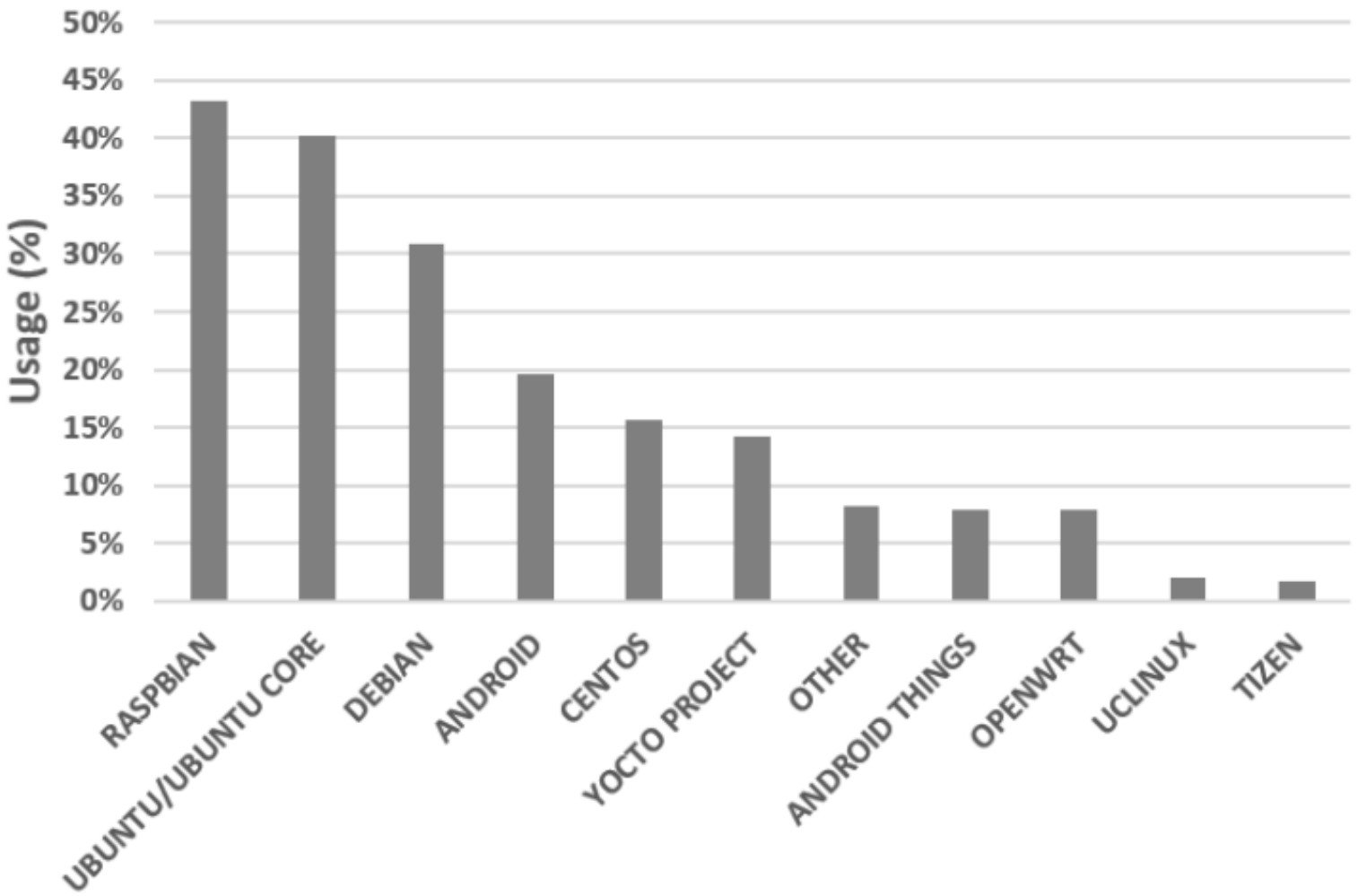


FIGURE 2. Ranking among linux distros for IoT [24] .

RANKING AMONG LINUX DISTROS FOR IoT

Overview:

- ✓ Figure 2 illustrates the ranking among distros within Linux.
- ✓ Lastly, Despite the differences in many flavors of Linux, much of the existing research work on IoT malware threat hunting declare the problem of handling the different CPU architectures such as MIPS, AARCH, and ARM, as the prominent challenge being encountered.

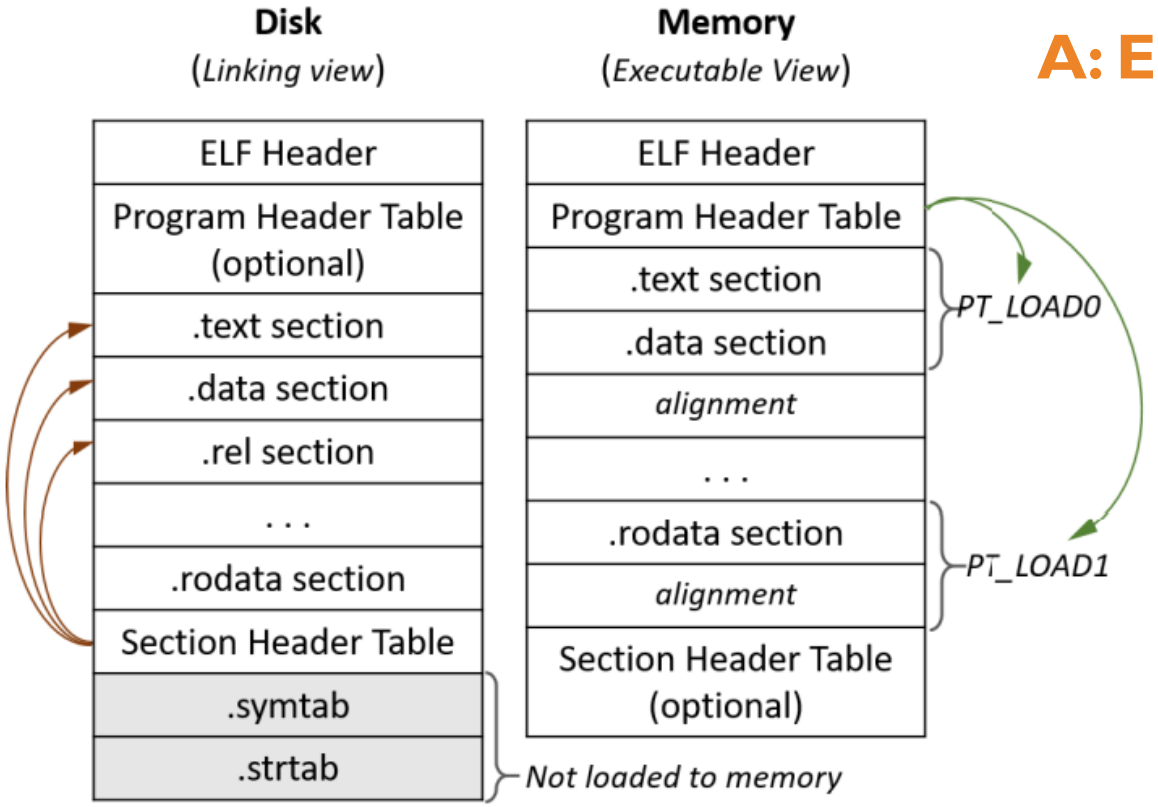


FIGURE 3. ELF file format - source [44].

A: ELF FILE FORMAT

- Executable and Linkable Format (ELF) is the standard binary file format for the file types Linux executables, used by operating systems like Linux, BSD, Solaris, BeOS, and Android.
  - ✓ ELF have cross-platform properties: this property allows ELF to be used across different CPU architectures: Intel (x86, x64), ARM, MIPS, Motorola, SPARC, PowerPC, Renesas SH, Motorola m68k, and different target devices: Routers, Printers, Cameras, etc.
- Figure 3 illustrates the general ELF file format. ELF file is composed of three major categories:
  - ✓ Program Header that aids in handling memory segments during run time execution by providing information to the system on how to create process images,
  - ✓ The individual 'sections' that hold various types of information such as 'code' and 'text,' and finally,
  - ✓ The Section Header that describes the various file sections such as their offset information and also helps in linking and relocation process.

A: ELF FILE FORMAT (Continued…)

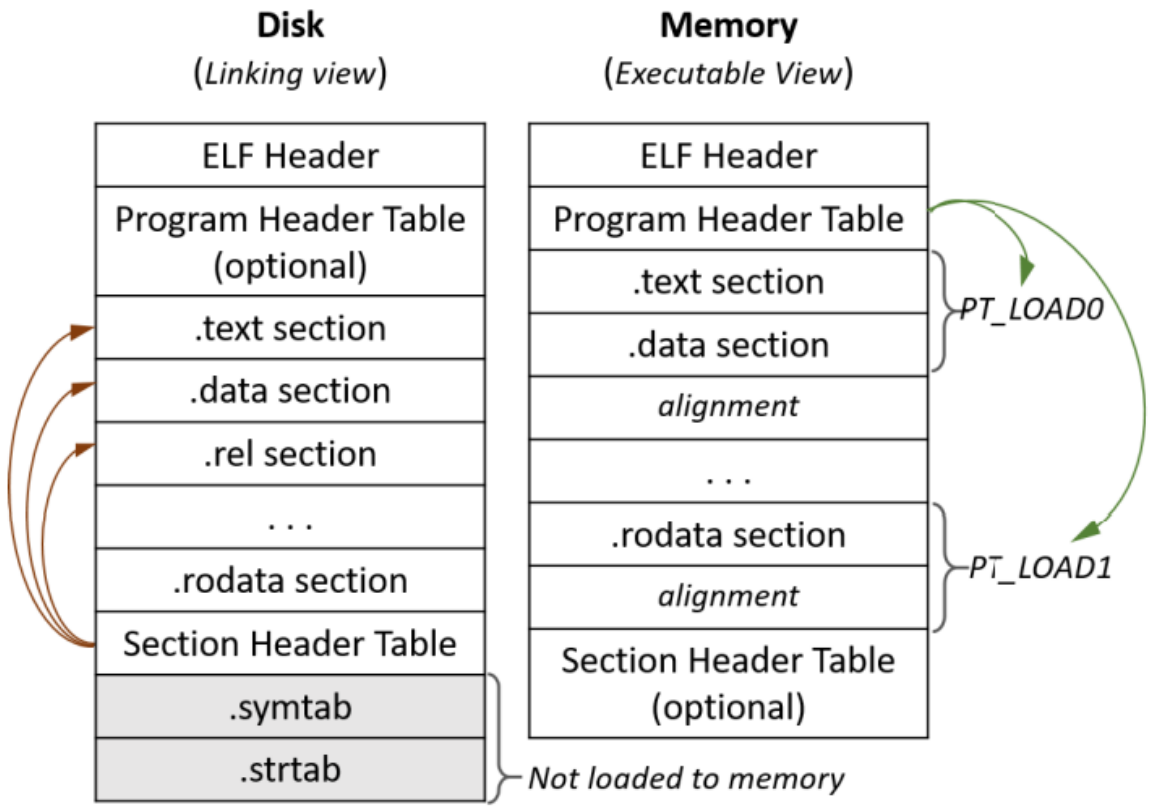


FIGURE 3. ELF file format - source [44].

- There are two types of views:
  1. Linking view, where the sections and the section header table are important but the program header table is optional;
  2. Execution view, where the segments and the program header table are important, but the section header table is optional.
- Table I (next slide) provides a list of some segment types typically found in an ELF binary.
- Windows PE file format and Linux ELF file format are similar in structure in that both use a Header that defines meta-information about the rest of the file structure, and in that, both formats use a Section Header to define the individual sections.

TYPES OF SEGMENT ENTRIES IN ELF

TABLE 1. Types of segment entries in ELF [25].

Segment Type	Description
PT_NULL	PT_NULL allows program header table to contain entries that can be ignored during execution
PT_LOAD	PT_LOAD is used to define a loadable segment using the values specified by p_memsz & p_filesz. Such loadable segment entries are sorted by by their p_vaddr member and are listed in the ascending order in the program header table
PT_DYNAMIC	PT_DYNAMIC is used to describe dynamic linking information
PT_INTERP	The segment type PT_INTERP meaningful only for executable files and possibly shared objects occurs once, preceding any of the loadable segment entry. It is used to describe the size and location of an interpreter to be invoked via a null terminated path
PT_NOTE	PT_NOTE segment is used to describe the size and location of auxiliary information
PT_SHLIB	Having unspecified semantics, the PT_SHLIB is a reserved segment type entry whose presence indicate that the binary may not conform to the ABI (Application Binary Interface)
PT_PHDR	The segment type PT_PHDR occurs once in a binary file preceding any of the loadable segment entry when program header table is part of the program's memory image, and is used to describe size and location of the program Header table in both the disk file as well as program's memory image
PT_TLS	PT_TLS is not a mandatory entry program header table. It is used for the specification of the template for thread-local storage.
PT_LOOS to PT_HIOS	Operating system specific semantics are described via these reserved entries
PT_LOPROC to PT_HIPROC	Processor-specific semantics are decribed via these reserved entries

B: IoT CPU ARCHITECTURES:

- The rapid proliferation of IoT devices that can perform an assortment of functionalities calls for complex product design across the IoT landscape to achieve high performance with low power demands.

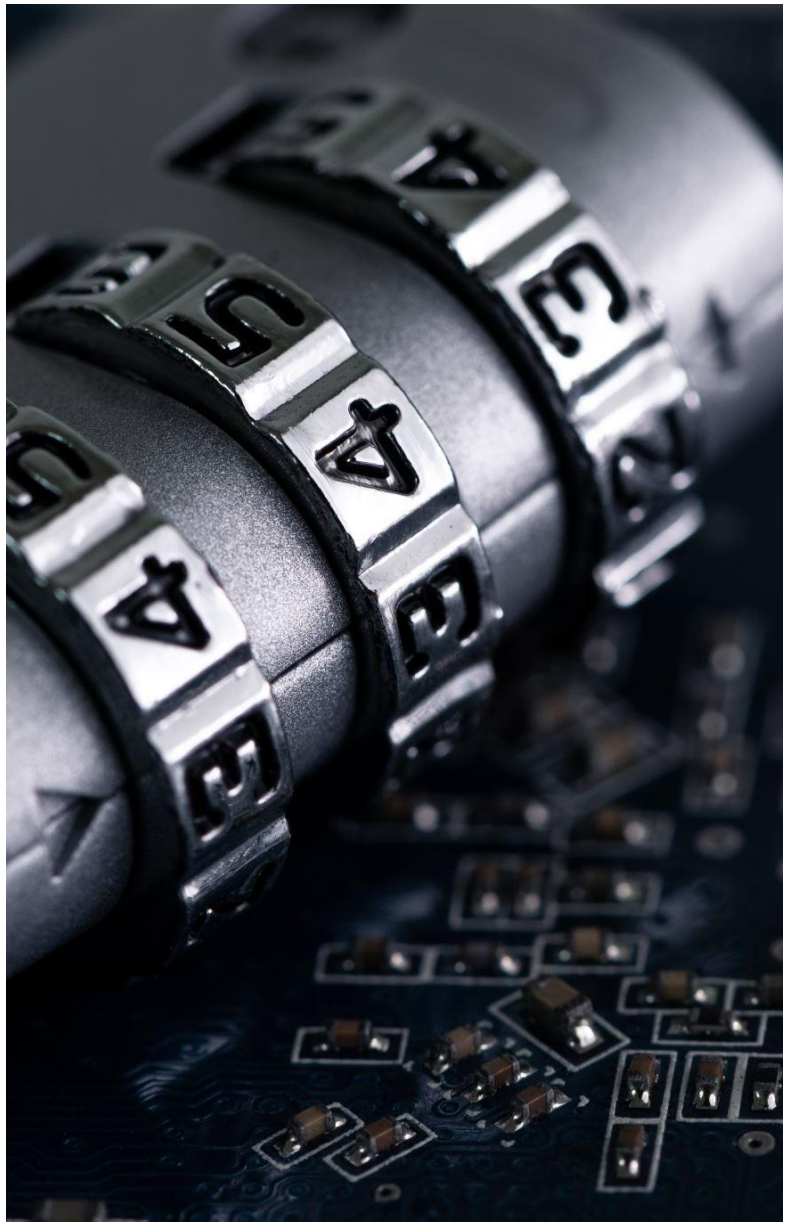
- Each CPU architecture in the IoT market is built for a specific purpose under various constraints that arise due to the trade-offs between power and performance.

- The complexity is compounded with recent developments in IoT to support Artificial Intelligence and Machine learning tasks which require far greater performance, power, and latency requirements.

  - ✓ Examples CPU Architectures: x86, ARM, MIPS, SPARC, AARCH64, PowerPC, Renesas SH, Motorola 68020.

C: IoT OS PLATFORMS:

- Similar to standard operating systems like Windows, iOS, and Linux, the IoT operating systems are expected to manage the embedded device functions but operate under the limited memory footprint, power, and processing capabilities.
- Some open-source operating systems for IoT include:
  - ✓ Raspbian, Contiki, FreeRTOS, Ubuntu Core, ARM mbed, Yocto, Apache Mynewt, and Zephyr OS and some of the commercial IoT OSes include Windows 10 IoT, Android Things, WindRiver VxWorks, Freescale MQX, Mentor Graphics Nucleus RTOS, Express Logic ThreadX, TI RTOS and Particle.
- In light of these diverse OSes, it is crucial to choose feature representations with capabilities for OS platform independence.

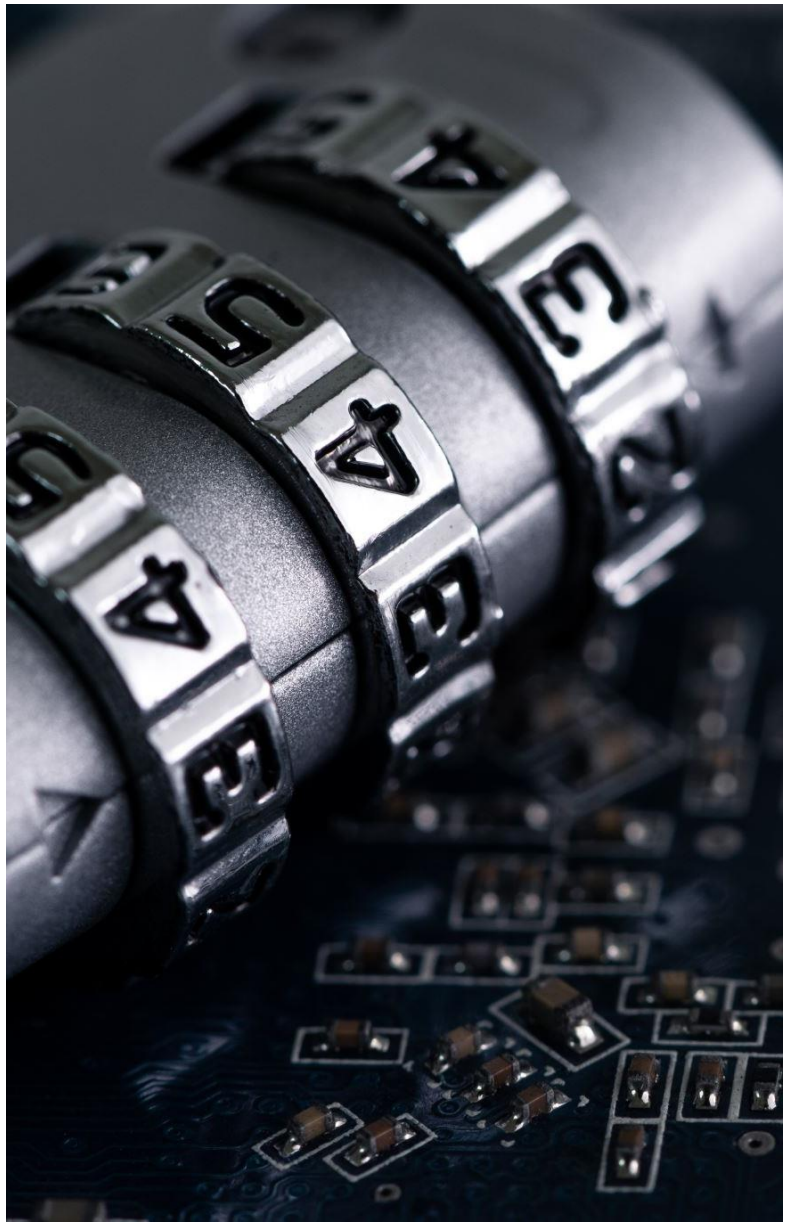


D: FEATURE EXTRACTION TOOLS FOR ELF STATIC ANALYSIS:

- A Linux-based operating system interprets the desired machine instructions using the formal ELF file format specification, which is the binary output format of a compiler or linker [54].
- In Table 2 (next slide), authors provided a short overview of the tools, including the tools used in surveyed works, that are helpful to analyze, debug and extract useful information from ELF files.
- Many of the surveyed works used the scanning services such as VirusTotal, Shodan, and Zmap to label their ground truth and validate the datasets employed in their studies.

TABLE 2. Tools for ELF static analysis.

Tool	Used By	Purpose of Use / Description
bindiff	[21]	Binary file analysis tool for disassembled code similarity and function similarity
binwalk	[30]	Extracting firmware images and reverse engineering
diaphora	[23]	Advanced binary program diffing tool with support to assembler and CFG diffing, call graph matching calculation, etc
elfdump	-	Available under Solaris and FreeBSD. Useful to find detailed information about dynamic linkages, relocations, non-stripped binary's symbol details, dependencies on shared objects, functions, sections and program segments [31]
elfutils	[32]	Faster, more featureful alternative tools to GNU Binutils purely for Linux [33]
file	[8], [30]	Useful to determine type of a file - not to be used as a security tool as it can be easily fooled by abusing a file's magic
ghidra	[34]	Reverse engineering tool like IDA. Extensible, supports analysis of very large firmware images, ability to decompile object code back to source code
hexdump	-	Utility for inspecting files via hex, decimal, octal and ASCII views. Allows data recovery and reverse engineering
hexedit	-	Helps to view/edit files in hex or ASCII
IDAPro	[28], [21], [34], [35], [36], [30]	Prominently used interactive disassembler and debugger tool
magic	-	file command's magic pattern file
nucleus	[28]	A structural control flow graph analysis based compiler agnostic function detection tool for binaries proposed by Andriesse *et al.* [37].
obj(ect)dump	[38], [39], [40]	Information dump about object files including intended target instruction set architecture (ISA) and structural information. Relies on BFD.
od (octal dump)	-	Tool for debugging, visualizing executable code, and dumping in octal (default), hex, ASCII formats.
openwrt	[21]	For benign firmwares
pyelftools	[28]	Python library to parse and analyze ELFs and debugging
radare2	[32], [34], [41]	Binary forensic analysis, reverse engineering, exploiting and debugging tool. Options such as 'afl' can be used to disassemble function lists, get count of functions etc.
readelf	[28], [42], [8], [43]	Prominently used to obtain ELF structural information. Provides more details than objdump. It is independent of Binary File Descriptor (BFD) library.
size	[8]	Provides total ELF size as well as section sizes
strings	[8], [1]	ASCII strings information from binary



E: MALWARE THREAT HUNTING APPROACHES:

- The malware analysis phases involved in the malware threat hunting process can be generally classified into static, dynamic, and hybrid analysis categories

- **Static malware** analysis occurs when a binary file is reverse engineered, disassembled, or dissected using different tools, then analyzed using various structural and semantical information found in the binary file without execution. This method is susceptible to evasive methods like anti-disassembly, code obfuscation techniques.

- **Dynamic analysis** is a behavioral method that observes or debugs a malware's behavior in an isolated environment such as sandboxes. Dynamic methods are also susceptible to evasive techniques such as anti-debugging and differed execution.

- **Hybrid analysis** combines both static and analysis methods.

```
ELF 64-bit LSB shared object,
x86-64, version 1 (SYSV),
dynamically linked,
BuildID[sha1]=3b41c6707ba33ecc7afe...,
stripped
```

FIGURE 4. Basic information provided by 'file' command.

Overview:

- Ngo et al. [22] claimed that the static analysis method has more ability than dynamic methods in analyzing malware structure without the need to consider processor architecture. Figure 4 provides a sample basic information that can be obtained using Linux 'file' command.
- As shown in Figure 4, symbol, debugging, and relocation information could be stripped from an ELF binary to make them lightweight.
- However, studies have shown that IoT malwares are mostly statically linked [35] and not stripped to reduce the dependency on the diverse IoT execution environments and avoid runtime failures. It also makes them hard to analyze under static analysis.

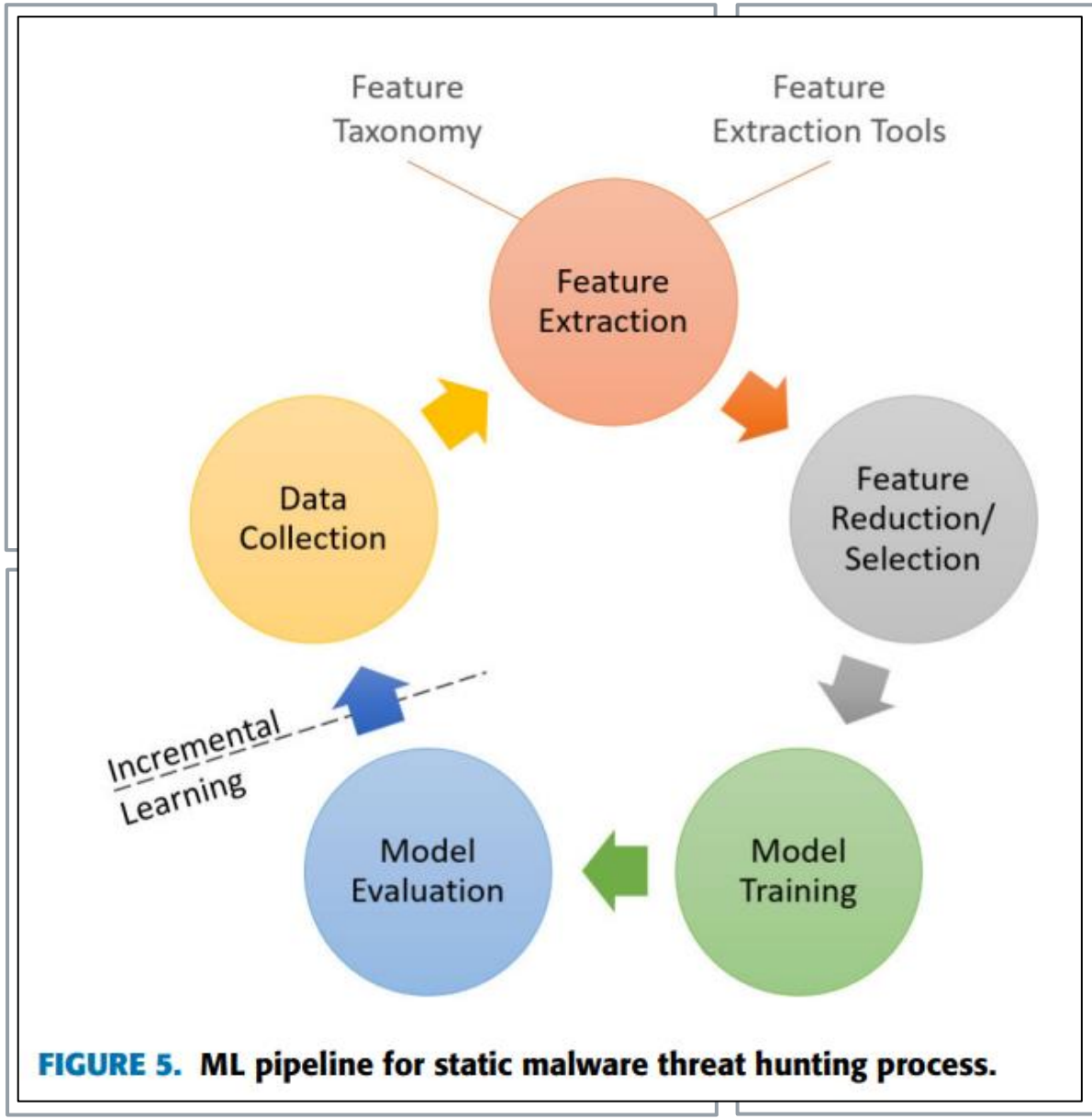


FIGURE 5. **ML pipeline for static malware threat hunting process.**

Overview:

- ✓ Figure 5 illustrates the generic machine learning-based pipeline for static malware threat hunting.
- ✓ It also showcases where the feature extraction tools described in Section II-D and the modern taxonomy described in Section III fit the pipeline.

3. TAXONOMY

- This section provides a taxonomy of feature representations used for static analysis-based malware threat hunting in the IoT landscape. Highlighted features are specifically useful for cross-architectural IoT malware threat hunting and they're OS platform independent.

  - ✓ Figure 6 (next slide) provides the categorization of the feature representations based on four major divisions, namely: metric-based, graph/tree-based, sequence-based, and interdependence.
  - ✓ These divisions encompass representations extracted from the content within a sample, such as strings and opcodes, as well as the external characteristics of a sample such as file-to-machine relations

3.

TAXONOMY

Taxonomy of ELF Feature Representations

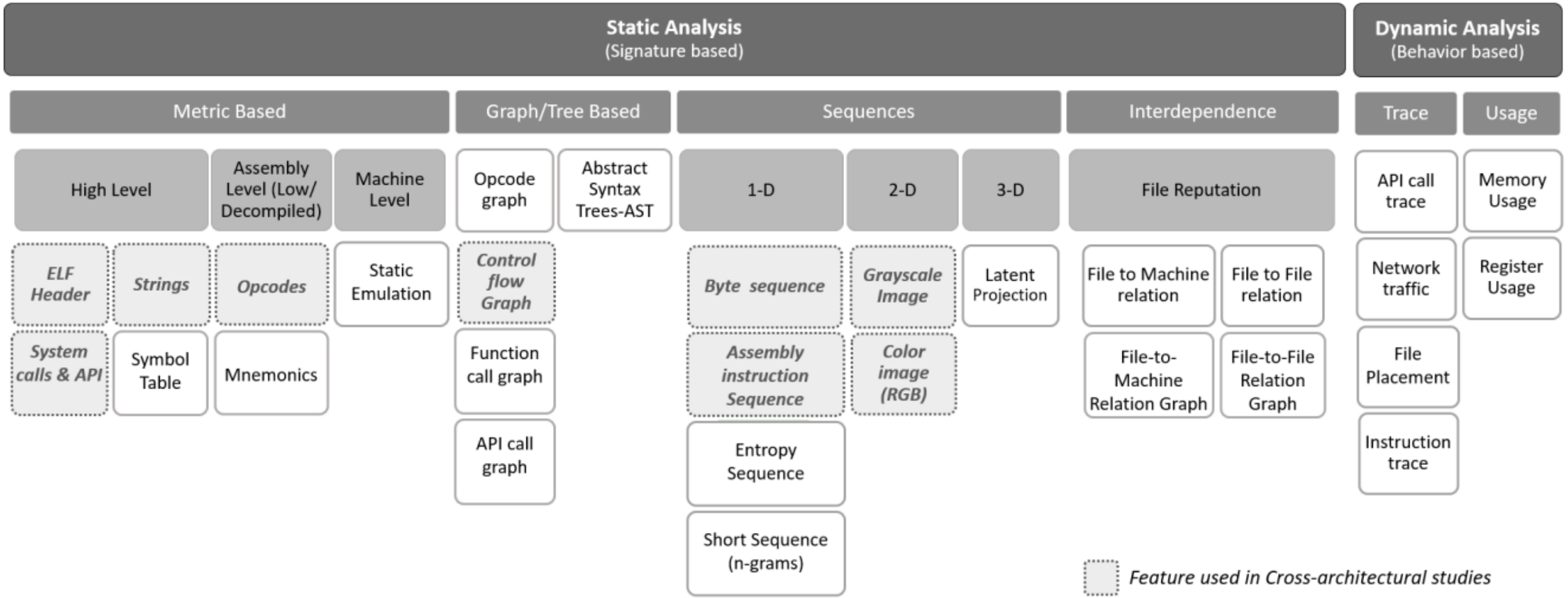


FIGURE 6. Taxonomy of ELF feature representations.

A: METRIC BASED

1. High-Level Features:

- ✓ High level informative metrics that can be extracted from binaries and used as features includes the following:

  A. ELF Header: ELF Header stores rich structural information that is important to support the framework , such as file's magic data, class (32-bit or 64-bit), entry points, target application binary interface (ABI), file interpretation indicators etc.

  B. Strings: Strings may contain some human-readable strings or sequence of characters within the binary content such as IP addresses, DLL names, error messages, and code comments

  C. Symbol Table: It acts as the lookup table holding the location and relocation information of symbolic references in a binary file to sup port the processes of linking and debugging.

  D. System Calls and APIs: act as an interface to access the OS provided services such as file and device management operations, controlling processes and communications. APIs are system call wrappers written in high-level languages.

3.

TAXONOMY

A: METRIC BASED

2. Assembly-Level Features:

A. **Opcodes and Mnemonics**: Opcodes (Operation codes) are unique and atomic executable instructions close to machine code. **Opcodes** have proved to be more useful in detecting and classifying malwares. **Mnemonics** are a special form of opcodes with symbolic names that are self-explanatory and easily understood by humans.

- ✓ **Tables 3** (Below**)** and **4** (next slide) showcases such an example of architecture dependency using the Mirai botnet disassembly for its 'dvrHelper' function call
- ✓ Different notations are used for the same operation by different processor architectures. dvrHelper is a DDoS attack module equipped with features to bypass anti-DDoS solutions.

TABLE 3. Architecture dependency of Mnemonics for *dvrHelper* (Mirai).

	Mnemonic	
Architecture	**Function call**	**Move operation**
x86	CALL	PUSH
ARM	BL	LDR
MIPS	JALR	LW
PPC	BL	LI
SPARC	CALL	MOV

3.

TAXONOMY

TABLE 4. Disassembly showing 'dvrHelper' (Mirai) botnet call for different IoT processor architectures.

x86

```
0x080481f8      83c41c          add esp, 0x1c
0x080481fb      68ff010000      push 0x1ff                  ; 511
0x08048200      6841020000      push 0x241                  ; 577
0x08048205      683d840408      push str.dvrHelper          ; 0x804843d ; "dvrHelper" ; int32_t arg_8h
0x0804820a      8945e4          mov dword [var_1ch], eax
0x0804820d      e8f5feffff      call fcn.08048107
```

ARM

```
0x000081d8      20119fe5        ldr r1, [0x00008300]     ;[0x8300:4]=0x241 ; int32_t arg2
0x000081dc      84008de5        str r0, [var_84h]
0x000081e0      1c219fe5        ldr r2, [0x00008304]     ;[0x8304:4]=511
0x000081e4      1c019fe5        ldr r0, [str.dvrHelper]  ;[0x8368:4]=0x48727664 ;"dvrHelper" ;int32_t arg1
0x000081e8      b8ffffeb        bl fcn.000080d0
```

MIPS

```
0x004002a4      a7a2001e        sh v0, 0x1e(sp)
0x004002a8      8fbc0010        lw gp, 0x10(sp)
0x004002ac      24050301        addiu a1, zero, 0x301
0x004002b0      8f848018        lw a0, -segment.LOAD0(gp)   ; [0x440638:4]=0x400000 segment.ehdr
0x004002b4      8f998060        lw t9, -0x7fa0(gp)          ; [0x440680:4]=0x400100
0x004002b8      248405f0        addiu a0, a0, 0x5f0
0x004002bc      240601ff        addiu a2, zero, 0x1ff
0x004002c0      0320f809        jalr t9
```

PPC

```
0x10000274      90610010        stw r3, 0x10(r1)
0x10000278      3c601000        lis r3, 0x1000
0x1000027c      38800241        li r4, 0x241                ; int32_t arg2
0x10000280      38a001ff        li r5, 0x1ff                ; int32_t arg3
0x10000284      38630548        addi r3, r3, 0x548          ; int32_t arg1
0x10000288      4bfffe61        bl fcn.100000e8
```

SPARC

```
0x00010204      9010204d        mov 0x4d, o0
0x00010208      92102601        mov 0x601, o1
0x0001020c      d027bfe8        st o0, [fp+-0x18]
0x00010210      941021ff        mov 0x1ff, o2
0x00010214      11000040        sethi 0x40, o0
0x00010218      7fffffb5        call fcn.000100ec
```

A: METRIC BASED

3. Machine Level Features:

- **Static Emulation:** The static emulation is inspired by dynamic analysis on emulated environments using software tools like QEMU. Static emulation refers to the analysis of loadable parts of the program.

  - For instance, in Figure 3 (Slides 10 & 11), the segments PT _LOAD0 and PT _LOAD1 denote sections that will be loaded for execution during runtime.

3.

TAXONOMY

B: GRAPH/TREE BASED FEATURES

1. Graph-Based Features:

- They are an extended version of metric-based features discussed above, where the relationship among the features is also accounted for and expressed. The nodes in the graph usually represent the actual metric-based features like APIs,

1. Tree-Based Features:

- Abstract Syntax Trees (ASTs) are the tree representations generated using parsers over the code constructs found in a source code's syntactic structure, and tree-based machine learning approaches are later employed to learn the latent information they hold. Being a byproduct of the compiler's syntax analysis phase, ASTs are useful for analyzing or transforming programs to a more simplified view for better understanding.

3.

TAXONOMY

C: SEQUENCE BASED FEATURES

1. 1-D SEQUENCE:

A. **Byte Sequence:** It is a sequential representation of byte-level data present in binary files, where each byte is converted into an 8-bit integer (unsigned) and translated to numerical representation with values ranging from 0 to 255.

B. **Assembly Instructions Sequence:** Assembly instructions extracted from a disassembled binary are concatenated into a one-dimensional sequence. The operands and registers may be pruned out to reduce sequence length. Tokenization or embedding of the resulting sequence may be required.

C. **Entropy Sequence:** It is the sequence of rolling entropy obtained by scanning a series of short windows of byte sequences [28], assembly instruction sequences, or simply the whole file [8].

D. **Short Code Sequence:** This is a special case of very short byte sequences. They divide long sequences into several disjoint or overlapping short sequences, typically comprising sequences of 2 to 11-byte length, generally called n-gram byte sequences, where 'n' denotes the sequence length.

3.

TAXONOMY

C: SEQUENCE BASED FEATURES

2. 2-D SEQUENCE:

A. **Gray Scale Image**: A two-dimensional image-like representation is obtained by reshaping and then resizing the one-dimensional byte sequence representation discussed above. Such 2-dimensional representations are usually downscaled to avoid computational overheads.

B. **Color Image:** It is an extension of the grayscale representation described above, where conversion to a colored format is done by extending grayscale values to RGB channel values using tools like BinVis.

3. 3-D SEQUENCE:

A. **Latent Projection**: Unlike dynamic analysis, the use of three-dimensional projection of latent information is still largely unexplored for static analysis

3.

TAXONOMY

D: INTERDEPENDENCE

- The features for static analysis discussed so far dealt with the structural properties of an ELF binary, its code-level properties, and its section and segment-level components. All of them are obtained from within the binary, hence, treated as 'Intra-file' properties. The 'interdependence' deals with the properties that are external to the binary and is concerned about its proximities with the surrounding environment.

A. **File to Machine Relation**: t represents the absolute or relative path information of a binary file which could provide contextual information with the capacity to reveal benign or malicious intents .

B. **File to File Relation:** It deals with the influences that a file inherits directly or indirectly from co-occurring files in the environment [60], [71], [72]. The variations in the importance of such relations to a malware file as opposed to a benign file help isolate malicious files.

3.

TAXONOMY

E: DYANAMIC ANALYSIS

- Dynamic analysis was not in the scope of this survey paper. However, authors provided a high-level taxonomy of dynamic features observed in the literature that can be categorized into traced-based and usage-based features.

A. **Traced Based:** These features deal with acquiring knowledge about malware activities and interactions over a period of time, such as tracing the API calls made by malware, tracing the sequence of instructions they executed, and their network interactions

B. **Usage Based:** These features deal with monitoring the usage of system resources such as memory, registers, and file access.

C. **File Placement:** It is a specialized file monitoring technique where files are placed in suspected locations of malware activity. For instance, when ransomware tries to access the file to steal information, its file system activity and access behavior are recorded for taking remedial actions in reality.

3.

TAXONOMY

SECT~ 4… (ABIR EL.)