AFRL-RY-WP-TR-2019-0135

MITOS: OPTIMAL DECISIONING FOR THE INDIRECT FLOW PROPAGATION DILEMMA IN DYNAMIC INFORMATION FLOW TRACKING SYSTEMS

Nicholas Sapountzis and Daniela Oliveira

University of Florida

DECEMBER 2019
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the gene public, including foreign nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2019-0135 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

TOD J. REINHART
Program Manager
Resilient and Agile Avionics Branch
Spectrum Warfare Division

DAVID G. HAGSTROM, DR-IV
Chief, Resilient and Agile Avionics Branch
Spectrum Warfare Division

JOHN F. CARR, DR-04
Chief, Spectrum Warfare Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//Signature//" stamped or typed above the signature

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
December 2019	Final	30 June 2015 – 1 July 2019

4. TITLE AND SUBTITLE
MITOS: OPTIMAL DECISIONING FOR THE INDIRECT FLOW PROPAGATION DILEMMA IN DYNAMIC INFORMATION FLOW TRACKING SYSTEMS

5a. CONTRACT NUMBER
FA8650-15-C-7565

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
61101E

6. AUTHOR(S)
Nicholas Sapountzis and Daniela Oliveira

5d. PROJECT NUMBER
1000

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
Y1B3

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Florida
601 Gale Lemerand Dr
Gainesville, Florida, 32603

8. PERFORMING ORGANIZATION REPORT NUMBER
AFRL-RY-WP-TR-2019-0135

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

Defense Advanced
Research Projects Agency
(DARPA/I2O)
675 North Randolph St.
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYWA

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2019-0135

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. Report contains color.

14. ABSTRACT
Dynamic Information Flow Tracking (DIFT) is a technique for tracking the information as it flows through a program's execution. Specifically, some inputs or data get tainted and then these taint marks (tags) propagate usually at the instruction-level. While DIFT has been a fundamental concept in computer and network security for the past decade, it still faces open challenges that impede its widespread application in practice; one of them being the indirect flow propagation dilemma: should the tags involved in an indirect flow, e.g., in a control or address dependency, be propagated? Propagating all these tags, as is done for direct flows, leads to overtainting, while not propagating those leads to undertainting. In this work, we analytically model that decisioning problem for indirect flows, by optimally weighting various tradeoffs including undertainting versus overtainting. Towards tackling this problem, we design and implement MITOS, a distributed-optimization algorithm that optimally decides about the propagation of indirect flows. We also perform a case-study scenario with a real in-memory only attack and show that MITOS improves simultaneously (i) system's spatiotemporal overhead and (ii) system's fingerprint on suspected bytes (up to 167%) compared to traditional DIFT, even though these metrics usually conflict.

15. SUBJECT TERMS
dynamic information flow tracking, distributed optimization, direct flows, indirect flows, in memory attacks

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Tod Reinhart
Unclassified	Unclassified	Unclassified	SAR	32	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

LIST OF EQUATIONS

Equation **Page**

1.0 SUMMARY

Dynamic Information Flow Tracking (DIFT), also called Dynamic Taint Analysis (DTA), is a technique for tracking the information as it flows through a program's execution. Specifically, some inputs or data get tainted and then these taint marks (tags) propagate usually at the instruction-level. While DIFT has been a fundamental concept in computer and network security for the past decade, it still faces open challenges that impede its widespread application in practice; one of them being the indirect flow propagation dilemma: *should the tags involved in an indirect flow, e.g., in a control or address dependency, be propagated?*

Propagating all these tags, as is done for direct flows, leads to *overtainting* (all taintable objects become tainted), while not propagating them leads to *undertainting* (information flow becomes incomplete). In this report, we analytically model that decisioning problem for indirect flows, by optimally weighting various tradeoffs including *undertainting* versus *overtainting*, importance of heterogeneous code semantics and context, and we show that the complete problem is NP-hard.

Towards tackling this problem, we design MITOS, a distributed-optimization algorithm, that: optimally decides about the propagation of indirect flows, is of low-complexity, is scalable, is able to flexibly adapt to different application scenarios and different security needs and converges to an optimal point. Additionally, MITOS is applicable to most DIFT systems that consider an arbitrary number of tag types, and introduces the key properties of fairness and tag-balancing to the DIFT field.

To demonstrate MITOS's applicability in practice, we implement and evaluate MITOS on top of an open-source DIFT, and we shed light on the open problem. We also perform a case-study scenario with a real in-memory only attack and show that MITOS improves simultaneously (i) system's spatiotemporal overhead (up to 40%), and (ii) system's fingerprint on suspected bytes (up to 167%) compared to traditional DIFT, even though these metrics usually conflict.

2.0 INTRODUCTION

Dynamic Information Flow Tracking (DIFT), or Dynamic Taint Analysis (DTA), systems operate by tainting various inputs or data of interest with some metadata (called tags) and keeping track of these tags during program or system execution. DIFT systems operate dynamically without requiring the availability of the source code, which makes them appealing for various types of applications, including enforcement of security policies, forensics analysis, and re-verse engineering. Prior work has attempted to leverage DIFT mainly for privacy and security purposes. For example, some early DIFT works (H. Yin, 2007) (J. Newsome and D. Song, 2005) (J. R. Crandall, 2004) (G. E. Suh, 2004)(J. R. Crandall, A security assessment of the minos architecture, 2005) attempted to detect different types of malware by following the information flow. Recently, DIFT has been leveraged to address different privacy and security vulnerabilities not only for modern operating systems (OSes), commodity software and honeypot technologies (A. M. Espinoza, 2016)(M. N. Arefi, 2018) but also for various IoT platforms (I. Bastys, 2018) and mobile devices (B. Gu, 2013).

Nevertheless, DIFT systems still face open challenges that impede their widespread application in practice. One of these challenges is the *dilemma of indirect flow dependency propagation.* An indirect flow occurs when information dependent on the program input determines from where and to where information flows. For example, in the code <a=b+ 1>, there is a direct flow from <b> to <a>, and all DIFT systems would propagate the tag of <b> to <a>. However, in the code<a= 0 ; if (b== 1) {a= 1}; >, the value of <a> is dependent on <b>, meaning that there is an indirect flow from <b> to <a>. Not propagating tags in these cases can lead to undertainting, where key important information flows are missed. Propagating tags for all indirect flow dependencies leads to overtainting, where most of the taintable objects in the system (e.g., bytes) become tainted with little useful information being acquired. While previous works have proposed some heuristics to tackle the problem, they usually make unrealistic assumptions to modern systems and have several limitations. For example, Panorama (H. Yin, 2007) relies on a human to manually label which indirect flows should be propagated. DTA++ (M. G. Kang, 2011) DTA++ or DYTAN (J. Clause, 2017) rely on offline analysis requiring multiple traces, which does not scale well. RIFLE (N. Vachharajani, 2004) and GLIFT are based on static analysis, and other works have prohibitive performance overheads (A. M. Espinoza, 2016) (M. N. Arefi, 2018). While useful, these techniques can only partially combat the problem.

Another, not well-studied, tradeoff in modern DIFT, is the one between semantics and applicability. Most of the DIFT systems ignore semantics, in order to be able to be applied to machine code or to be scaled to whole live systems, including all processes and the kernel. For example, it is difficult to properly keep track of the flow of different semantics even after they get inserted into the system, as they usually have heterogeneous properties, different propagation speeds, and contribute differently to the execution context. Further, ignoring them or adapting an one-size-fits-all handling may improve the DIFT applicability, but it usually misses some important knowledge about the information flow, putting a heavy toll on the DIFT performance and the detection efficiency for attacks.

In this work, we propose MITOS, an analytical framework that tackles the open problem of: *when an indirect flow should be propagated* in an optimal (providing the best solution) and efficient (of low-complexity, scalable, flexible and easily implementable) manner. In other words, MITOS theoretically addresses and tackles the open problem of indirect flow propagation encountered in practical DIFT systems, by unifying the two, usually conflicting, worlds of theory and practice. To the best of our knowledge, this is the first work in that direction, namely to analytically study this practical problem that remains open since the past decade. Specifically, the contributions of our work are:

(1) We theoretically model the open problem of optimal decisioning for indirect flow dependencies, optimally weighting various tradeoffs encountered in practical DIFT systems such as the undertaining vs. overtainting, importance of heterogeneous code semantics and context, and we show that the complete problem is NP-hard.
(2) We relax the problem and by leveraging distributed optimization we propose an algorithm that: optimally decides about the propagation of the indirect flows, is of low-complexity, is scalable, is able to flexibly adapt to different security or privacy scenarios, is applicable to most DIFT systems and converges to an optimal point.
(3) To the best of our knowledge, we are the first to introduce the fairness and tag balancing properties to the DIFT field, which control the balancing among the propagations of different tags or/and tag types. It matches information-theoretic intuitions about how tags should be propagated: e.g., flipping a coin that has 50%−50%chance of heads-tails carries more information than a coin that is biased in one direction. Similarly, when tag propagation becomes unbalanced towards one tag (e.g., due to the considered semantics or run dynamics), every object is tagged, and we show that little information is gained.
(4) To assess MITOS potential in real DIFT systems, we implemented and evaluated MITOS on top of FAROS, an existing open-source DIFT system (M. N. Arefi, 2018). We investigated the complex tradeoffs involved in the indirect flow dilemma e.g., the impact of undertainting vs. overtainting weight on the Pareto optimal distribution. Also, we performed a case-study scenario with a real in-memory attack and showed that MITOS improved simultaneously(i) system's time and memory overhead (up to 40%), and (ii) system's fingerprint on suspected bytes (up to167%) compared to standard DIFT, even though these metrics usually conflict.

Note: MITOS, in Greek mythology, was a ball of thread, that Ariadne gave to Theseus to help him escape the labyrinth of Minos kingdom. As MITOS helped Theseus to reversely find his way back to labyrinth's entrance by minimizing his wandering, our framework minimizes the incoherent tag propagations (e.g. of indirect flows), helping to illuminate the information flow from a certain output all the way back to the input.

3.0 METHODS, ASSUMPTIONS AND PROCEDURES

3.1 DIFT Methods and State of the Art

Dynamic Information Flow Tracking (DIFT), or Dynamic Taint Analysis (DTA), a fundamental concept in computer and network security, is a promising method to make systems transparent and to enable a wide variety of applications, such as enforcement of security policies, real-time forensics analysis, and reverse engineering.

The main idea is based to tag certain inputs or data (tag insertion), and then, propagating these tags as the program or system runs (tag propagation) with the goal of illuminating the flow of information.

Tag insertion is usually straight-forward, as the bytes being involved in certain system activities get tagged with some metadata. For example, in MINOS (J. R. Crandall, A security assessment of the minos architecture, 2005), an early DIFT system, all data coming from network were tagged with an extra bit indicating if the byte was suspicious.

There are two types of tag propagation flows: direct and indirect. Direct flow propagations (DFP) come from copy and computation dependencies. In a copy dependency, a value is copied from one location (e.g., from a byte, word of memory, CPU register) to another. To track this information flow, DIFT systems propagate the tag from the source to the destination. In computation dependencies, tags must be combined, e.g., after the computation of a sum between two variables, the tag of the result should contain both tags of variables. Indirect flow propagations (IFP) occur when information dependent on program input determines from where and to where information flows. There are two types of indirect flows: address and control dependencies. The follow example is an example of an address dependency:

```
char InputString = "This string is tainted";
char OutputString [128];
for (i = 0; i < strlen(InputString); i++)
OutputString[i]= lookuptable[InputString[i]];
```

We note an example in C that converts an array of tainted input from one format to another using a lookup table. There, as the string *InputString* is tainted, the string *OutputString* should also be tainted, since they carry the same information. To ensure that *OutputString* is properly tainted we check the taintedness of the address used for the load with *LookupTable* as its base, and propagate this taint. This example appears in special handling of ASCII control characters to ASN.1 encodings. Generally, indirect flows are expected to be the rule rather than the exception in modern systems, occurring in operations such as in compression/ decompression, encryption/decryption, hashing, switch statements, string manipulations. Indirect flows can create blindspots for practical DIFT analysis or vulnerabilities in security applications e.g., Trojans embedded in PDF documents or attacks that use encryption mechanisms are common, but cannot be tracked without tracking indirect flows.

Similarly, a control dependency, illustrated in the following code:

```
char InputTainted;
char OutputUntainted = 0;
for (bit = 1; bit < 256; bit <<= 1){
  if (bit & InputTainted) { OutputUntainted |= bit; }
```

There, *InputTainted* is copied to *OutputUntainted* bit by bit. Information flows one bit at a time through the control dependency in the if statement. Indirect flows can create blindspots for practical DIFT analysis or vulnerabilities in security applications e.g., Trojans embedded in PDF documents or attacks that use encryption mechanisms are common, but cannot be tracked without tracking indirect flows

Propagating all indirect flows can lead to overtainting, where most of the objects become tainted with little being be learned about the information flow. Conversely, not propagating indirect flows can lead to undertainting, where important knowledge about the information flow might be lost, which can be crucial for security applications to detect attack or violation of security policies. *While many works have attempted to tackle this dilemma, it still remains open, and constitutes one of the major impedances to the widespread usage of DIFT. The focus of this work is to model at hand and optimally tackle this problem.*

3.2 Assumptions

Tag differentiation. First, MITOS assumes that the DIFT system will leverage an arbitrary number of tag types. For example, it could include network tags (representing bytes coming from network), file tags (representing bytes coming from a file), process tags (representing bytes coming from the address space of a process). For the sake of presentation, we denote the different tag types as: t1, t2, t3, etc. Tag differentiation is a promising feature of modern DIFT systems since it captures the information flow from different perspectives. Note that, depending on the DIFT system and the security or privacy application needs, MITOS is open to the consideration of any type of code semantics as soon as they get captured with different tags or tag combinations (e.g., different data types or pointer tags).

Provenance list. MITOS assumes that for each byte in the main memory, register bank and Ethernet card memory, a provenance list of tags accumulated during the system execution. MITOS also assumes that different tag types will have different formats and sizes depending on the type of information they represent; i.e., network, file, process, string, pointer tags. The provenance list, through the set of tags it stores, keeps all information flow history for the life cycle of a byte in the system. For example, the following figure illustrates the provenance list for the byte representing memory address #7FFFFF8. This byte came from a network source (IP=10.245.44.43), was read as part of the address space of a process (PID 3543), was written into a file (with file ID 14) and then was read as part of an address space of another process (PID 2912).

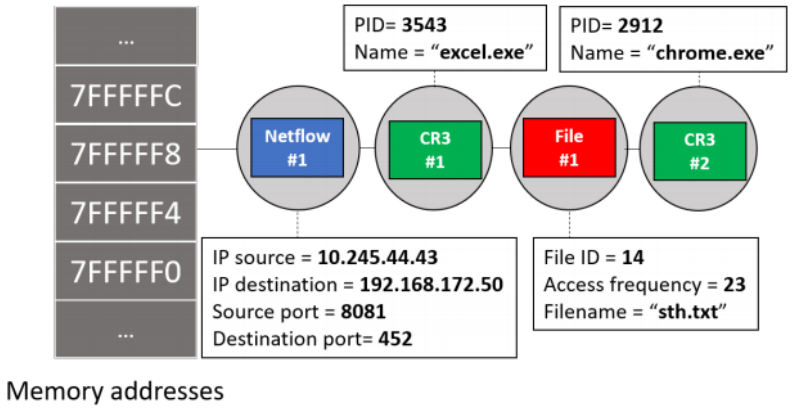


Figure 1. Provenance list of a byte

Provenance list size. The provenance list size is finite, denoted as M_{prov}. For example, M_{prov} = 10 means that each byte can keep up to 10 different tags in its provenance list.

Shadow Memory. MITOS assumes that the provenance list of tags for each byte will be stored in a shadow memory, whose implementation depends on the DIFT system, e.g., hashmap or duplicated memory.

3.3 Problem Definition

We first (A) define the variables that are associated with the indirect flow dilemma corresponding to our control variables. Then, (B) we design a new cost function that attempts to optimally weight the different tradeoffs involved at the indirect flow propagation. Finally, (C) we define our optimization problem.

(A. Control Variables: n). When the DIFT system is confronted with an indirect flow dependency, it needs to decide whether it is worth propagating that particular tag to one additional byte. For the sake of presentation, let us assume that each particular tag has a unique ID t,i; where t (t=t1, or t=t2, etc.) indicates the tag type, and i (i=1 or i=2, etc.) represents an integer that differentiates tags of same type. We now define $n_{t,i}$ to be the number of bytes whose provenance lists contain the tag with ID t,i. Throughout this work, we will often refer to $n_{t,i}$ as the number of copies of the tag with ID t,i. The vector **n** looks like:

$$\mathbf{n}=\begin{bmatrix} n\_\{1,1\} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \vdots & \cdots & \ddots \end{bmatrix}$$

The number of dimensions of **n** changes dynamically as the system runs, since new tags are created/deleted due to the continuous creation/ termination of processes, network connections, etc. This prevents the usage of standard optimization techniques, further complicating the problem

(B. Cost function: c (n)). We now define our cost function c (n) that dynamically weights the cost of α -fair undertainting and the cost of β-steep overtainting.

$$\boldsymbol{c}\,(\boldsymbol{n}) = \boldsymbol{c}\,{}_{\boldsymbol{a}}^{\boldsymbol{under}}(\boldsymbol{n}) + \boldsymbol{\tau} * \boldsymbol{c}\,{}_{\boldsymbol{\beta}}^{\boldsymbol{over}}(\boldsymbol{n}) \tag{1}$$

where $c\,{}_{a}^{under}(n)$ is the cost of undertainting, and $c\,{}_{\beta}^{over}(n)$ is the cost of overtainting.

In the next two paragraphs (see B.1 and B.2) we elaborate on these two costs. τ, that is a positive constant is input and dynamically weights the tradeoff between over- and under- tainting. When τ = 0 the cost of overtainting disappears and, thus, the undertainting cost dominates, and all tags are propagated. As we increase τ the emphasis moves towards the overtainting, which limits tag propagation. This weighting parameter is often used in multicriterion optimization problems. We are interested in finding Pareto efficient operating points.

A solution n* is Pareto efficient if for any other feasible n, it holds:

$$\boldsymbol{c}\,{}_{\boldsymbol{a}}^{\boldsymbol{under}}(\boldsymbol{n}) + \boldsymbol{\tau} * \boldsymbol{c}\,{}_{\boldsymbol{\beta}}^{\boldsymbol{over}}(\boldsymbol{n}) < \boldsymbol{c}\,{}_{\boldsymbol{a}}^{\boldsymbol{under}}(\boldsymbol{n}*) + \boldsymbol{\tau} * \boldsymbol{c}\,{}_{\boldsymbol{\beta}}^{\boldsymbol{over}}(\boldsymbol{n}*) \tag{2}$$
$$=> \boldsymbol{n} = \boldsymbol{n}*$$

The above relation suggests that any other solution could improve the undertainting or the overtainting, but not both. All Pareto efficient points can be found by scalarization, e.g. minimizing our considered cost function for different values of τ.

(B1. Cost function of undertainting.) Now, we model the undertainting cost function. We introduce the fairness parameter $\alpha \in R+$ that is input and balances the number of propagations for different tags, and the parameter $u_t \in R+$ that weights the importance of different tag types.

$$\boldsymbol{c}\,{}_{\boldsymbol{a}}^{\boldsymbol{under}}(\boldsymbol{n}) = \sum_{\boldsymbol{t}} \boldsymbol{u_t} \sum_{\boldsymbol{i}} \frac{\boldsymbol{n_{t,i}^{1-\alpha}}}{\boldsymbol{\alpha-1}} \tag{3}$$

In the following, we discuss the properties of our considered cost function. Figure 2 Considered cost functions for undertainting and overtainting. depicts this function for different values of α. When α = 1, the above function is not defined, and log is used instead. Note that, the proposed α-fair fairness function was inspired by the fairness in resource allocation for wireless networks (N.

Sapountzis, 2018). It is monotonically decreasing on $n_{t,i.}$ This means that the more the copies of a tag, the lower the undertainting cost for that tag. Thus, the slope of undertainting cost is continuously decreasing, meaning that it has negative gradient.

As $\alpha \rightarrow \infty$ tag-balancing is achieved through max-min fairness. As we increase the slope becomes more and more steep. $\alpha \rightarrow \infty$ the slope maximizes and thus our function attempts to maximize the propagation of tags with fewer copies, i.e. max-min fairness. The latter maximizes the entropy of the system from an information-theory perspective. This fairness has interesting implications for DIFT systems.

For example, assume that a system service reads input from the network that contains the number of bytes that a remote machine will send as an integer, followed by the actual bytes of data. Suppose the service first reads the integer from the network and then allocates enough space on the heap for all the data based on that integer. The entire placement of data on the heap has now been influenced by tagged input. If address dependencies are propagated too aggressively, then for the rest of the lifetime of that process all information stored on the heap will become tainted with the tag that had been associated with that integer (a netflow tag). Other examples include, the scenario where a stack pointer is tainted by variable-sized arrays on the stack, or the stack pointer being popped or set from a register while the program counter happens to be tagged. Then, everything on the stack becomes tainted and starts overtainting all taintable objects in the system because the stack is heavily accessed. Slowinska and Bos provide more examples with different semantics in that direction (Bos, 2009). In all such scenarios, MITOS will adjust the tag propagations that attempt to hurt the system entropy.

Tag-balancing alone may not be sufficient for a good propagation decision. Different tag types carry heterogeneous information (e.g., network, pointer, file) and potentially propagate differently in the system. This calls for schemes that are able to weight the propagation speed for different tag types, based on e.g. the application, the system workload, or the security policies implemented. Our cost function flexibly overcomes this obstacle by using $u_t \geq 0$ which weights the importance of different tag types and can boost or decelerate their propagation respectively. We define u to be the vector weighting the different tag types: $u = [u_{t1} ; u_{t2} ; \ldots]$.

One could even consider a tag confluence (when two or more tags come together to the same provenance list) to control the tag propagation of the involved tags based on a certain run context. For example, one could weight differently the undertainting cost when tag types tn; tm confluence together as that confluence might highlight a certain context scenario of interest. In our results we consider a study case scenario where two particular tag types confluence together illuminating an inmemory only attack. These considerations lead to a weighted fair optimization.

(B2. Cost function of overtainting). If R is the memory capacity of the system in bytes (e.g., main memory, register bank, Ethernet card memory) and M_{prov} is the maximum size of the provenance list, then the total tag space in the provenance lists is $N_R = R * M_{prov}$. For example, if R = 4GB and for each byte we keep a list up to 10 elements, there are in total $N_R = 40 * 10^9$ provenance list elements.

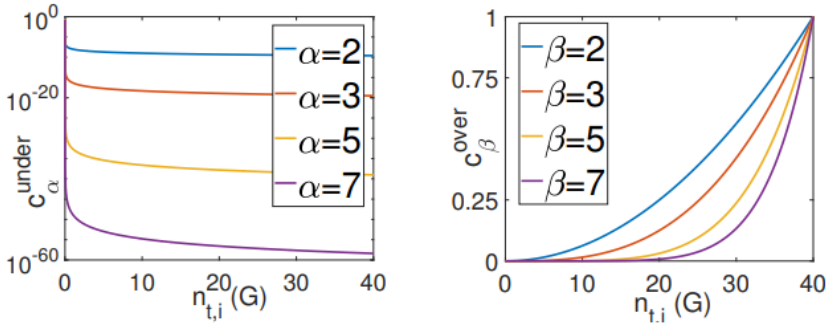
We introduce the parameter β that is an input and dictates the slope, namely steepness, on the overtainting cost. Then,

$$c_{\beta}^{over}(n) = \left(\frac{\sum_t \sum_i n_{t,i}}{N_R}\right)^{\beta} \quad (4)$$

In the following, we discuss the properties of our considered cost function. Figure 2 Considered cost functions for undertainting and overtainting depicts the function of overtainting It is monotonically increasing on $n_{t,i.}$, i.e. the larger the number of tags in the system, the higher the cost. Thus, its slope is continuously increasing with positive gradient. Following the standard penalty functions, it should have at least quadratic penalty on the memory pollution, thus we keep $\beta >= 2$, ensuring also that it is twice differentiable.

As β increases the cost of overtainting gets steeper. Similarly, to the undertainting cost, different tag types may impact memory pollution differently. Our cost function flexibly takes memory pollution into account by using o = [ot1; ot2; …] that weights the partial pollution of different tag types and adapts their impact on the total pollution.

In the following figure we plot the undertainting cost and overtainting cost for different values of α and β.



(a) undertainting cost c_{α}^{under}. (b) overtainting cost c_{β}^{over}.

Figure 2. Considered cost functions for undertainting and overtainting

(C. Optimization Problem). Based on the defined control variables and cost function we formulate our problem.

Problem 1: The indirect flow propagation problem at hand is:

$$\min_{n} \quad c_{\alpha}^{under}(n) + c_{\beta}^{over}(n) \quad (5)$$

We now explain the physical meaning of our optimization problem. Our control variable is the vector n that determines the decision of propagating the tags coming from indirect flows. Specifically, if a tag in such a scenario is worth propagating (i.e., it improves the information flow), e.g. due to the semantics or context priorities, we propagate it and increase the corresponding value of n, otherwise we do not (see A. Control Variables). This will become more clear in the next subsection where we describe how n should be best derived: n is updated, by leveraging the marginal costs dictated by the considered cost function (see B. Cost Function) and the considered linear constraints, every time our system encounters an indirect flow leveraging distributed optimization techniques. The marginal cost will optimally take into consideration all the tradeoffs discussed above.

This problem has two main challenges at hand.

- First, the control variables $n_{t,i}$ of the considered vector n takes integer values, i.e. $n_{t,i}$ = 1,2,3… since a tag can be propagated to one or several bytes. Thus, this is a NP-hard integer optimization problem, which is hard to solve optimally.
- Second, the number of control variables $n_{t,i}$ can experience sharp increases and decreases in very short intervals (e.g., a video game reads data from files and downloads content from Internet, thus generates hundreds of file and network tags in a few milliseconds), thus continuously changing the dimensions of n. This further complicates the problem since the system dynamics and the optimal points change continuously as the system runs.

3.4 Optimal Solution to the Problem

We now tackle Problem 1.

We start by discussing how we are going address the two major challenges discussed earlier: the problem is NP hard and the number of dimensions of the control variable change continuously. Similarly to various works, we first propose to consider the continuous relaxation of the problem to obtain a closed form real-valued solution. Specifically, we relax the allowed values for $n_{t,i}$ such that $n_{t,i} \epsilon$ R+ (it takes all real values). This relaxation brings two fundamental advantages: first, it is possible to evaluate the quality of a feasible set of solutions; second, it is much faster to optimize than the original integer problem.

Lemma 1. The relaxation of $n_{t,i}$ from $n_{t,i}$ N+ to R+ in Problem 1 transforms it in a convex optimization problem.

Proof. To show that the relaxed problem is a convex optimization problem we need to show that the objective function and the constraints are convex functions on the control variables. Our objective function is the weighted sum of the undertainting and over-tainting cost cover (n). The cost function of undertainting is convex for $n_{t,i \epsilon}$ R+, as its second derivative is positive. Note that, the sum of convex functions is also convex, and, thus, their sum is also convex. Also, the

cost function of overtainting is a sum of exponential functions that are all convex, thus cover (n) is also convex. To sum up, our relaxed objective function is convex as it is the weighted sum of two convex functions.

This relaxed problem can be solved analytically using the method of Lagrange multipliers and Karush Kuhn Tucker (KKT) conditions, to derive the optimal vector. 2 However, such a solution would require a centralized implementation that would need to (i) gather all the necessary information from all tags, and (ii) re-calculate the global optimal point every time a new tag is inserted/deleted at the system. Note that, this solution might not scale well, as new tags are created and deleted very frequently as the system runs, and, thus, the overhead in re-calculating the new optimal points can be prohibitive.

Thus, we propose a distributed solution that scales well, namely, it dynamically adapts to the rapid tag creations/deletions and the system changes and show that it converges to an optimal point in the long term. Solution roadmap: In the following, we start with Indirect Flow Propagation (IFP) Scenario 1, where we assume that the source operand of the indirect flow attempts to propagate a single tag and the destination has at least one available space in its provenance list to accommodate it. Then, we generalize it to IFP Scenario 2 and we assume that the destination has limited available space in its provenance list, and cannot accommodate all tags scheduled for propagation.

IFP Scenario 1: **Single tag propagation with sufficient space at the destination provenance list.** Assume an indirect flow scenario in a particular instruction where (i) the source operand has only one tag for potential propagation. Also, (ii) the destination has (at least) one available space in its provenance list, e.g., see the figure below. The store word instruction illustrated, copies data from a register to memory. In our example, it attempts to store a word from register t0 to the memory location corresponding to 7FFFFF0 + t3 = 7FFFFF0 + 8 = 7FFFFF8, given that the value of t3 = 8. This is an address dependency, since the value of t3 will dictate the memory address location that the data of register t0 will be stored, and further the system execution. Note that, there is a direct flow too from t0 to 7FFFFF8 that will be propagated following the basic DIFT rules and is out of the scope of this work. Our objective is to answer the following question: should the DIFT system propagate the red tag C?

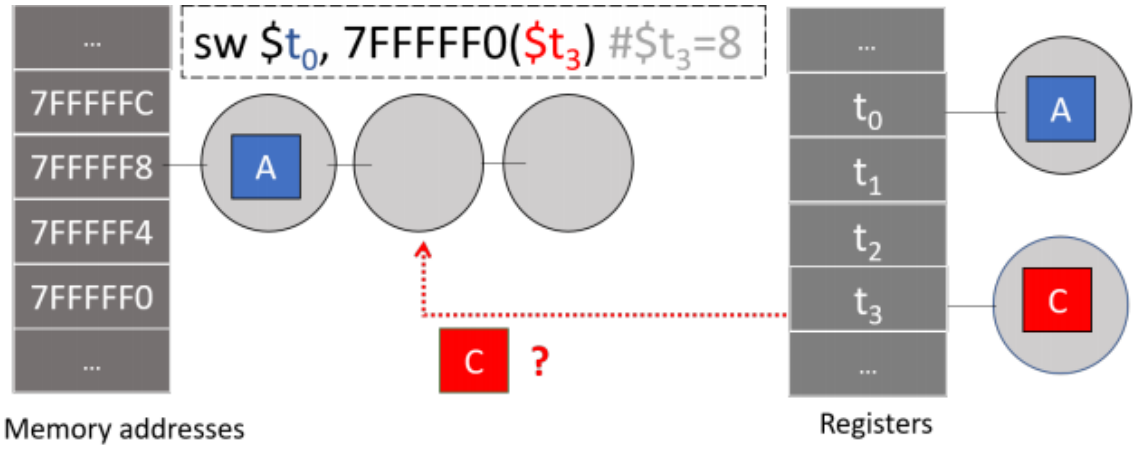


Figure 3. Address dependency example: store word sw instruction with sufficient space

Algorithm 1. IFP Scenario 1: Propagation of the tag with ID with sufficient space at the destination' provenance list.
1: Step1: Derive the direction of the gradient towards nT;I .
2: Step2: Use the gradient-descent crit. for IFP decisioning.
3: If marginal cost < 0 then propagate the tag.
4: Else, block the tag.

Alg. 1 shows our proposed method towards answering this question. The main idea is to take the indirect flow propagation decision based on the first-order optimization criterion. In the following two paragraphs we elaborate on the two-steps of Alg. 1.

First, we derive the direction of the gradient towards the dimension we are interested in. In our case, this dimension refers to the control variable that is associated with the tag considered for indirect flow propagation. For the sake of presentation, we assume that the type of the tag considered for propagation (e.g., the red tag in Fig. 5) is T, with identification number I, i.e. the involved control variable $n_{T,I}$ (we use capital letters T; I to refer to a particular tag). The partial derivative of the tag involved in an indirect flow with ID T,I , namely $n_{t,i}$, that will determine its propagation is:

$$\Delta n_{T,I} = \frac{\theta\, c(n)}{\theta\, n_{T,I}} = -u_T * n_{T,I}^{-\alpha} + \tau * \beta * \left(\frac{\sum_t o_t \sum_i n_{T,I}}{N_R}\right)^{\beta-1} \quad (6)$$

The variable $n_{T,I}$ refers to the cost added by propagating that tag with ID T,I to one more byte, and, therefore, can be seen as the marginal cost of the indirect flow propagation.

This marginal cost depends on: (i) the submarginal cost of undertainting that attempts to decrease it (left part of marginal cost) , and on (ii) the submarginal cost of overtainting that attempts to increase it (right part of marginal cost). Note that, while the former quantity differentiates for

different tags (it can be derived with local information, in practice), the latter quantity is the same for all tags (it is actually the memory pollution, kept in a globally available variable, in practice). The sign of their sum, and, thus, the direction of the gradient, follows the sign of the highest absolute value. Second, following the direction of the gradient, we attempt to improve our considered cost function. Since our function is convex and we attempt to minimize it, we need to follow the opposite direction of the considered gradient. More precisely, if the partial derivative of the marginal cost is negative, then the first-order optimization criterion suggests increasing the 6 involved control variable by +1 and thus to propagate the indirect flow. On the other hand, if the partial derivative is positive such a decision would hurt our objective and thus the DIFT system should not propagate the tag.

Lemma 2. *[Optimal decisioning for indirect flow propagation] The optimal rule for determining the propagation of a tag currently involved in an indirect flow, is:*

propagate it if: $n_{T,I} < 0$*; block it otherwise.*

IFP Scenario 2: **Multiple tag propagations with insufficient space at the destination provenance list**. We now generalize the above scenario. Assume an indirect flow scenario where (i) the source operand has multiple tags for potential propagation, and (ii) the destination operand does not have enough space in its provenance list to accommodate all the potential tags scheduled for propagation, as depicted in the figure below. In this store word instruction we see again an address dependency from the register t3 (source) to the same memory location 7FFFFF8 (destination). However, now the source has three potential tags for propagation and the destination only two available spaces in its provenance list. Our objective is to answer the following question: which, at maximum two, tags (out of the C, E, B) should the DIFT system propagate ?

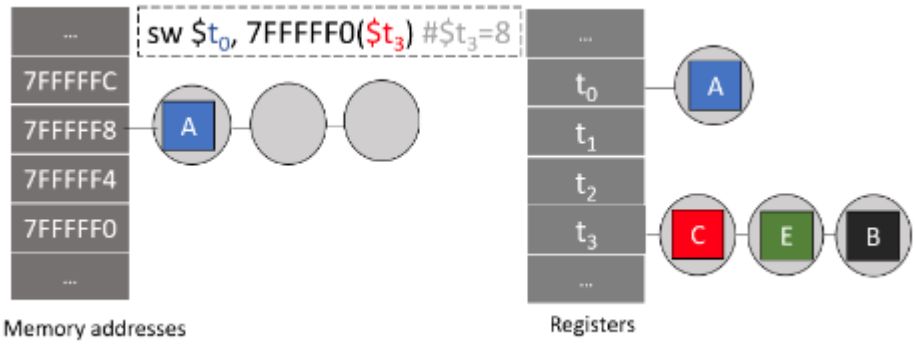


Figure 4. Address dependency example: store word sw instruction without sufficient space

This is a challenging question and considering all possible combinations would require exponential complexity. We extend Algorithm 1 to Algorithm 2, and use a prioritized first-order optimization criterion, by exploiting that the improvement of the cost function can be maximized if we follow the gradient towards the lowest submarginal costs.

Algorithm 2. IFP Scenario 2: Propagation of multiple tags with limited space available in the destinations' provenance list, namely A available tags space.
1: Derive the partial derivatives (marginal costs) for all involved tags in the current IFP.
2: Sort the tags wrt their partial derivatives increasingly.
3: Set j = 1. // tag being considered currently for propagation.
4: Set #props= 0. // number of successfully propagated tags.
5: while (#props _ A) and ($n_{Tj,Ij} < 0$)
6: Propagate tag j.
7: #props++. // increase by 1 the propagated tags.
8: j++ //move for the next tag.
9: Recalculate

First, we derive all the partial derivatives of all tags involved in the considered indirect flow at the source using Lemma 2. (line 1, Alg. 2). Then, we sort the partial derivatives in an increasing order, such that the first tag (j=1) has the lowest marginal cost (line 2, Alg. 2). 3 Then, we set (i) the first tag to be considered for propagation to the one with lowest marginal cost i.e. j=1, and (ii) the tags that have been successfully propagated to 0, #props = 0 (line 3-4, Alg. 2). The while loop that follows keeps propagating the tags while the available space in the destination provenance list is not exceeded (#props <= A) and while the marginal cost of the current tag is negative. More precisely, if the above two conditions hold true, we propagate the tag and increase by +1 the counter of propagations (line 6-7, Alg. 2). Then, we move the pointer to the next tag (line 8, Alg. 2) and recalculate the partial derivative of the next tag since the cost of overtainting might have changed (line 9, Alg. 2). Since the tags are ranked according to their marginal cost, and the propagation decision is based on them, during each indirect flow the improvement in c(n) is maximal among all feasible directions (a variable $n_{T,I}$ cannot change during an indirect flow propagation, if the tag T,I is not present in the list of the source); given the convexity of our cost function, this method is shown to correspond to a distributed implementation of a gradient decent algorithm.

We now discuss various properties of our proposed rule in Lemma 2 and of our generic Algorithm 2.

1) Our derived rule optimally decides about the propagation of all indirect flows encountered. It does so by optimally weighting the involved tradeoffs (e.g., undertainting versus overtainting, semantics and context priorities) and system dynamics (e.g., memory pollution).
2) Our rule for the IFP is of low-complexity. The time complexity is O(1), since every time MITOS needs to make an IFP decision it only needs to sum two real numbers. For the space complexity, we need (i) O(NR) space for the submarginal cost of undertainting, as our policy is byte level attributable. Also, we only require (ii) O(1) space for the overtainting cost, as we keep a single estimation of the memory pollution.
3) It is scalable. MITOS only needs to retrieve a local value about how undertainted the tag is, for the IFP decisioning. This keeps MITOS scalable as its complexity doesn't change on the number of tags in the system.

4) It is flexible, since by changing the input parameters one can flexibly weight the involved tradeoffs differently and capture different performance degrees based on the 7 application scenarios and security needs. It is also - fair, since captures different degrees of tag balancing.
5) Alg. 2 converges to an optimal point leveraging the descent direction of the first-order optimization criterion.

Generalization to direct flows and different objectives. Modern instruction-level DIFT systems usually incur an intolerable overhead due to also aggressive direct flow propagations. MITOS can be extended to optimize the propagation of direct flows through the considered objective. In that case, Alg. 2 should be invoked every time the system has to make either a direct or an indirect flow decision. Additionally, MITOS can be used to optimize different cost functions for propagation decisioning. One should just: (i) change the cost function by modeling the tradeoffs he wants, (ii) use the new partial derivative in the tag sorting (line 2, Alg. 2), and then check whether the partial derivative is negative to perform the tag propagation (line 5, Alg. 2). The nature of our proposed algorithm would stay the same, e.g., it will still be scalable, of low complexity, convergent, etc.

4.0 RESULTS AND DISCUSSIONS

To evaluate MITOS applicability in practice we implemented it in an existing, open-source DIFT system, FAROS (M. N. Arefi, 2018). We start by detailing our implementation, and then we evaluate MITOS' performance under various tradeoffs encountered in different indirect flow scenarios, such as undertainting vs. overtainting, tag type importance, and different fairness degrees in tag balancing. Then, we evaluate MITOS in a study case, where FAROS is detecting stealthy in-memory-only attacks. We show how the application of MITOS for all types of flows can not only improve FAROS' spatiotemporal performance, but also its detection accuracy, in terms of recognizing the bytes that are part of an exploit. In the following, if not explicitly mentioned, we assume $\alpha = 1.5$, $\beta = 2$, $u_t = o_t = 1$, $\tau = 1$, and that all τ values are normalized up to the power of 10^{-6}. Also, unless otherwise stated, we consider the address dependencies, as indirect flows, in this evaluation section, and we plan to consider the control flow dependencies in our future work. Finally, we varied the parameters in our evaluations and reached similar conclusions. Finally, our discussion concludes the section.

4.1. Applying MITOS to FAROS – an existing software-based DIFT

The following figure illustrates our architecture which consists of five layers: (i) the host Linux 14.04 machine, (ii) the QEMU virtual machine (VM) with the PANDA plugin, (iii) the opensource DIFT tool FAROS, (iv) MITOS implemented as a FAROS extension for the IFP problem, (v) Windows 7 as guest OS 4. FAROS was implemented for Windows 7. Note that, the type of OS does not affect the nature of the indirect flow propagation problem considered in this work, thus the OS choice is not expected to (directly) impact the insights offered by MITOS.

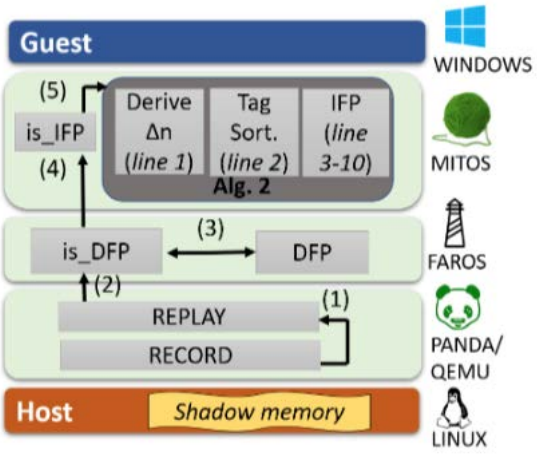


Figure 5. System Architecture

PANDA is built upon the QEMU whole-system emulator adding to it capabilities for instruction level analysis including recording and replaying a system run. FAROS was implemented as an PANDA plugin extension leveraging direct flow propagations for malware analysis. We now describe the implementation of MITOS along with its interaction with PANDA and FAROS. PANDA provides access to all instructions emulated in a previously recorded run (steps (1)-(2)).

Then, FAROS component *is_DFP* filters and processes the instructions that involve a direct flow propagation (DFP) (step (3)) and propagates all the DFPs by inspecting and modifying the shadow memory at the host. Then, FAROS invokes MITOS, to propagate any indirect flow propagations (IFPs). Next, the instructions associated with IFP are subject to Alg. 2 (step (5)). Specifically, MITOS inspects the shadow memory and calculates the marginal costs for all tags appeared in the source operand of the instruction, it sorts them and decides if they worth being propagated (see Alg. 2). Finally, MITOS updates the shadow memory of the propagated tags.

4.2. Sensitivity Analysis

Sensitivity analysis explores the impact of the inputs in a mathematical model (e.g., MITOS inputs include α) to the output (e.g., for MITOS we are interested in the indirect flow decision impact, memory pollution, overhead etc). We focus on three scenarios and investigate MITOS performance on the tradeoffs involved in the indirect flow decisioning.

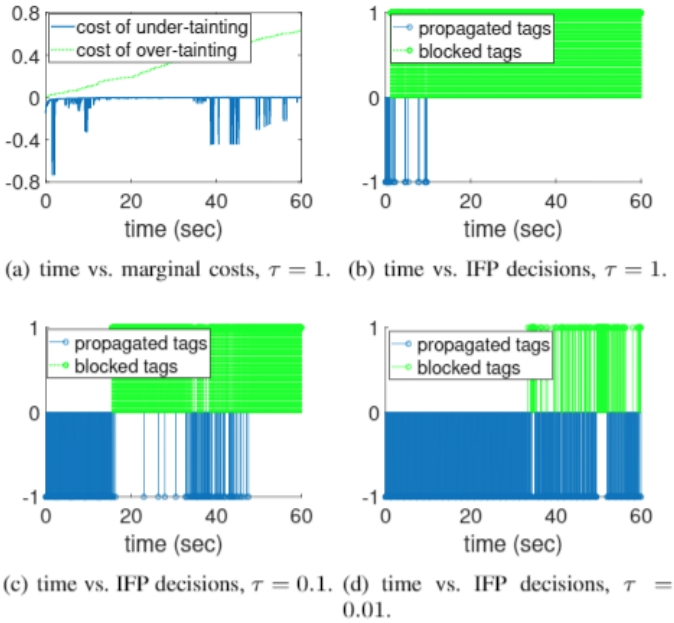
(Scenario 1: Network (1 minute record - 11 hours of replay)) We ran an one-minute network-benchmark on Windows using the PerformanceTest tool of Passmark (https://www.passmark.com/.), where the guest acts as a client and downloaded several megabytes. There, (i) the functions is_DFP and DFP are removed from FAROS, and (ii) both direct and indirect flows are forwarded to MITOS and further to Alg. 2. To do so, we replace the function is_IFP with is_DFP_or_IFP, i.e. MITOS now handles instructions related to both direct and indirect flows. 8 data from a remote server. We replayed this record multiple times with MITOS on top of FAROS using different values for our inputs. Below, we focus on the impact of our inputs on the tradeoffs involved on IFP decisioning.

Undertainting vs. overtainting. The parameter that weights this tradeoff is $\tau > R+$, where the higher the τ the more emphasis is put on the cost of overtainting. Figure 6 [x-axis] time vs. [y-axis] marginal costs and IFP decisions. shows how the system reacts for three different values of τ . We replayed the one-minute recordings three times, using different values of $\tau = 1$; $\tau=0.1$; $\tau=0.001$, keeping all other parameters fixed, and waited until the system converges to a point at the end of the replay (actually, the control vector n converges to a value). Figure 6 [x-axis] time vs. [y-axis] marginal costs and IFP decisions., in the (a) plot, shows the marginal costs of under- and overtainting (y-axis) for different indirect flow propagations that MITOS encountered as a function of time (x-axis).

For the sake of presentation, we have included the effect of τ at the cost of overtainting. The undertainting costs of different indirect flows varies; this cost is only dependent on the current number of copies of the considered tag. The overtainting cost is (mostly) monotonically increasing over time since the memory pollution is (mostly) increasing on time due to the new tag insertions/propagations. Figure 6 [x-axis] time vs. [y-axis] marginal costs and IFP decisions., in the (b) plot, shows the corresponding decisions for these indirect flows: if the cost of undertainting dominates the tag is propagated (and we plot +1 in the y-axis), otherwise the tag is blocked (and we plot -1). Since we keep a relatively high value of τ, most of the tags are blocked. In Figure 6 [x-axis] time vs. [y-axis] marginal costs and IFP decisions., in the (c) and (d) plots, we decrease τ,

which further decreases the emphasis of overtainting and plot the decisions of the indirect flows encountered. Indeed, more tags get propagated over time due to τ decrease.

Message: The undertainting vs. overtainting tradeoff should not be a challenge in the indirect flow dilemma and DIFT efficiency. Instead of applying simple heuristics that attempt to improve either side of that tradeoff, MITOS weights it with respect to τ, converging to a point that flexibly optimizes it.



(a) time vs. marginal costs, $\tau = 1$. (b) time vs. IFP decisions, $\tau = 1$.

(c) time vs. IFP decisions, $\tau = 0.1$. (d) time vs. IFP decisions, $\tau = 0.01$.

Figure 6. [x-axis] time vs. [y-axis] marginal costs and IFP decisions

Optimal distribution. We showed that for a fixed value of τ, MITOS converges to an optimal point. This point is an optimal point, as its found through the gradient descent criterion. Figure 7 Optimal Points for different values of τ. depicts all optimal points for different values of τ. Each point in the x-axis (different τ value) corresponds to a MITOS' run and the y-axis (c(n)) corresponds to the value of the cost function. Note that, this point is not necessarily a global optimal point.

Message: The optimal distribution is quite useful when one wants to investigate, or even predict, how a change in τ affects the cost function and performance.

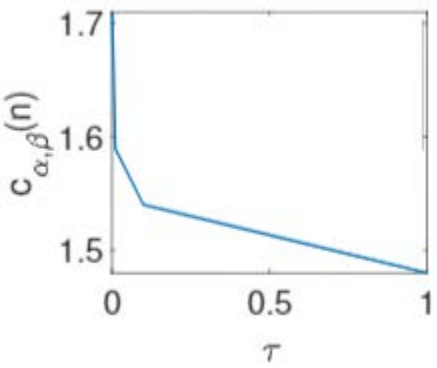


Figure 7. Optimal Points for different values of τ

Fairness - Tag balancing. The higher the α, the more fair MITOS is. Specifically, as α increases the undertainting cost becomes steeper, i.e. it penalizes more intensely the overpropagated tags. This attempts to maximize the tags with fewer copies. In other words, through a maxmin fairness MITOS can perform tag balancing based on the input. Figure 8 α vs. fairness (and tag balancing).corresponds to six different MITOS' runs for six different values of α. We measure the fairness degree, or taint-balancing efficiency, based on the mean square error difference between the number of copies of different tags. The sharp deviations of the tags can be alleviated by adapting α, thus improving tag balancing performance, and entropy, up to 2X. This is important as traditional DIFT systems tend to overpropagate tags in multiple scenarios, consequently hurting their overall performance and wasting memory resources from the provenance lists.

Message: MITOS input parameter flexibly captures different fairness degrees, in terms of tag-balancing. We envision this to have immense impact on modern DIFT systems, since these systems experience situations where they tend to overpropagate certain tags. MITOS is generic enough to capture all these cases and handle tag balancing an optimal way.

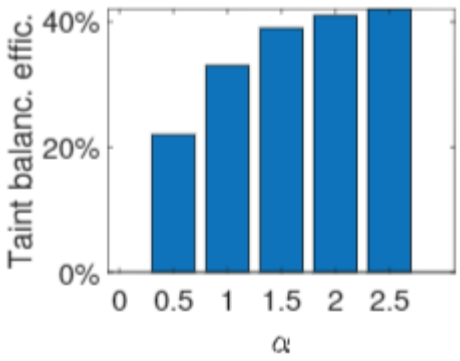


Figure 8. α vs. fairness (and tag balancing)

Tag type importance. We now focus on the tradeoff arising when tags of different tag types compete for propagation. Modern DIFT systems might include different tag types with different properties, importance, and propagation speeds. There are many reasons that warrant dynamic

and somewhat personalized strategies to determine the weight of over vs under tainting based on tag type. For example, for applications handling sensitive files one might want to put a higher cost on undertainting so that the probability of not tracking a file tag is low. MITOS takes this into account through the parameters u_t (determining the importance of a particular tag type), for all tag types t. In Figure 9 Netflow tag importance vs. portion of netflow tags that got propagated.we consider different values of $u_{netflow}$ (by keeping the remainder parameters fixed and equal to 1). For each value we plot in blue (netflow line in the legend) the percentage of netflow tags, encountered at the end of each replay, that is normalized by the maximum value taken when $u_{netflow} = 100$. Increasing $u_{netflow}$ monotonically boosts the netflow tag propagation speed. The boosting of certain tag type propagation speed impacts how MITOS handles other tag types, because a speed boosting means an increase in memory pollution. For example, as shown in Figure 9 Netflow tag importance cs. portion of netflow tags that got propagated. the number of propagation of CR3 (process) tags decrease as MITOS increases the important of netflow tags. Note that, since export table tags are, in general, more mildly propagated compared to the CR3 tags, the undertainting cost of the former is higher, and, thus, thus the propagation speed of export table tags is mildly decelerated. Results for file tags are not shown because there were no indirect flow propagations for them for this particular benchmark.

Message: MITOS introduces a flexible way to dynamically fine-tune the propagation speed of different tag types though the tag type importance, to accommodate the inherently heterogeneity of tag priorities for a particular system.

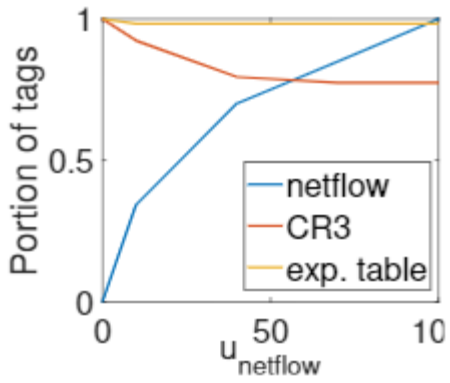


Figure 9. Netflow tag importance vs. portion of netflow tags that got propagated

(Scenario 2,3: CPU and file-system (1 minute record - 12 and 11 hours of replay, respectively)) We evaluated MITOS in two additional scenarios, leveraging (i) a CPU-benchmark using the PerformanceTest tool of Passmark, where the guest OS created, ruan and terminated various processes performing tasks related to compression, physics, and prime numbers and (ii) a file-benchmark, by manually creating, copying and pasting various files including other actions that would impact indirect flows e.g. search and replace. We could not use the file-benchmark of Passmark because it only uses basic file actions, which would not lead to indirect flows. For these two scenarios we found results similar to those detailed for Scenario 1 (Network). For completeness we plot the optimal distributions in Figure 10 Optimal points for different benchmark scenarios.. The File benchmark creates some indirect flows, with a few tags. Thus, increasing τ will not considerably affect the cost function after some point. The CPU benchmark is steeper compared to both the Network and File benchmarks, since CR3 tags are significantly more involved in indirect flows.

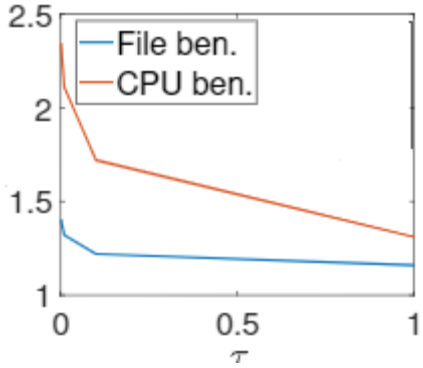


Figure 10. Optimal points for different benchmark scenarios

4.3. Case study: Flagging In-Memory-Only Attacks

We now apply MITOS in FAROS while it is flagging stealthy in-memory attacks and show the substantial improvement MITOS brings in spatiotemporal performance and detection efficiency. In particular, we study: how much time MITOS/FAROS needs to replay such an attack compared to the standard FAROS (time complexity), how much memory is used (space complexity), and how many bytes could successfully detected as suspicious (detection efficiency) in each case.

In an in-memory-only attack, the attacker, usually through a shell, injects a payload inside a legitimate process address space. The hallmark of the in-memory-only attack is the following. The payload comes from the Internet and is associated with netflow tag. Then, these bytes are written into the kernel memory area where linking/loading operations occurs and are also associated with the tag export-table. FAROS flags the attack when these two tags (netflow and export-table) come together on a byte.

We implemented the in-memory attacks using the Meterpreter module from Metasploit in a way similar to that done for FAROS (M. N. Arefi, 2018).

We set up the attacker's VM (Linux Kali) and generated a shell code that ran in the victim's VM (Windows 7). This opened a session for the attacker and we then perform a remote reflective DLL injection targeting the victim process calculator.exe. As explained earlier, we consider two systems and attempt to compare their performance: (i) [FAROS] propagating aggressively all direct flows and no indirect flows as suggested in various DIFT systems including FAROS, and (ii) [MITOS] propagating all flows (direct and indirect) at the MITOS level. For (ii) we generalize MITOS to also capture and optimize direct flows, as explained earlier.

The spatiotemporal complexity and the detection performance of both systems is depicted in the following table. We ran six Metasploit shells (reverse https, reverse https proxy, reverse tcp rc4 dns, reverse tcp rc4, reverse tcp) and show the average performance. While FAROS aggressively propagates all tags (M. N. Arefi, 2018), MITOS propagates only the tags that are important based on our considered objective that measures the information flow (see Alg. 2). We note that MITOS achieves the following improvements simultaneously (even though they usually come in an antagonistic fashion): (i) it propagates fewer tags than FAROS by further alleviating the space and time needed for the tracking analysis 1.65X and 1.11X times respectively, and (ii) it can successfully detect 2.67X times more bytes that were involved in the in-memory attack.

Table 1. In-memory attack: MITOS vs. FAROS

Variable	FAROS	MITOS	IMPROVEMENT
Time (sec)	837	509	1.65 X
Space (Mega bytes)	2.21	1.99	1.11 X
Detected Bytes	543	1449	2.67 X

Message: MITOS opens new horizons of how information flow can be measured and optimized in modern DIFT systems where spatiotemporal complexity emerges as a key performance bottleneck that penalizes the enforcement of security policies, forensics analysis, real-time system inspection and reverse engineering.

4.4. Discussion and Future Work

MITOS consists of an analytical algorithm and set of optimal policies for the open problem of indirect flow propagation in modern DIFT systems.

From a theoretical viewpoint, MITOS is novel as it analytically models the tradeoffs between undertainting and overtainting and between the importance of heterogeneous code semantics and context (e.g., when different tags come across to the same provenance list). It also introduces fairness and tag-balancing: two key properties for the success of modern DIFT, and investigates the impact of different -fair degrees on system performance. While the complete problem is shown to be NP-hard, MITOS provides the optimal rules for indirect flow propagation using distributed optimization.

From a systems viewpoint, MITOS is distributed, scalable, flexible, of low complexity, applicable to most DIFT systems. In our extensive performance evaluation we demonstrated that it can be easily applied to an existing software-based DIFT system, and then shed some light on the various dimensions of the open problem. Additionally, we performed a study case scenario with an in-memory only attack. There, we showed that MITOS can improve simultaneously spatiotemporal complexity up to 40% and the detection efficiency up to 167%, compared to traditional DIFT.

Summing up, MITOS analytically studies and addresses a problem of DIFT systems that has been open for the last decade: the problem of whether indirect flows should be propagated or not; a dilemma that impedes the application of DIFT to practice. Then, we demonstrate our optimal decisioning framework under an existing DIFT system. In the remainder of the section we discuss our future work directions and some limitations of MITOS.

Scheduling management in the lists. We have assumed that the provenance lists follow a First-In-First-Out (FIFO) queue: we drop the head of the list if the list is full and an additional tag attempts to enter. We defer to future work the design of a proper tag scheduling and dropping decisioning using penalty functions for indirect flows, as Matzakos et al. did for delay tolerant networks (P. Matzakos, 2018).

Additional optimization degrees and semantic considerations. We plan to modify our objective to consider how often tags are propagated at specific program counter locations and different semantics contexts, e.g., the confluence of different types of data at a program counter location. Additionally, based on the application scenario one can easily extend our framework and consider τ as an additional control variable that would change dynamically based on the system dynamics to further improve performance or decrease memory pollution. Similarly, one could consider provenance list sizes that are dynamically changing, while keeping their sum constant. Also, we plan to evaluate MITOS in DIFT systems with different application scenarios that do not consider the dilemma of indirect flow propagation, beyond FAROS, e.g. including the "If This Then What system" that explores the impact of information flow tracking on several IoT attacks (I. Bastys, 2018).

MITOS in Hardware. To ensure implementation flexibility for different hardware platforms, MITOS can be implemented as a configurable component in a System on Chip (SoC). Configuration parameters for the MITOS algorithm can be saved in newly added model specific registers, allowing an interface to a trusted OS module or platform loader to set up the interfaces. Information flow during execution, tag information can be stored in dictionary-like structures that reside in a segmented portion of main memory. Segmentation can be performed during platform initialization, such as the Pre-EFI Initialization (PEI) portion of Unified Extensible Firmware Interface (UEFI), much like the enclave page cache is reserved for usage in Intel's Software Guard Extensions (SGX). Recently accessed information can be stored in a MITOS-specialized series of caches to mask memory latency. We move the computational process employed by MITOS to decide tag propagation to specialized hardware. We extract data flow information directly from the CPU as code executes. For out of order cores, we look at the commit stage in the CPU, as to capture the proper architectural state and not violate execution model. The decision on whether to propagate tag information is then performed by hardware.

Limitations. Note that, as our implementation was based on FAROS and PANDA, we encountered several limitations in the performance evaluation. For example, FAROS poses a large overhead on the host machine: e.g., the memory required to replay a record increases exponentially on the record duration, prohibiting us to run scenarios longer of one minute. Also, PANDA restricts the size of the record and further the system activities that can be recorded simultaneously. The latter prevented us from running complex evaluation scenarios, e.g., run multiple attacks of benchmark scenarios jointly. Finally, note that FAROS runs on Windows 7, restricting our OS choice for our case study analysis. While the OS itself does not affect the nature of the open IFP problem and the insights offered by MITOS, we plan to also apply MITOS in more modern OSes in our future work.

5.0 CONCLUSIONS

In this research work we propose MITOS, an iterative algorithm along with set of policies for optimal propagation decisioning under an indirect flow for modern DIFT systems. We have taken into consideration the tradeoffs of undertaining versus overtainting, different semantics and run context priorities, and different -fairness degrees. We then pointed out how fairness relates to tag balancing and entropy maximization, and we further claim that they are key properties to improve performance in modern DIFT. Putting everything together, MITOS theoretically studies and tackles the open problem of indirect flow propagation encountered in practical DIFT systems, by unifying the two, usually conflicting, worlds of theory and practice, opening new horizons on how DIFT should optimize their performance. Experimental evaluation sheds light on the open problem of indirect flow propagation and investigates the complex tradeoffs involved. Additionally, we performed a case-study scenario with a real in-memory attack and showed that MITOS improves simultaneously (i) system's time and memory overhead (up to 40%), and (ii) system's fingerprint on suspected bytes (up to 167%) compared to standard DIFT, even though these metrics are usually antagonistic.

6.0 REFERENCES

BIBLIOGRAPHY

https://www.passmark.com/. (n.d.).

A. M. Espinoza, J. K.-A. (2016). Vdift: Vector-based dynamic information flow tracking with application to locating cryptographic keys for reverse engineering,. *IEEE ARES.*

B. Gu, X. L. (2013). D2taint: Differentiated and dynamic information flow tracking on smartphones for numerous data sources. *IEEE INFOCOM.*

Bos, A. S. (2009). ointless tainting?: evaluating the practicality of pointer tainting. *ACM EuroSYS.*

G. E. Suh, J. W. (2004). Secure program execution via dynamic information flow tracking. *ACM ASPLOS.*

H. Yin, D. S. (2007). Panorama: capturing system-wide information flow for malware detection and analysis. *ACM CCS.*

I. Bastys, M. B. (2018). If this then what?: Controlling flows in IoT apps. *ACM CCS.*

J. Clause, W. L. (2017). Dytan: a generic dynamic taint analysis framework,. *ACM ISSTA.*

J. Newsome and D. Song. (2005). Dynamic taint analysis for automatic detection analysis, and signaturegeneration of exploits on commodity software. *NDSS.*

J. R. Crandall, F. T. (2004). Minos: Control data attack prevention orthogonal to memory model orthogonal to memory model,". *IEEE/ACM MICRO.*

J. R. Crandall, F. T. (2005). A security assessment of the minos architecture. *ACM SIGARCH Computer Architecture News*.

M. G. Kang, S. M. (2011). "Dta++: dynamic taint analysis with targeted control-flow propagation.". *NDSS.*

M. N. Arefi, G. A. (2018). Faros: Illuminating in-memory injection attacks via provenance-based whole-system dynamic information flow tracking. *IEEE/IFIP DSN*.

N. Sapountzis, T. S. (2018). Joint Optimization of User Association and Dynamic TDD for Ultra-Dense Networks. IEEE INFOCOM.

N. Vachharajani, M. J. (2004). Rifle: An architectural framework for user-centric information-flow security,. *IEEE/ACM MICRO.*

P. Matzakos, T. S. (2018). Joint scheduling and buffer management policies for dtn applications of different traffic classes. *IEEE Transactions on Mobile Computing*.

LIST OF ACRONYMS, ABBREVIATIONS AND SYMBOLS

ACRONYM	DESCRIPTION
DIFT	Dynamic Information Flow Tracking
DFP	Direct Flow Propagation
IFP	Indirect Flow Propagation
2-D	2-dimension

SYMBOL	DESCRIPTION
t	t = t1, t2,... ; tag type (e.g., t1 = network, t2 = process, etc.)
i	i = 1,2,... ; increasing number that differentiates the tags belonging to the same type
$n_{t,i}$	number of copies in memory for the tag with unique ID t,i
n	2-D optimization vector (control variable vector)
α *	fairness degree in undertainting cost
β *	* steepness of the overtainting cost
τ *	weight for the undertainting vs. overtainting tradeoff
u_t *	weight of tag type t while considering semantics, context and tag types
o_t*	weight of tag type t for the memory pollution