BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2019
http://excel.fit.vutbr.cz

LiSa – Multiplatform Linux Sandbox for Analyzing IoT Malware

Daniel Uhříček*

Abstract
Weak security standards of IoT devices levereged Linux malware in past few years. Exposed telnet and ssh services with default passwords, outdated firmware or system vulnerabilities – all of those are ways of letting attackers build botnets of thousands of compromised embedded devices. This paper emphasizes the importance of open source community in the field of malware analysis and presents design and implementation of multiplatform sandbox for automated malware analysis on Linux platform. Project LiSa (Linux Sandbox) is a modular system which outputs json data that can be further analyzed either manually or with pattern matching (e.g. with YARA) and serves as a tool to detect and classify Linux malware. LiSa was tested on recent IoT malware samples provided by Avast Software and it solved various problems of existing implementations.

Keywords: IoT — Malware — Linux — Security — Dynamic Analysis — Network Analysis – Yara

Supplementary Material: LiSa repository — Disspcap repository

*xuhric00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Community of malware analysts was primarily focused on Windows platform for many years. Current rise of Internet of Things and embedded devices connected to the Internet puts more focus of malware creators on Linux based systems. In order to keep up with recent malware, sharing resources is necessary.

VirusTotal serves as a great example as it has free API to submit file to scan in more than 50 anti-viruses and file scanners and it offers a possibility to share newly found samples to the community. Another project serving community and also propagated by VirusTotal is YARA[1] – an open source pattern matching engine which helps to detect and classify malware. When using YARA, malware samples are described through *rules* in simple readable format.

Recent milestones include open sourcing RetDec decompiler[2] by Avast Software (December 2017) and public reselase of Ghidra[3] by NSA's (March 2019).

Manual analysis of potential malware is time expensive task even with tools like RetDec and Ghidra. Core of proposed solution is automated malware analysis tool which is easy extensible. Extensibility and modularity is important because of dynamics of information security field. Solution should be able to reliably emulate target architecture and thus be able to overcome anti-debug protections. Focus is given to dynamic (behavioral) and network analysis.

Some specialized tools for automated binary analysis already exist [1, 2]. The most popular one is Cuckoo. Cuckoo is an industry leading automated

[1] https://virustotal.github.io/yara/

[2] https://github.com/avast-tl/retdec

[3] https://github.com/NationalSecurityAgency/ghidra

malware analysis sandbox. Cuckoo's results are great on Windows platform. However, Linux analysis lacks many features. Another downside of Cuckoo is that users must configure sandbox and prepare target images by themselves.

Next solution is Limon. Limon is Linux analysis sanbox which starts its guests as VMware virtual machines and thus it is limitted only to i386 and x86-64 architectures.

REMnux is a system providing automated analysis on Ubuntu. REMnux submits samples to VirusTotal API, statically analyzes ELF header. Dynamic analysis is again only implemented with strace. Moreover, REMnux lacks enough target architectures.

Detux supports 5 different processor architectures (i386, x86-64, ARM, MIPS, MIPSel). It uses Debian emulated by QEMU[4]. Nonetheless, Detux analysis is limited only to basic static and network analysis. Static analysis part extracts strings in binary and parses ELF information from readelf command.

Padawan sandbox significantly improved multiplatform linux malware analysis. Static analysis uses custom IDA Pro scripts. These scripts go through disassembled/reversed code and aggregate statistics. Sandboxing uses QEMU because it supports most architectures. Dynamic analysis uses SystemTap to monitor low-level kernel probes and user probes. Padawan sandbox exists only as a service. It has no available source code and neither it can be downloaded.

Considering custom needs in analysis and need of supported architectures, the decision was to implement a new solution. Final product – project LiSa – offers full analysis environment with core utilities for static, dynamic and network analysis with possibility to define extension modules. It is simple to setup, generate output jsons and write YARA rules, to be useful for community, corporate-level companies or hobbyists securing their own home network.

2. Linux Malware Analysis Challenges

Initial experimenting with malware took a big part of product specification. As an example, I will use well-known Satori, a Mirai variant, on MIPS platform. Mirai [3, 4, 5] is a malware family used to create botnets and conduct Distributed Denial-of-Service (DDoS) attacks. Mirai was encountered in 2016 by malware analysis team MalwareMustDie. Botnet began to spread rapidly and reached its peak, when up to 600,000 devices – bots – were able to simultaneously attack. Bots were mostly composed of IP cameras, DVRs, SOHO routers and other devices equiped with busybox type shell.

[4]https://www.qemu.org/

```
header->e_shoff = 0xffff;
header->e_shnum = 0xffff;
header->e_shstrndx = 0xffff;
```

Figure 1. Part of leaked Mirai source code which manipulates ELF header in order to prohibit analysis in GNU Debugger (gdb).

Already after running standard Linux tools as readelf or gdb, we can see that binary was manipulated to prevent analysis. Symbols were stripped away and section headers were hidden by setting invalid offsets. This technique might be seen also in the source code repository of Mirai (see Figure 1).

Modifying section header offsets, section headers count and string table offset (fields `e_shoff`, `e_shnum` and `e_shstrndx`) prohibits proper functionality of readelf as it hides desired values. Moreover, this manipulation causes malfunction of objdump and gdb – both of these tools signal that file format was not recognized. These anomalies however do not prevent actual execution of a program.

2.1 Program Tracing

One of the most informative dynamic analysis techniques is syscall analysis [6]. In order to get information about syscall we might use strace tool. However after running binary with strace, another anti-debugging technique was found. Sample used:

```
ptrace(PTRACE_TRACEME, 0, 0, 0)
```

Thanks to -1 (not permitted) result of ptrace syscall, sample found out it was being traced. Since ptrace syscall is widely used, this simple anti-debugging technique prohibits proper functioning of strace, ltrace, gdb and others.

One anti-anti-debugging solution could be using `LD_PRELOAD`[5] – environment variable that lists shared libraries to be loaded before the standard ones. We can easily specify custom shared library with our own implementation of ptrace syscall. Simple implementation of ptrace wrapper involves calling real ptrace on its address found by dlsym[6]. However, this approach is viable only if the binary was dynamically linked.

Most of current Linux malware is statically linked – more than 80 % of 10,548 samples in dataset by authors of *Understanding Linux Malware* paper [2]/ closed-sourced Padawan sandbox were statically linked.

[5]https://linux.die.net/man/8/ld.so
[6]https://linux.die.net/man/3/dlsym

In this case, static linking is used to ensure portability on multiple devices.

As a demonstration how to evade anti-ptrace method, I have prepared a tool which searches for ptrace syscalls and replaces them with an instruction to set output register to 0. In case of MIPS, the tool is searching for `syscall` instruction and in previous instructions finds syscall number which has been placed into register `v0`.

2.2 Kernel-level Tracing

Instead of evading different anti-tracing method, approach I chose was to omit user-level tools like strace and focus on kernel-level[7] tracing [7]. In Linux, we may gather information from three main event sources:

- Tracepoints – Statically defined events in kernel.
- Dynamic Probes – Kprobes (kernel probes) and uprobes (user probes). Using probes, we might attach our code to both kernel and user-level functions.
- PMCs (performance monitoring counters) – low level monitoring of CPU cycles, number of instructions, etc.

These event sources are accessible from various front-ends. I considered using perf, ftrace, eBPF and SystemTap.

Perf is part of kernel tree (`/tools/perf`). It is relatively easy to use and offers wide scope of metrics (hardware metrics, scheduler, kernel-level and user-level probes). Ftrace is also part of kernel upstream and it is available through virtual filesystems debugfs and tracefs. Those are usually mounted to `/sys/kernel/debug/`. Extended Berkeley Packet Filter (eBPF) is part of recent kernel versions. Its architecture consists of small virtual machine with JIT (just in time) compilation.

SystemTap unlike previous mentions is not part of kernel upstream. It is robust, powerful and widely programmable monitoring/tracing tool. SystemTap should support most architectures. If we want to use full functionality of SystemTap, it is needed to compile kernel with multiple flags to enable tracing and debug symbols. Moreover, everything has to be compiled with `-g` flag.

3. Analysis Pipeline Proposal

Considering experiments from previous section and other manual examination of IoT malware, I defined following pipeline for automated analysis (see Figure 2):

Top Level Analysis creates beginning of the pipeline. It serves as a director of a whole analysis. In this part of the pipeline, binary file is pre-analyzed in order to provide necessary meta-data for other analysis modules (e.g. file format details or CPU architecture).

Static Analysis searches for relevant static patterns using one of many static analysis tools. Considered tools are mainly readelf, objdump, pyelf, radare2 and RetDec fileinfo. Dynamic Analysis traces running binary, newly created processes, filesystem operations and system calls.

Pipeline should be easy extensible so that users can define their own analysis modules. Example of custom module is VirusTotal module which calls free VirusTotal API to get malware scan results.

Outputs of individual modules are combined into final JSON output. This final output can be then further processed. Further processing can be either manual in web graphical user interface or by custom module of YARA.

4. Implementation

This section explains key points in sandbox and automated malware analysis implementation. Languages of implementation are Python (modular system, analysis pipeline and web API), bash (build systems) and C++ (parsing network data).

4.1 Emulating Target Architecture

First step to run and analyze sample on target architecture was to prepare emulation environment. IoT malware is being built for wide spread of architectures. This lead to choosing open source project QEMU as only viable option to emulate prepared system. QEMU supports many architectures including MIPS, ARM, SPARC, AArch64, PowerPC.

Target system consists of self-built Linux kernel and prepared filesystem with analysis tools. In order to cross-compile images, buildroot[8] project was used. Buildroot is a common tool that helps to develop Linux for embedded systems.

4.2 Dynamic Analysis

As it was already mentioned in previous sections, the key to implement dynamic analysis was omitting user-level tools ptrace syscall and prepare kernel-level tracing. SystemTap was chosen as the final solution because it offers simple C-like language to define probes and malware analysts can more easily change or add

[7] Brendan Gregg explains very well kernel tracing features on his blog – http://www.brendangregg.com/blog/

[8] https://buildroot.org/

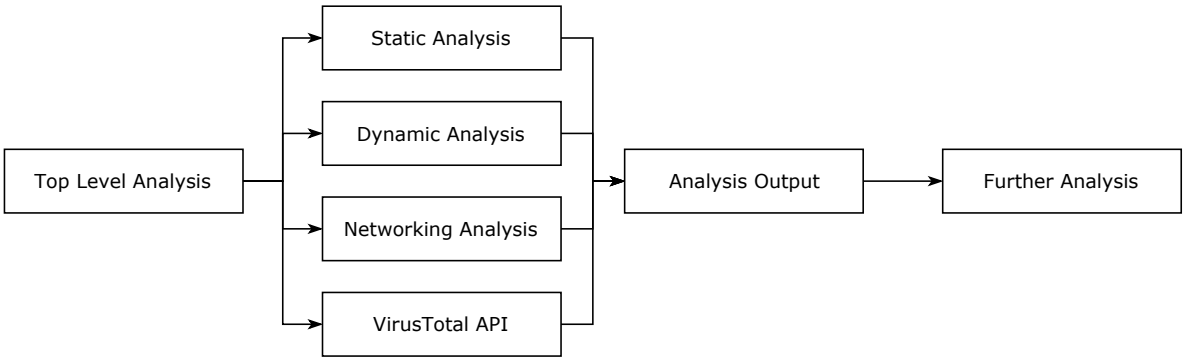

Figure 2. Proposed analysis pipeline: Top Level Analysis orchestrates sub-analysis modules and it gathers their outputs to final output JSON. The output is then further processed manually by malware analysts or with pattern matching engine YARA.

functionalities to dynamic analysis. Current implementation uses probes to create a process tree, trace syscalls and mark open or deleted files.

The build process is as follows:

1. Translating SystemTap (.stp) script to C language.
2. Cross-compiling C code to kernel module for target architecture against already prepared kernel.
3. Inserting built kernel module to target filesystem.

4.3 Network Analysis

Network analysis module loads tcpdump pcap file and provides packet-by-packet analysis. In the initial prototype, I used Python libraries – Scapy and dpkt. Both of those proved to be slow when analyzing and they had even problems to load bigger pcap files. Thus I have prepared C++ library with bindings to Python to parse pcap data.

First important functionality implemented in network analyzer is endpoint examination. Analyzer uses free Geolite2 database of countries, cities and ASN (Autonomous System Numbers). IP address is then searched in multiple blacklists[9].

Other general metrics include port statistics, amount of data trasfered for each endpoint, TCP SYN, TCP FIN packets. Finally, analyzer provides L7 analysis. Outputed L7 information currently consist of parsed DNS queries, HTTP requests, IRC messages and telnet data.

Network anomalies are also noted. Those anomalies are e.g. hitting blacklisted IP address, sending malformed packets or obvious IP and port scanning.

[9] https://github.com/firehol/blocklist-ipsets

5. Containerization and Deployment

This section briefly describes final architecture of LiSa, docker containerization, connection of individual sandbox elements, and deployment. Individual sandbox parts are:

- Flask API – Web API has endpoints for creating new tasks (either full file analysis or just pcap analysis), for viewing status of task, results and for downloading analysis related files (e.g. captured pcaps).
- RabbitMQ – It is used as a message broker using AMQP protocol. It stores and controls queues of tasks coming to the sandbox.
- MariaDB – Open source MySQL alternative. It stores analysis task results.
- Nginx – In implemented system architecture, nginx proxy passes incoming requests to the uwsgi server (running web API).
- Sandbox workers – Worker nodes running analysis pipeline.

For simple deployment and transfer from testing to production environment, docker was used. Docker is containerization platform. Containers use same kernel as host machine so they have minimal overhead. Configuration files – Dockerfiles – are composed of instructions. Dockerfile instructions describe process of building final docker images. These docker images carry all of their dependencies and should run properly on all systems with installed docker.

I prepared docker-compose file which suits best for running multi-container applications. It is possible to simply run:

```
docker-compose up --scale worker=10
```

to start full system and scale amount of sandbox workers to fulfill all requests.

6. Dataset and Experiments

In order to experiment and evaluate LiSa's analysis results, 150 IoT malware samples were chosen to analyze. These samples were mostly built for ARM architecture (see Table 1. for samples' architecture distribution). More then 80 % of samples in dataset were statically linked.

Table 1. Architecture of samples in dataset.

Architecture	Amount
ARM	57
MIPS R3000	52
Intel 80386	23
x86-64	13
Aarch64	5

Moreover, previosly captured 20 GB of pcaps containing various IoT malware internet traffic were analyzed.

Behaviour seen in analyzed malware included:

- IP scanning – The most common behaviour in our dataset. Since most of the current IoT malware creates botnets, its targets – bots – spread by finding vulnerabilities in other devices. LiSa differentiates between local network and Internet scanning and triggers anomaly when such an event occurs. Usual target port was port 23 (SSH).
- Port scanning – Malware was scanning many ports in local network to find which ports are open.
- HTTP requests – Multiple unusual HTTP requests were detected and marked as an anomaly. HTTP requests could contain either garbage or target endpoints for data acquisition (e.g. `GET /version`).
- Malformed DNS packets – These packets contained high numbers in header values (number of questions, answer, authority and additional resource records).
- IRC – Malware connected to IRC server and was waiting for commands from CC server.
- Targeting specific countries – Sample was contacting hundreds of hosts only in specific countries. These countries were Vietnam, China, Bangladesh, Thailand and India.
- Changed process name – Syscall `prctl` with request `PR_SET_NAME` was used in order to hide malware process.
- Blacklisted endpoints – Several endpoits' IP addresses were found on blacklist.
- Targeting specifing application – Sample was targeting Wordpress applications and their endpoints `/wp-login.php`.
- Ptrace – Ptrace syscall was detected as an anti-debugging technique.
- Process creation – Detected forked processes commonly served for daemonizing application.

7. Conclusions

This paper presented open source IoT malware analysis tool. Taking account dynamic field of IT security, extensible analysis pipeline was presented. The paper stated multiple problematic parts as anti-debugging techniques and architecture emulation.

Main contributions of this project are, firstly, broad network analysis, detection of anomalies and implementation of C++ library with Python binding that overcomes commonly used packages Scapy and dpkt. Secondly, it is preparation of SystemTap monitoring environment and its cross-compilation toolchain. Using kernel-level analysis could be expanded to full system monitoring solution. This solution would have similar functionalities as Intrusion Detection System or antivirus but with minimal overhead and thus suitable for embedded systems.

Final sandbox – LiSa – can help researchers to analyze Linux malware, write YARA rules and detect newly discovered malware.

Acknowledgements

I would like to thank Doc. Dr. Ing. Dušan Kolář, David Jursa and Ing. Jakub Křoustek, PhD. for their guidance.

References

[1] Olaboyejo Olowoyeye. *Evaluating Open Source Malware Sandboxes with Linux malware*. PhD thesis, Auckland University of Technology, 2018.

[2] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Understanding linux malware. In *IEEE Symposium on Security & Privacy*, 2018.

[3] C. Kolias, G. Kambourakis, A. Stavrou, and Voas J. *DDoS in the IoT: Mirai and Other Botnets*. In *IEEE Computer, 50(7)*, pages 80–84. IEEE, 2017.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas,

and Yi Zhou. *Understanding the Mirai Botnet*. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, 2017. USENIX Association.

[5] Ya Liu and Hui Wang. *Tracking Mirai Variants*. In *VB2018*, Montreal, 2018.

[6] Jim Keniston, Ananth Mavinakayanahalli, Prasanna Panchamukhi, and Vara Prasad. Ptrace, utrace, uprobes: Lightweight, dynamic tracing of user apps. In *Proceedings of the 2007 Linux symposium*, pages 215–224, 2007.

[7] Ananth Mavinakayanahalli, Prasanna Panchamukhi, Jim Keniston, Anil Keshavamurthy, and Masami Hiramatsu. Probing the guts of kprobes. In *Linux Symposium*, volume 6, page 5, 2006.