Mapping memory access using controlled channel

Radu-Alexandru Mantu, BCS
Mihai Chiroiu, PhD

ABSTRACT

Recent strides in research based on side channel attacks reveal that data obtained by means of an untrusted operating system can empower an attacker to extract large bodies of sensitive information even from protected applications.

We improve upon existing attacks by eliminating the prerequisite of manually analyzing the binaries involved at runtime. We introduce an automated method of determining the expected behavior of a program when exposed to certain external factors.

CONTACT

Radu-Alexandru Mantu
Email: andru.mantu@gmail.com
Phone: 0729 589 162

Mihai Chiroiu
Email: mihai.chiroiu@cs.pub.ro

INTRODUCTION

During the past few years we have seen the emergence of systems able of provisioning user space applications with safety guarantees against a compromised kernel by making use of trusted hardware such as Intel SGX [1]. Haven [2], the first platform to utilize this neoteric technology to shield unmodified legacy applications, was demonstrated to be susceptible to side channel attacks [3].

The core element of this attack is a modified kernel that is able to corrupt the page table of a user space application. This action allows the attacker to produce irregular exceptions that are normally covertly handled by the kernel and that leave a trace of accessed memory areas.

The preexisting attack consists of two stages:
1) The offline analysis of the application's binaries that reveals branching execution paths that are dependent on the input.
2) The analysis of the trace yielded by the kernel's intervention. This, in conjunction with the knowledge gained from the offline analysis, can lead to the recovery of the data used during runtime (see Figure 1).

Our objective is to eliminate the former of the two steps, removing the need for the attacker to have access to the application's binaries. We prove that the same information that was gained from the offline analysis can be extracted from the very traces used in the second analysis step.

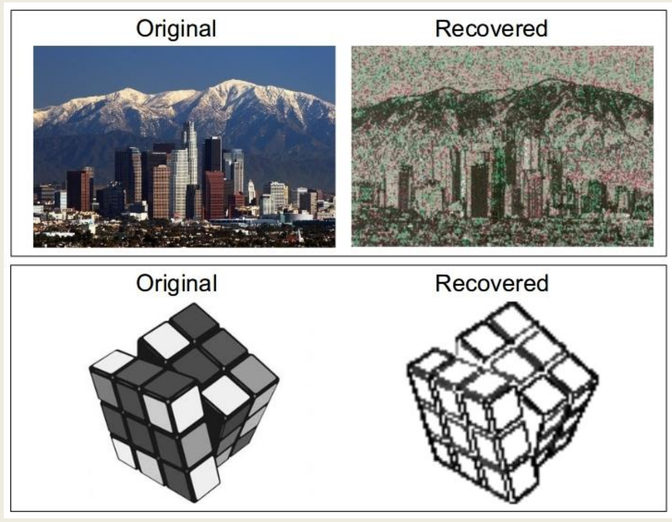

Figure 1. Recovered Images from InkTag [3]

RESULTS

Our method of offline analysis yields a directed graph where the nodes represent the totality of detected accessed addresses and the edges represent immediate transitions from one address to another (see Figures 2, 3). This graph is constructed in stages by extracting previously unknown information from new traces.

We obtain these traces by exposing the application to varying input that we know will produce a certain effect in the execution flow (e.g. pressing a button will lead to the application connecting to a database). When integrating the newly obtained information in the graph, we can specify what function this sequence has within the application. Our system can thus categorize the nodes (addresses) by the functionality which they represent and even by the files mapped at those addresses, if any. This last bit of information is guaranteed to be provided by the modified kernel.

The strategy involved in the training process is to obtain progressively more complex traces. For instance, the first trace should offer a foundation for future additions: opening and closing the target application. Next, we obtain a trace for a specific function of the application and our system will separate the new, unique path from the base path. Finally, we can explore cases where the application is terminated unexpectedly during the execution of the added function and create new junctures in its path. Once the training phase is complete, we can follow the execution path of a test memory access trace and identify the invoked functions.

OUR WORK

Our efforts were dedicated toward these three tasks:
1) Implementing a kernel tracing system capable of handling complex, multi-threaded applications that can spawn subordinated processes.
2) Creating and implementing a model that still has access to the binaries but streamlines the analysis process to create testing references for the main model.
3) Creating and implementing the model that generated the results previously mentioned.

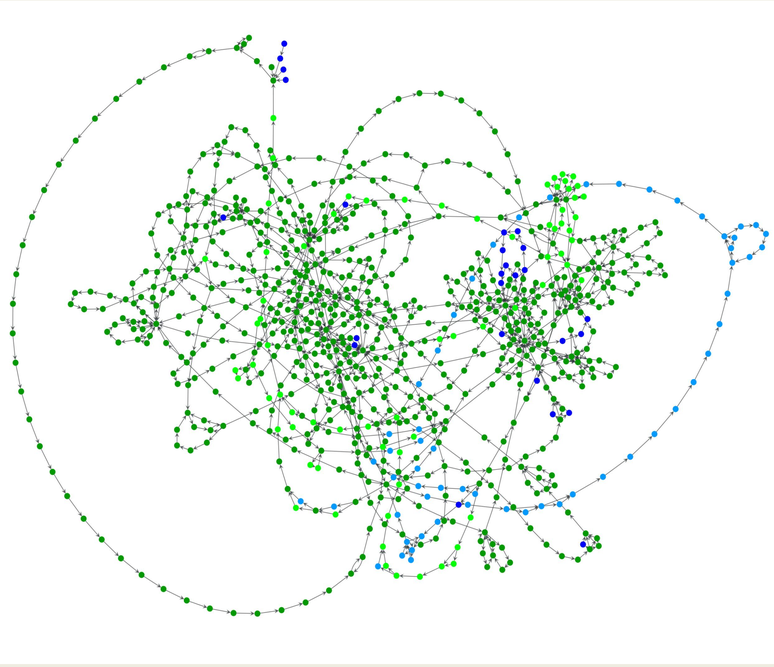

Figure 2. Execution paths for the loading/closing of an application (green) and a certain function (blue). Lighter shades represent areas mapped to libc while darker shades represent areas mapped to the loader

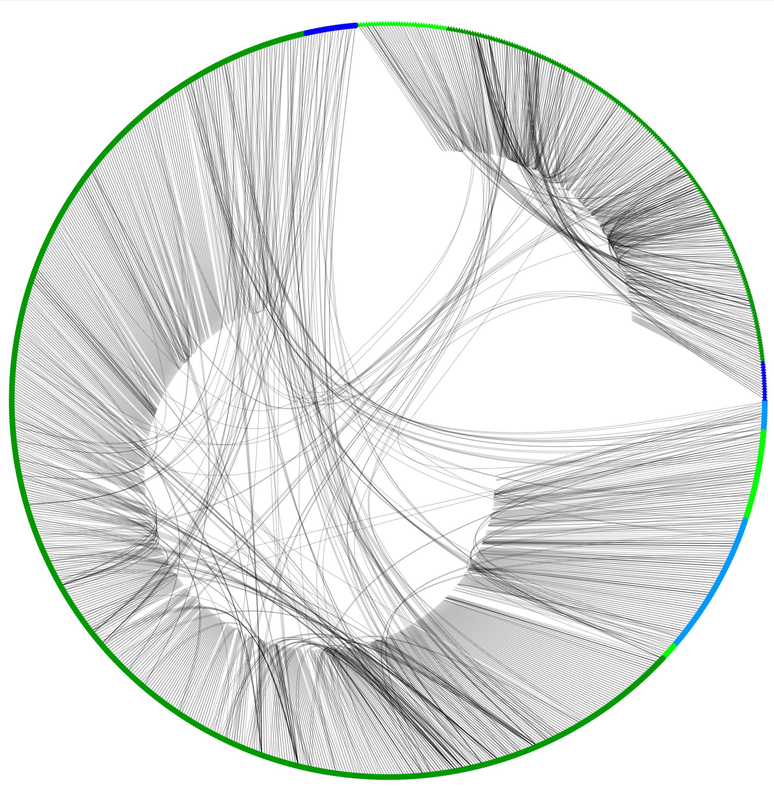

Figure 3. Illustration of the sequentiality of the memory accesses from Figure 2

REFERENCES

[1] V. Costan, S. Devadas. *Intel SGX Explained*

[2] A. Baumann, M. Peinado, G. Hunt. *Shielding Applications from an Untrusted Cloud with Haven*

[3] Y. Xu, W. Cui, M. Peinado. *Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems*