PROTECTING SENSITIVE INFORMATION USING DYNAMIC DATA-FLOW ANALYSIS

Pavel Zůna

Doctoral Degree Programme (2), FIT BUT

E-mail: xzunap00@stud.fit.vutbr.cz

Supervised by: Petr Hanáček

E-mail: hanacek@fit.vutbr.cz

Abstract: Security mechanisms of modern operating systems are most often overcome by exploiting vulnerabilities or social engineering to hijack privileged user accounts. To counter these threats, this paper introduces a complementary approach that uses dynamic data-flow analysis to protect sensitive information contained in memory instead of logical memory regions such as files. It proposes a state-based data-flow model for formal definition and verification of security policies. It also discusses possible implementation and deployement implications of such a system in production environments.

Keywords: security, security policies, data-flow analysis, data loss prevention, operating systems

1. INTRODUCTION

Security mechanisms of modern operating systems that include protected processor mode, virtual memory with separate address spaces for processes and mandataroy access control are time proven to provide the basic attributes of a secure information systém. These attributes are namely integrity, accountability and confidentiality. Studies [1, 2] have shown that the major of security threats are vulnerability exploitation and social engineering to hijack privileged user accounts. When these threats become incidents, standard OS mechanisms can still (if setup correctly) provide the first two of the basic attributes, but confidentiality is always at stake. At this point we can no longer prevent the attacker from accessing sensitive information, but we can stop him from spreading it further.

This paper introduces a complimentary approach to operating system security that surfaced only recently. It focuses on analyzing data-flows from and to sources of sensitive information instead of files, directories and devices [1]. The basic idea is that sensitive information flows from an initial container to others and security policies are defined by restricting the flows. Containers in this context can be files and processes.

Commercial solutions based on this approach have emerged in the last couple of years under the broad business term of Data Loss Prevention or DLP. However, none of them uses sophisticated mechanisms of evaluating data-flows and implement only 'ad-hoc' logic.

This paper tries to solve the problem of creating a formal platform independent framework for modeling data-flow of sensitive information It proposes a model that can be used to define and verify security policies following the described approach to operating system security.

The framework discussed in the following pages targets only modern operating systems designed for Von Neumann architecture computers with virtual memory and a separate address spaces of processes. Notable examples of such operating systems are all POSIX compatible ones including Microsoft Windows and Linux.

2. STATE BASED DATA-FLOW MODEL

For the purpose stated in the introductory chapter, I propose the following state-based data flow model

$$(D, K, P, H, S, Sp, V, T) \tag{1}$$

where:

D: set of sensitive information types

K: set of possible containers for elements of D

elements of K are representations of files and processes

P: set of all possible processes in the system $P \subset K$

H: set of all container identifiers

elements of H are representations of filenames, file descriptors, handles etc.

S: set of all possible states of the model $S = (K \to 2^D) \times (K \to 2^K) \times (P \times H \to K)$

Sp: initial state of the model $S_p \in S$

V: set of system calls on the modeled operating system

T: transition function $S \times P \times V \to S$

It is based on the model proposed by M.Harvan [3].

2.1. SENSITIVE INFORMATION POISONING

The model has been challenged by my academic collegues as practically unusable because of a side effect that occurs with evaluating sensitive information profileration in this way. I have named this effect 'sensitive information poisoning' and is best described as uncontrollable proliferation of sensitive information into most or even all containers in the model as a direct implication of their direct and undirect connections.Fig. 1 shows the evolution of this effect.

Larger circles represent containers while small ones represent that sensitive information is possibly present. Arrows depict connections between containers resulting from file operation system calls being invoked. Process containers are highlighted by dashed borders.

To prevent this from happening, we need to automatically add limitations to process containers when they connect with a container containing sensitive information. These limitations should restrict them from connecting with other containers containing other types of sensitive information or those that contain no sensitive information at all. Automatic limitations can be easily incorporated into the model with a basic library of security policies. The last section of this chapter deals with details. Unfortunaly, the presented solution to the 'sensitive information poisoning' problem is also the cause of some practical disadvantages for end users as described in the Practical limitations chapter.

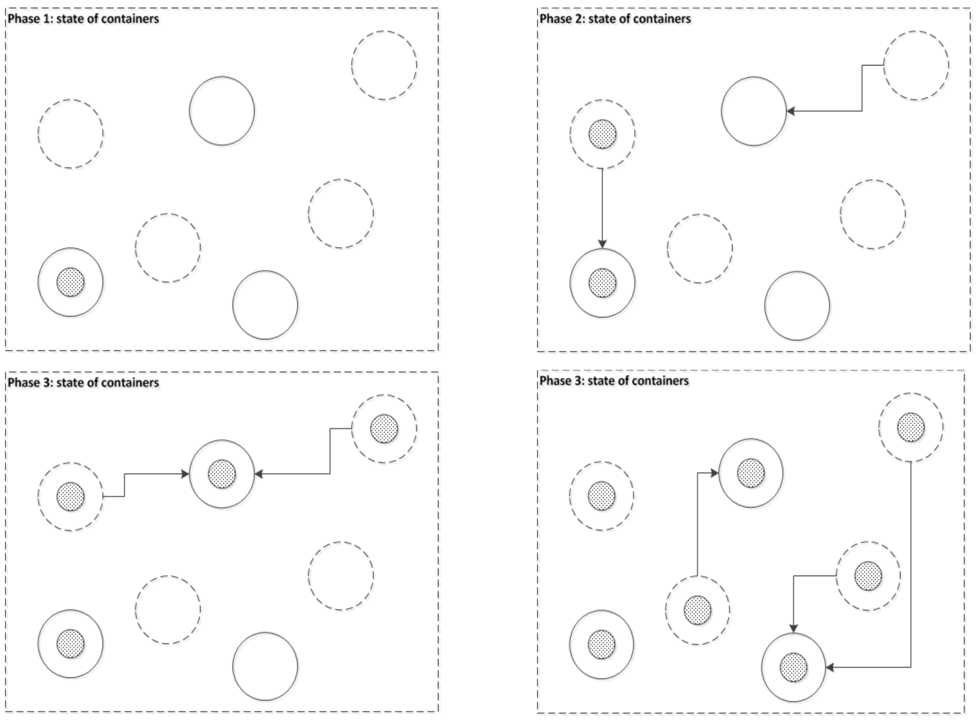
Phase 1: state of containers
Phase 2: state of containers
Phase 3: state of containers
Phase 3: state of containers

Figure 1: Evolution of sensitive information poisoning effect.

2.2. TRANSITIONS: SYSTEM CALL MODELING

Transitions between states of the model represent system calls being executed. There are as many transition types as there are different system calls on the particular operating system we're currently modeling. Each operating system is going to have its own set of transitions altought some are bound to be universal as operating systems that satisfy the conditions stated in the first chaper tend to follow the same basic principles.

To be able to define a transition representing a specific system call, we need to look at how its semantics affect the sets of our data flow model. Because states of the model are defined as triples of functions, we need an additional notation for specifying their changes. The easiest way to explain it is by example, so let us take a look at a teoretical transition for a system call that opens a file descriptor in eq. (2)

$$\begin{gathered}\forall s \in \left[C \to 2^D\right], \forall l \in \left[C \to 2^C\right], \forall f \in [P \times F \to C], \forall p \in P, \forall n \in F_{fn}, \forall rv \in F_{dsc} : \\ ((s, l, f), p, open(n, rv), (s, l, f[(p, rv) \to f(p, n)])) \in R\end{gathered} \tag{2}$$

Eq.2 translates to: if an open system call is successfully executed, process p opens a file with name n. The operating system return a file descriptor rv. This leads to the state being modified, so that a mapping from (p, rv) to the container named by f(p,n).

2.3. DEFINING SECURITY POLICIES

Using the proposed model makes it possible to define security policies as logical predicates with operators on sets of the model. For example:

3. IMPLEMENTATION

Implementation of a system, that would be able to maintain a digital representation of the proposed model with policies and apply transitions to it based on real-time monitoring of system calls being executed, is platform specific and is bound to faces challenges. The target operating system needs to provide an interface for evaluating system calls or directly incorporate the proposed system with other security mechanisms.

To effectively apply security policies that come with the model, the implemented system needs to block system calls that would result in a transaction that would put the model in a forbidden state. This process must be transparent to applications and should have the same consequences as if the system call was denied by the operating system.

3.1. PROOF OF CONCEPT

I successfuly created a proof-of-concept implementation on Microsoft Windows using a system call interposition technique known as 'API hooking'. It involves finding the addresses of system calls in process memory and replacing them with addresses of new placeholder functions [4].

In my implementation, the placeholder functions are used to apply transactions to the current state of the model, invoke the original system call and pass its return value to the calling process. That is, if the applied transactions do not violate predefined policies. In the opposite case, it reverts the transaction and passes an access denied return value to the calling process.

The current state of my proof-of-concept only evaluates transactions for basic system calls used for opening, reading, writing and closing files. Interprocess communication, clipboard and screenshot related system calls are not supported in this version. It works well even for process trees with some practical limitations described in the next chapter. A high level diagram of my system architecture is shown in Fig.2.

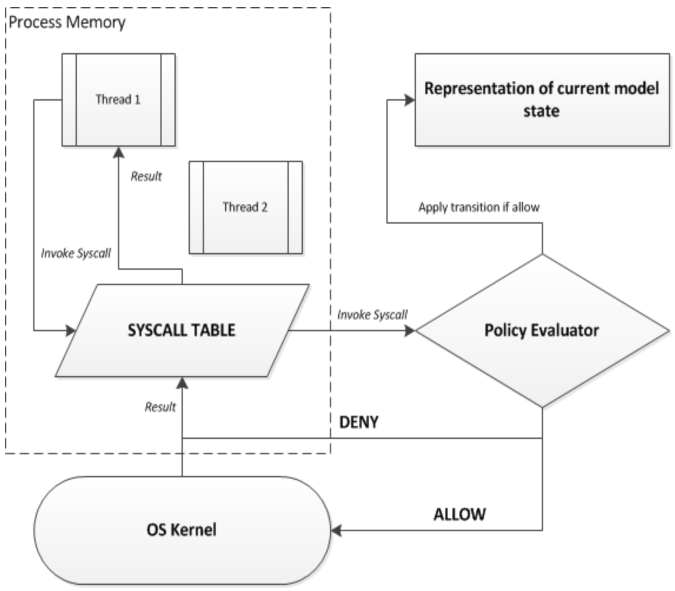

Figure 2: Proof-of-concept architecture diagram.

4. PRACTICAL LIMITATIONS

There are applications that rely on configuration files and need to have write access to them all the time. This is something we can not allow otherwise this file would soon 'poison' all files opened by these applications as containing sesitive data. There are two possible ways of solving this problem without breaking the applications functionality:

1) Denying writes to configuration files while passing a success return values to applications.
2) Coding exceptions into the system to ignore transactions reflecting system calls targeting the configuration files.

Both are far from ideal. First case could result in faulty application behavior. Second would introduce a potentially serious security hole. Even if we trust the application itself, it could be exploited by malicous users with knowledge of exceptions in the system [4].

Another practical limitation is that automatic policies (described in previous chapters) need to be generated and enforced. The side effect is that users are limited to opening only files containing the same type of sensitive information for writing at a time per application instance. While it does not present us with a security flaw or dysfunctions, it is an annoyance for users and might objectively affects their productivity.

Direct support from target operating systems will be required to tackle these and more upcoming challenges to successfully implement and deploy a production usable solution based on the proposed approach.

5. CONCLUSION

In this paper, I have introduced a complementary approach to information security on the operating system level. Because current state-of-the-art software products following its philosofy are using ad-hoc methodologies to achieve their goals, I have proposed a state-based data flow model to be able to formally define, verify and enforce security policies based on sensitive information proliferation. To prove my point, I have created a test implementation and evaluated practical limitations as challenges for further improvement and integration into operating system security mechanisms.

It is clear that the proposed system is never going to provide an absolute protection of confidentiality of sensitive information without support from target operating systems, but it does add a new layer of security against social engineering and account hijacking attacks. Due to practical limitation described in this paper, it currently needs to rely on trusted applications being utilized to access sensitive information. It is important to understand that this new approach is not there to replace time tested ones, but to complement them.

REFERENCES

[1] Clifton Phua, Protecting organisations from personal data breaches, Computer Fraud & Security, Volume 2009, Issue 1, January 2009, Pages 13-18, ISSN 1361-3723, 10.1016/S1361-3723(09)70011-9.

[2] Ouellet, E., and Proctor, P.E., Magic Quadrant for Content-Aware Data Loss Prevention, Technical Report, RA4 06242010, Gartner RAS Core Research, 2012

[3] M. Harvan, State-based Usage Control Enforcement with Data Flow Tracking using System Call Interposition, Network and System Security, 2009, ISBN 978-1-4244-5087-9 p.373-38

[4] Garfinkel, Tal. "Traps and pitfalls: Practical problems in system call interposition based security tools." Proceedings of the Network and Distributed Systems Security Symposium. Vol. 33. 2003.