DeepBits Technology LLC

www.deepbitstech.com

Innovate The Analytics For Securer Cyberspace

DEEPBITS TECHNOLOGY

Introduction

Backed by over ten years' top-notch research on cybersecurity analytics techniques funded by NSF, DARPA, ONR, AFRL, etc., DeepBits Technology LLC is constantly innovating the analytics for securer cyberspace.

Our Innovation:
We have built two novel analysis techniques for cyber security applications – "Whole-system emulation based dynamic binary analysis", and "Deep-learning based binary code similarity detection".

The first technique utilizes virtualization based techniques to extract dynamic behaviors of malware. Compared with current VM-based analysis techniques, our techniques can provide better analysis capabilities and much lower R&D expenditure.

The second technique employs deep learning for binary code similarity detection. It generates "code gene" for every piece of code and can quickly search the "code gene database" like searching keywords in google.

Our Products:

On top of our two innovative techniques, we have created two unique solutions for malware intelligence and N-day vulnerability discovery respectively.

Holmes Malware Intelligence provides essential information when people handle cyber incidents. Holmes is designed to generate actionable intelligence. The intelligence not only include 0-day malware detection, malware classification, but also show evidence to correlate the malicious samples to suspicious crime groups in a fully automatic way.

EagleEye Discovering vulnerabilities in an IoT ecosystem is like finding a needle in a haystack, even when we are dealing with known vulnerabilities. EagleEye is designed to scan N-day vulnerability in IoT devices. It supports binary only analysis (No source code is required) and provide scalable vulnerability search on cross-platform (ARM, mips, etc.). Given a firmware image, it can search all vulnerabilities in the vulnerability database. Or given a newly identified vulnerability, It can search all firmware images in the firmware database.

Our Innovation

Deep Learning based Code Similarity Detection

Given two binary functions, we would like to detect whether they are similar. This problem is known as "binary code similarity detection", which has many security applications, such as plagiarism detection, malware detection, vulnerability search, etc. This binary code similarity detection must be efficient and scalable, since we are dealing with a large volume of malware samples, firmware images, and labeled functions and function groups in our knowledge base.

Existing works are expensive and inaccurate. There have been plenty of works that try to detect similar code in binary executables, from simple syntax-based solutions like n-gram, to control-flow graph based approaches like BinDiff, to the most expensive symbolic execution and theorem proving like BinHunt. Recently development focus on cross-architecture support . These methods all lack in accuracy, and most of them are fairly expensive and do not satisfy our needs for processing large volume of malware samples and search over a large knowledge base.

Our approach: code graph embedding. To address this scalability problem, our approach learns high-level feature representations from the control flow graphs (in short, code graph) and employs deep learning to encode (i.e., embed) the graphs into embeddings (i.e., high dimensional numerical vectors). In doing so, we can index the functions in a large corpus using embeddings via locality sensitive hashing, so as to reduce the online search time to within a split of second per query.

Whole System Emulation based Dynamic Analysis

The virtual machine based malware analysis engine has become a standard equipment for antivirus companies like FireEye, Pan Alto Networks, etc. However, the current design conducts in-VM monitor and needs to be constantly updated to reflect the system changes. Thus, this design requires huge resources to support different platforms (Windows 7/10, Mac, Android, etc. Our analysis engine, DECAF, requires much less resources for multiple platforms support, and has several salient features:

1. **Cross-platform support.** DECAF's plugin API is engineered to hide many architecture and OS specific details. Therefore, we can easily extend our tool to support malware analysis on mobile and IoT platforms. DECAF has provided support for Windows, Linux, Android systems.
2. **Whole-system view**. AS a virtualization-based analysis platform, DECAF is able to reveal malicious behaviors happening anywhere in the software stack, including user-level processes, OS kernel, boot loader, and even BIOS.
3. **Dynamic code instrumentation management and optimization**. DECAF provides a powerful code instrumentation interface and conducts code optimization under the hood.
4. **Precise and lossless tainting.** DECAF maintains bit-level precision taint information for every bit of registers and memory locations and applies precise tainting rules for most instructions. This capability allows DECAF to detect and analyze spyware, and analyze how malware reacts on specific trigger conditions, etc.

Our Product - Holmes

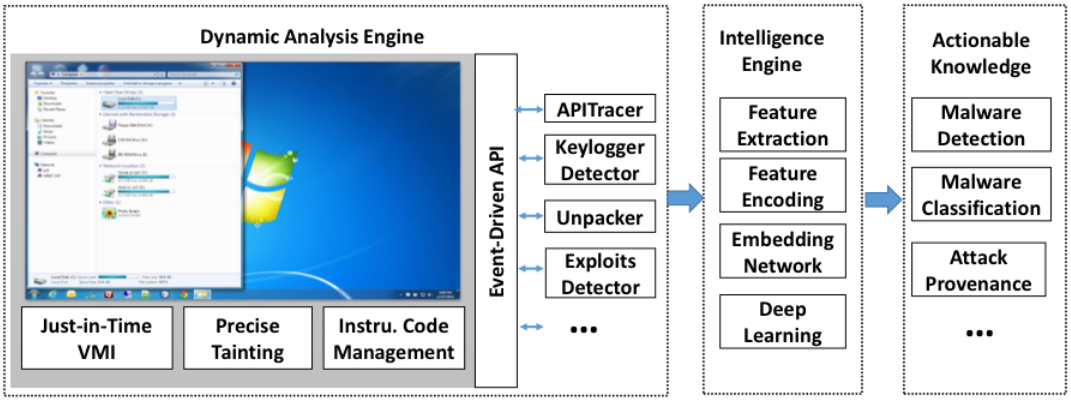

Figure 1 Overview of Malware Intelligence Engine - Holmes

Malware has been a long-standing as well as increasingly complex cybersecurity problem. Traditional signature based detection and manual reverse engineering approach can no longer keep up with the pace of increasingly sophisticated obfuscation and attack techniques. As a simple fact, the number of malware samples emerging on the Internet is growing exponentially year-over-year, and existing tools and procedures for malware analysis do not sufficiently assist defenders in extracting actionable knowledge from this enormous malware dataset.

Our analysis platform **Holmes** effectively combines the state-of-the-art binary analysis techniques and newly emerging deep learning techniques to expose malicious behaviors, correctly unpack and disassemble malicious code and data, and attribute the malware (or its subcomponents) to the related families, and possibly trace back to its authors.

Figure 1 illustrates the overall architecture of our malware analysis platform. It comprises two core components - Dynamic Analysis Engine and Intelligence Engine. The dynamic analysis engine is armed with Just-in-Time virtual machine introspection(VMI), dynamic taint analysis, instrumentation management, etc. Its capabilities can be extended through its event-driven APIs to finish different analysis tasks. The suspicious binary sample is first fed to the dynamic analysis engine for unpacking and correct disassembly. Then different binary code level features are extracted from the disassembly. These features are piped to the Intelligence Engine. The Intelligence Engine conducts feature learning and neural-network based feature encoding (as known as embedding). Deep Learning is finally applied to process these complex data to generate actionable knowledge (malware detection and classification, attack provenance) for malware intelligence.

Our Product-EagleEye

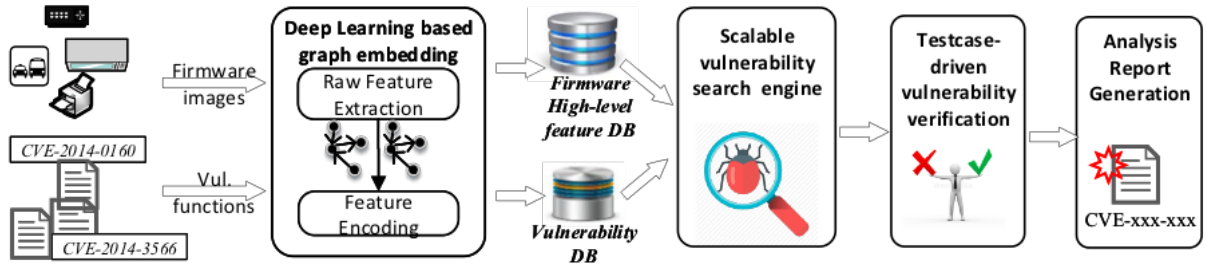

Figure 2 Overview of N-day vulnerability search engine EagleEye

Figure 2 illustrates the overall architecture of our N-day vulnerability search engine **EagleEye**. The inputs are IoT firmware images and binary functions containing vulnerabilities (labeled by the CVE numbers). Both functions in firmware images and vulnerable functions go through the same deep neural network-based encoding process that generates a numeric feature vector (or embedding) for each function. The embeddings for the firmware functions and vulnerable functions are indexed and stored in firmware database and vulnerability database respectively, by using Locality-Sensitive Hashing (LSH).

With these two databases indexed using LSH, a search can be done in constant time. In particular, we consider two search scenarios:

1. Given a firmware image, search all vulnerabilities in the vulnerability database. In this case, we will extract all functions from this image, generate embeddings for them, and search similar embeddings in the vulnerability database.
2. Given a newly identified vulnerability, search all firmware images in the firmware database. In this case, we generate an embedding for this new vulnerability, and search similar embeddings in the firmware database.

To further validate the search result and remove false positives, we conduct testcase-drive vulnerability verification. For each vulnerable function, we first generate a set of testcases to cover its different code paths. Then for a candidate function in the search result, we feed each testcase and observe its output. If the candidate and the corresponding vulnerable function take the same testcases and produce the same outputs, then we consider them to be equivalent, and the candidate is a true positive.

Finally, an analysis report is generated for user-friendly interpretation of the query results.