PIITracker: Automatic Tracking of Personally Identifiable Information in Windows

Meisam Navaki Arefi
University of New Mexico
Albuquerque, New Mexico, USA
mnavaki@unm.edu

Geoffrey Alexander
University of New Mexico
Albuquerque, New Mexico, USA
alexandg@cs.unm.edu

Jedidiah R. Crandall
University of New Mexico
Albuquerque, New Mexico, USA
crandall@cs.unm.edu

ABSTRACT

Personally Identifiable Information (PII) is information that can be used on its own or with other information to distinguish or trace an individual's identity. To investigate an application for PII tracking, a reverse engineer has to put considerable effort to reverse engineer an application and discover what an application does with PII. To automate this process and save reverse engineers substantial time and effort, we propose PIITracker which is a new and novel tool that can track PII automatically and capture if any processes are sending PII over the network. This is made possible by 1) whole-system dynamic information flow tracking 2) monitoring specific function and system calls. We analyzed 15 popular chat applications and browsers using PIITracker, and determined that 12 of these applications collect some form of PII.

CCS CONCEPTS

• **Security and privacy → Software security engineering**;

KEYWORDS

Privacy, Reverse Engineering, Dynamic Information Flow Tracking

ACM Reference Format:
Meisam Navaki Arefi, Geoffrey Alexander, and Jedidiah R. Crandall. 2018. PIITracker: Automatic Tracking of Personally Identifiable Information in Windows. In *EuroSec'18: 11th European Workshop on Systems Security , April 23–26, 2018, Porto, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3193111.3193114

1 INTRODUCTION

Personally Identifiable Information (PII) is information that can be used on its own or along with other information to distinguish or trace an individual's identity. Other information that is linkable to an individual, such as a MAC address or hard drive serial number, also counts as PII. As users face new threats to their privacy and anonymity, VPNs and anonymity tools such as Tor [14] are becoming increasingly popular. However, applications that send PII over the network still pose a threat to user privacy and anonymity.

In this paper, we propose PIITracker, a reverse engineering tool for automatic tracking of PII. PIITracker can determine whether an application is collecting PII, and, if so, does it send it out over the network. PIITracker is based on dynamic information flow tracking (DIFT) [29], also known as Dynamic Taint Analysis [22], which is a promising technology for making systems transparent. It works by tagging certain inputs or data with meta information and then propagating tags as the program or system runs, usually at the instruction level, so that the transparency of the flow of information can be achieved.

The PII that we have investigated in this paper are (1) MAC address, (2) hard drive serial number, (3) hard drive model name, (4) volume serial number, (5) host name, (6) computer name, (7) security identifier number (SID), (8) CPU model, and (9) Windows version and build.

Our goal is to develop a tool that can automatically reveal how a Windows application treats PII and determine whether an application sends any form of PII out using the network. For this end, we have three requirements. (i) We must be to able to monitor reading of PII by the application. As an input to our system, we need to know when and where PII may be read. (ii) We must be able to track PII throughout the system. Applications often read PII but use it for other purposes, never sending it our over the network. (iii) We must be able to automatically analyze an application and return the result back to the analyst.

To fulfil these requirements, we have developed PIITracker as a plugin for PANDA [15], which is a framework for dynamic analysis. We have monitored specific function and system calls to track reading PII, and we have used PANDA's taint analysis engine to track PII throughout the system.

In summary, this paper presents the following contributions:

- We propose PIITracker, a DIFT-based reverse engineering tool for PII tracking, which can automatically track PII and capture if a process sends any form of PII over the network.
- We implement PIITracker as a PANDA plugin supporting Windows 7 as the guest operating system. We also make PIITracker open source for the community, which is available for public download.
- We analyze 15 popular Windows applications using PIITracker. Our analysis show that the majority of these application collect some form of PII and send it over the network.

The paper is organized as follows. Section 2 provides background information on dynamic information flow tracking systems. Section 3 discusses PIITracker architectural design and implementation details. Section 4 presents our experimental results, analysis of false positives and false negatives, as well as performance evaluation. Section 5 discusses related works and finally section 6 concludes the paper.

```
x = y + 1;
```

(a) Direct flow.

```
x = 0;
if (y == 1)
    x = 1;
```

(b) Indirect flow.

Figure 1: Examples of direct and indirect flow

```
const char str1 = "Tainted string";
char str2[80];
char lookuptable[256];
for (i = 0; i < 256; i++)
    lookuptable[i] = i;
for (j = 0; j < strlen(str1); j++)
    str2[j] = lookuptable[str1[j]];
```

Figure 2: An example of address dependencies in C code

2 BACKGROUND - DIFT

Dynamic Information Flow Tracking (DIFT) [29], also called dynamic taint analysis [22], has been used for over a decade but has still seen limited application in real reverse engineering tasks. DIFT works by tagging certain inputs or data and then propagating tags as the program or system runs. In this way the information flow of a program can be determined.

There are two main types of flows that DIFT systems can track: direct and indirect [29]. Direct flows are data-flow based, while indirect flows are control-flow based. For example, in Fig. 1a, there is a direct flow from y to x. However, in Fig. 1b, the value of x is dependent on y, meaning that there is an indirect flow from y to x.

There are two types of direct flows: copy and computation dependencies. In a copy dependency, a value is copied from one location to another, where a location can be a byte, a word of memory or a CPU register. The simplest way to track this information flow is to propagate the tag so that the destination location is tainted with the same tag as the source location. In computation dependencies, tags must be combined. For example, after the computation for $x = y + z$ the tag for x should contain the union of the tags for y and z; or, in single-bit tag DIFT systems, x should be tainted if either y or z is tainted.

An indirect flow occurs when information dependent on program input determines how information flows. There are two types of indirect flows: *address* and *control dependencies*. Figure 2 provides an example of address dependencies. In this figure, if the characters in `str1` are tainted then the characters in `str2` at the end should also be tainted, because they carry the exact same information. This arises in real reverse engineering tasks in everything from special handling of ASCII control characters to ASN.1 encodings. The only way to ensure that `str2` is properly tainted is to propagate tags through the address dependency where `str1[j]` is used as an offset to index into `lookuptable`.

Control dependencies, illustrated in Figure 3, present a dilemma for all existing DIFT systems. In Figure 3 `taintedinput` is copied to `untaintedoutput` bit by bit. Information flows one bit at a time through the control dependency in the `if` statement. Control dependencies can also arise in real reverse engineering applications,

```
char taintedinput;
char untaintedoutput = 0;
for (bit = 1; bit < 256; bit <<= 1){
    if (bit & taintedinput)
        untainedoutput |= bit;
}
```

Figure 3: An example of control dependencies in C code

but how to handle them in a purely dynamic environment is still an open area of resaearch. See Espinoza *et al.* [17] for more details.

The taint2 PANDA plugin that we utilize in this paper tracks direct flows in the standard fashion (by copying or combining taint tags). It also handles one type of indirect flow, namely address dependencies. It achieves this by propagating the taint from addresses used to load data used in computations or copies. The taint2 plugin does not handle control dependencies. One of the interesting results of our paper is that tracking address dependencies, but not control dependencies, is sufficient to handle common operations applied to PII such as encryption and hashing.

3 IMPLEMENTATION AND DESIGN

This section describes in details *PIITracker* architectural design and implementation details.

3.1 Architecture

PIITracker was implemented as a plugin to PANDA [15, 33] which is an open-source platform for dynamic analysis. PANDA is built upon the QEMU whole-system emulator [5]. PANDA added to QEMU three main features:

(1) The ability to record and replay executions to enable whole system analysis.
(2) An architecture to streamline the addition of other plugins to QEMU.
(3) The ability to use LLVM [26] as QEMU's intermediate language instead of TCG [4] for analysis.

Figure 4 shows an overview of the PIITracker architecture. PIITracker runs on top of Linux (we used Ubuntu 14.04 for development and testing) as the host OS, and supports Windows 7 32-bit as the guest OS. PIITracker interacts with three other plugins, namely *taint2*, *syscalls2* and *OSI/Win7x86intro* which are described below:

- **taint2:** This plugin tracks information flow throughout the system via whole-system taint analysis. It provides APIs and callbacks to tag memory locations and later query a memory location to find out what tags it has. The taint2 plugin is implemented by translating TCG code to LLVM and then inserting extra LLVM operations to propagate taint as instructions execute [33].
- **syscalls2:** This plugin provides callbacks that allow notifications whenever system calls are invoked in the guest. We have modified the syscalls2 plugin to add hooks and monitor network sockets for the outgoing network flows using the NtDeviceIoControlFile system call.
- **OSI/Win7x86intro:** This plugin provides callbacks whenever a process-related event happens in the system, such as

the start/end of a new process. It also provides APIs to get information about the current process.

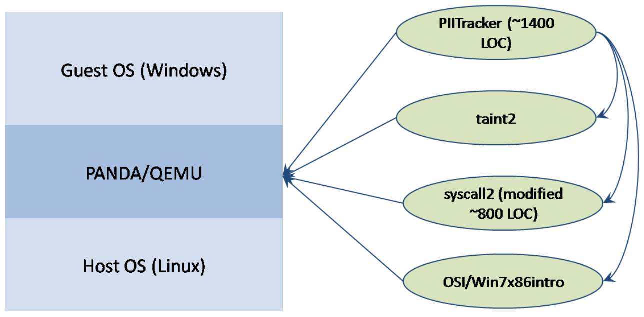

Figure 4: PIITracker architecture.

3.2 Placing Hooks

In order to track PII, we first need to add hooks to monitor any reading of PII. For this purpose, we have utilized Windows API function calls and system calls as hooks. Thus, once a specific function or system call occurs we get a callback and we parse the arguments of that function or system call and get the memory address of the desired argument. Then we taint that memory location using the *taint2* plugin API. For instance, once the target process calls *GetAdaptersAddresses*, we taint the memory location of the returned MAC address with a unique tag that indicates MAC address.

To monitor function calls, we have parsed the export table to get a callback once a function gets called. This callback includes the function name along with its arguments. Table 1 shows what function and system calls are used for each PII data point.

3.3 Query

To monitor the outgoing network traffic, PIITracker uses the *NtDeviceIoControlFile* system call. All device communications that we are concerned with utilize this system call. We parse the arguments of this system call to get the memory address of the outgoing TCP or UDP buffers. Then we query the memory address of every byte in that buffer, using a *taint2* plugin API, to determine if it has any tags. If so, we log the memory address, the destination IP address, along with the tags associated with that byte.

3.4 Usage scenario

To use PIITracker, an analyst needs to set up a Windows 7 VM, start the PANDA recording mode, and then run the Windows application they want to analyze inside the VM. Once the interesting activities are completed, the analyst stops the recording mode and initiates the PANDA replay of the recorded capture with the PIITracker plugin loaded. As a plugin input argument, analyst can specify a target process that they are interested in tracing its PII activities. Eventually, PIITracker generates an output file indicating whether the target process has sent any form of PII over the network. If such an activity is captured, PIITracker also provides the relevant memory addresses of the PII, the destination IP address, along with the full tag list of every byte.

```
>> The following PII are going over the network:
    MAC, Volume Serial Number, Hard Drive Serial Number
>> Target process: baidu-browser.exe
>> Destination address: 103.235.46.114
>> See below for details:
Memory Address | Taint Labels
---------------------------------
134203702 | Hard Drive Serial Number,
134203706 | Hard Drive Serial Number,
134203710 | Hard Drive Serial Number,
134203714 | Hard Drive Serial Number,
131910452 | MAC Address, Volume Serial Number,
131910456 | MAC Address, Volume Serial Number,
131910460 | MAC Address, Volume Serial Number,
131910464 | MAC Address, Volume Serial Number,
```

Figure 5: A sample output of PIITracker.

Figure 5 shows the actual output of PIITracker once it captures PII going out over the network.

4 EXPERIMENTAL EVALUATION

This section presents the results we obtained when evaluating PIITracker's effectiveness in capturing of PII by Windows applications. We have investigated 15 popular Windows applications using PIITracker. In addition, we have conducted several experiments to show the correctness of our tool. We have also evaluated the performance overhead of PIITracker compared to PANDA.

4.1 Analyzing Popular Windows Applications

We have investigated 15 popular Windows applications (see Table 2 column 1), mostly chat applications and web browsers. We have used the last version of these applications at the time of paper submission. Using PIITracker, we determined that 12 of these applications collect some form of PII, meaning that they send PII over the network.

Table 2 shows our results using PIITracker. It shows whether applications collect or read the PII. In case an application collects the PII, PIITracker provides the destination address of the outgoing PII. As we can see, most of these applications collect some form of PII. The chat applications that we could not find any serious PII-related privacy issues were **Telegram** and **Viber**. In addition, our study showed that **all** Chinese chat and web browser applications that we investigated collect some form of PII.

In addition, our results show that *Firefox* and *Chromimum* also collect some form of PII. We verified that Chromimum actually sends this information over the network by reading in the source code that there is a *GetMachineID* function that creates a machine ID using the volume serial number and security identifier number, which is then sent to Google.

4.2 False Positive and False Negative Analysis

To evaluate the correctness of PIITracker we have conducted several experiments which can be represented in the following two categories:

- **Comparison with previous works:** We have compared our results with the results of other researchers who have

PII data point	Monitored function and system calls
MAC address	GetAdaptersInfo, GetAdaptersAddresses, NtDeviceIoControlFile
Hard Drive Serial Number	NtDeviceIoControlFile, ZwDeviceIoControlFile, DeviceIoControlFile
Hard Drive Model Name	NtDeviceIoControlFile, ZwDeviceIoControlFile, DeviceIoControlFile
Volume Serial Number	GetVolumeInformation, GetVolumeInformationByHandle,
Host Name	gethostname
Computer Name	GetComputerName, GetComputerNameEx
Security Identifier Number (SID)	LookupAccountName, LookupAccountNameLocal
CPU Model	GetSystemInfo
Windows Version and Build	GetVersion, GetVersionEx

Table 1: Hook placement in PIITracker.

PII / Application	Network MAC Address	Hard Drive Serial Number	Hard Drive Model Name	Volume Serial Number	Host Name	Computer Name	Security Identifier Number	CPU Model	Windows Version and Build	Destination address
[1] Baidu Browser V7.6.100.2089	Yes	Yes	Yes	Yes	Read	No	Yes	Yes	Yes	*.br.baidu.com
[2] QQ Browser V9.2.5748.400	Yes	Yes	Yes	No	Yes	No	Read	Yes	Yes	wup.imtt.qq.com
[3] UC Browser V5.5.10106.5	No	Yes	Read	Yes	Read	Yes	Read	Yes	Yes	uc.ucweb.com
[4] 360 Secure Browser V9.1.0.358	No	Yes	Yes	No	No	Read	Yes	Read	Read	dd.browser.360.cn
[5] WeChat V2.6.0.56	Yes	No	No	No	Yes	Yes	No	No	Yes	qq.com 183.232.96.107
[6] Tencent QQ International V2.11	Yes	Read	Yes	No	Read	No	No	Read	Read	203.205.144.238
[7] Viber V7.5.0.97	No	No	No	No	No	Read	No	Read	Yes	content.cdn.viber.com
[8] Line V5.4.2.1560	No	Yes	Read	No	No	No	No	No	Yes	webmaster.naver.com
[9] Telegram V1.1.23	No	No	No	No	No	No	No	No	No	NA
[10] IMO V1.1.2	No	No	No	Read	No	No	Yes	Yes	Yes	192.12.31.77 38.90.96.67(PageBites Inc.)
[11] KakaoTalk V2.6.3.1672	Read	Read	Read	Read	No	Read	Yes	Read	Read	app.pc.kakao.com
[12] Firefox V57.0.3	No	Read	Yes	Read	No	Read	Yes	Read	Yes	184.51.0.249 (detectportal.firefox.com)
[13] Internet Explorer V8.0.7601	No	No	No	No	No	No	Yes	Yes	Yes	204.79.197.200 (Microsoft)
[14] Chromium V63.0.3239.108	No	Read	Read	Yes	No	Read	Yes	Read	Yes	172.217.12.3 (Google Inc.)
[15] Mullvad	No	No	No	No	No	No	No	Read	Read	NA

Table 2: Results of analyzing Windows applications using PIITracker. "No" means that the application does not read the data point and thus does not send it over the network. "Yes" means the application collects that data point. "Read" means the application only reads the data point, but does not send it out over the network.

already *manually* reverse-engineered the Windows applications under study in this paper. To the best of our knowledge, only three of them had previously been reverse-engineered for PII tracking, namely QQ Browser, UC Browser, and Baidu Browser. We have compared our results in these applications with the results in [11, 23–25]. We were able to verify the previous results about these three applications using PIITracker.

- **Evaluating PIITracker via our own developed Windows applications:** We developed three test applications that read some PII with different settings and send them over the network. The three applications developed were:
  1. An application that reads hard drive serial number and encrypts it using AES256-CBC and then sends it over the network.
  2. An application that reads hard drive serial number and encrypts it using AES256-CBC and then creates a MD5 hash of that ciphertext and sends it over the network.
  3. An application that reads a combination of PII, but does not send them over the network.

  We ran PIItracker against the above-mentioned test applications, and it worked as expected. PIITracker successfully determined that hard drive serial number is going over the network in the first and second test applications, and also correctly did not discover any PII going over the network.

4.3 Performance Evaluation

Whole-system information flow tracking is intrinsically heavyweight, and thus performance has not been a priority for PIITracker. Instead, we have focused on the accuracy and correctness of PIITracker. However, we have evaluated the performance overhead of PIITracker compared to PANDA. We have recorded the replay time in PANDA, once without PIITracker and once with PTTTracker loaded for six random applications, and then measured the overhead for each PANDA recording.

Table 3 illustrates the slowdown of PIITracker compared to PANDA. PIITracker exhibited a 67X slowdown on average compared to PANDA replay. Additionally, table 3 illustrates that PIITracker performance overhead depends on the workload of the recorded application. In other words, PANDA recordings with more complex behavior present more performance overhead.

Experimental Setup All experiments were done on a system with an Intel Core i7-6700K 4.00GHz processor, and 32G RAM running on Ubuntu 14.04. The guest was Windows 7 Ultimate 32-bit.

5 RELATED WORKS

Our research intersects with reverses engineering, DIFT, and privacy. Thus, in this section we summarize the previous works in these areas and highlight their features.

5.1 Reverse Engineering

Reverse engineering tools, such as CWSandbox [34], Norman Sandbox [3], Cuckoo Sandbox (or, CuckooBox) [2], and Anubis [1], focus on malware. The emphasis of these tools is on sandboxing the malicious code [20], identifying it, and extracting markers such as domain names. PIITracker is built on PANDA [15], and the emphasis is on using information flow tracking to detect leaks of PII. Panorama [35] and VMscope [21] were proposed as whole system QEMU-based malware analysis systems [5]. Ether was proposed to perform transparent malware analysis based on the extension of hardware virtualization [13].

In some cases exfiltration of data, such as passwords or credit card numbers, is tracked with DIFT, but with a focus on malware. Because these systems look for specific things and our focus is on widely used applications which requires a broader view of information flow, where there are many possibilities for how PII can be leaked. This is in contrast to malware analysis, where often there is a specific prescribed information flow (*e.g.*, a key logger).

5.2 Dynamic Information Flow Tracking

Most early DIFT systems ignore indirect flows, or use simple heuristics. In Suh *et al.* [29] address dependencies are not propagated if the address is calculated using a scaled index base (an x86 construct for calculating addresses). In Minos [9, 10], address dependencies are only propagated for 8- and 16-bit loads and stores, but not for 32-bit loads and stores. Additionally, as an attempt to mitigate control dependencies, 8- and 16-bit immediate values (*i.e.*, constants that are compiled into the program's machine code) were tainted automatically even if the code did not come from a tainted source. TaintCheck [27] and Vigilante [8] ignore indirect flows.

More recent DIFT systems either are designed for flexibility but offer no policies that address indirect flows [6, 12, 22, 28, 32], or are overly conservative in how they handle indirect flows [18, 30, 31].

5.3 Privacy

TaintDroid [16] detects data leakage of Android applications using variable-level tracking within the VM interpreter. It does not track taints for native code, and only applies a heuristic that propagates taints from input arguments to that of the return value of functions. Vision [19] is an extension of TaintDroid to detect indirect information flows. TaintEraser [36] identifies whether an application leaks sensitive data such as password and credit card numbers in Windows, but they require users to manually specify what actually is a password or credit card number. Continella et al. [7] proposes an obfuscation-resilient privacy leak detection approach for Android applications based on a black-box differential analysis technique. To the best of our knowledge, none of the previous works track PII in Windows in an automatic fashion.

6 CONCLUSION

In this paper, we presented PIITracker, a novel tool for tracking personally identifiable information (PII) in Windows by leveraging the synergy of whole-system taint analysis and monitoring specific function and system calls. Using PIITracker, we analyzed 15 popular Windows applications and showed that the majority of these applications collect some form of PII. PIITracker not only saves reverse engineers substantial time and effort in practice, but also provides valuable information including the relevant memory addresses of leaked PII, as well as network socket info. PIITracker demonstrates that DIFT can be a useful tool in a reverse engineer's toolbox.

ACKNOWLEDGEMENTS

This material is based on research supported by the U.S. National Science Foundation under grant Nos. #1518878 and #1518523. We would like to thank the anonymous EuroSec reviewers for their valuable comments to improve our paper. We would also like to thank Antonio Espinoza for commenting on drafts.

Application	Number of Instructions	Replay time w/o PIITracker	Replay time w/ PIITracker	X overhead
QQ Browser	2281062205	45s	46m	62X
UC Browser	763167441	18s	21m	72X
Baidu Browser	825742849	20s	22m	66X
360 Secure Browser	1085848918	22s	26m	65X
Tencent QQ International	1551545206	24s	26m	68X
WeChat	3476153268	32s	37m	69X
				Average = 67X

Table 3: Performance evaluation: PANDA vs PANDA + PIITracker

REFERENCES

[1] Accessed May 13, 2017. Anubis. http://anubis.iseclab.org/. (Accessed May 13, 2017).

[2] Accessed May 13, 2017. Cuckoo Sandbox. https://cuckoosandbox.org/. (Accessed May 13, 2017).

[3] Accessed May 13, 2017. Norman Sandbox. http://download.norman.no/\product_sheets/eng/SandBox_analyzer.pdf. (Accessed May 13, 2017).

[4] Accessed May 13, 2017. Tiny Code Generator (TCG). http://wiki.qemu.org/Documentation/TCG. (Accessed May 13, 2017).

[5] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*. ACM, Berkeley, CA.

[6] James Clause, Wanchun Li, and Alessandro Orso. 2007. Dytan: A Generic Dynamic Taint Analysis Framework. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA '07)*. ACM, New York, NY, USA, 196–206. https://doi.org/10.1145/1273463.1273490

[7] Andrea Continella, Yanick Fratantonio, Martina Lindorfer, Alessandro Puccetti, Ali Zand, Christopher Kruegel, and Giovanni Vigna. 2017. Obfuscation-resilient privacy leak detection for mobile apps through differential analysis. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*. 1–16.

[8] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham. 2005. Vigilante: End-to-end containment of internet worms. In *ACM SIGOPS Operating Systems Review*, Vol. 39. ACM, 133–147.

[9] Jedidiah R. Crandall and Frederic T. Chong. 2004. Minos: Control Data Attack Prevention Orthogonal to Memory Model. *MICRO* (December 2004), 221–232.

[10] Jedidiah R. Crandall, S. Felix Wu, and Frederic T. Chong. 2006. Minos: Architectural support for protecting control data. *ACM Trans. Archit. Code Optim.* 3, 4 (2006), 359–389. https://doi.org/10.1145/1187976.1187977

[11] Jakub Dalek, Katie Kleemola, Adam Senft, Christopher Parsons, Andrew Hilts, Sarah McKune, Jason Q. Ng, Masashi Crete-Nishihata, John Scott-Railton, and Ron Deibert. 2015. Privacy and Security Issues with UC Browser. (2015). Retrieved Febrarury 2018 from https://citizenlab.ca/2015/05/a-chatty-squirrel-privacy-and-security-issues-with-uc-browser/

[12] Michael Dalton, Hari Kannan, and Christos Kozyrakis. 2007. Raksha: A Flexible Information Flow Architecture for Software Security. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*. ACM, New York, NY, USA, 482–493. https://doi.org/10.1145/1250662.1250722

[13] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. 2008. Ether: Malware Analysis via Hardware Virtualization Extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*.

[14] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router.* Technical Report. Naval Research Lab Washington DC.

[15] B. F. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan. 2014. Repeatable Reverse Engineering for the Greater Good with PANDA. In *Columbia University Computer Science Technical Reports*. New York.

[16] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)* 32, 2 (2014), 5.

[17] Antonio M Espinoza, Jeffrey Knockel, Pedro Comesaña-Alfaro, and Jedidiah R Crandall. 2016. V-DIFT: Vector-Based Dynamic Information Flow Tracking with Application to Locating Cryptographic Keys for Reverse Engineering. In *Availability, Reliability and Security (ARES), 2016 11th International Conference on.* IEEE, 266–271.

[18] J. S. Fenton. 1973. Information Protection Systems. In *Ph.D. Thesis, University of Cambridge.*

[19] Peter Gilbert, Byung-Gon Chun, Landon P Cox, and Jaeyeon Jung. 2011. Vision: automated security validation of mobile apps at app markets. In *Proceedings of the second international workshop on Mobile cloud computing and services*. ACM, 21–26.

[20] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. 2015. Needles in a Haystack: Mining Information from Public Dynamic Analysis Sandboxes for Malware Intelligence. In *Proceedings of the 24th USENIX Security Symposium*.

[21] Xuxian Jiang and Xinyuan Wang. 2007. "Out-of-the-BoxâĂİ Monitoring of VM-Based High-Interaction Honeypots. In *International Workshop on Recent Advances in Intrusion Detection*.

[22] Min Gyung Kang, Stephen McCamant, Pongsin Poosankam, and Dawn Song. 2011. DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium*. San Diego, CA.

[23] Jeffrey Knockel, Sarah McKune, , and Adam Senft. 2016. Privacy and Security Issues with Baidu Browser. (2016). Retrieved Febrarury 2018 from https://citizenlab.ca/2016/02/privacy-security-issues-baidu-browser/

[24] Jeffrey Knockel, Adam Senft, and Ron Deibert. 2016. Privacy and Security Issues in QQ Browser. (2016). Retrieved Febrarury 2018 from https://citizenlab.ca/2016/03/privacy-security-issues-qq-browser/

[25] Jeffrey Knockel, Adam Senft, and Ronald J Deibert. 2016. Privacy and Security Issues in BAT Web Browsers.. In *FOCI*.

[26] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization*.

[27] James Newsome and Dawn Song. 2005. Dynamic taint analysis: Automatic detection, analysis, and signature generation of exploit attacks on commodity software. In *In In Proceedings of the 12th Network and Distributed Systems Security Symposium*. Citeseer.

[28] Feng Qin, Cheng Wang, Zhenmin Li, Ho seop Kim, Yuanyuan Zhou, and Youfeng Wu. 2006. LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks. *MICRO-39* (December 2006), 135–148.

[29] G. Edward Suh, Jaewook Lee, and Srinivas Devadas. 2004. Secure Program Execution via Dynamic Information Flow Tracking. In *Proceedings of ASPLOS-XI.*

[30] Mohit Tiwari, Hassan M.G. Wassel, Bita Mazloom, Shashidhar Mysore, Frederic T. Chong, and Timothy Sherwood. 2009. Complete information flow tracking from the gates up. *SIGPLAN Not.* 44, 3 (2009), 109–120. https://doi.org/10.1145/1508284.1508258

[31] Neil Vachharajani, Matthew J. Bridges, Jonathan Chang, Ram Rangan, Guilherme Ottoni, Jason A. Blome, George A. Reis, Manish Vachharajani, and David I. August. 2004. RIFLE: An Architectural Framework for User-Centric Information-Flow Security. In *Proceedings of the 37th International Symposium on Microarchitecture (MICRO)*. citeseer.ist.psu.edu/711861.html

[32] Guru Venkataramani, Ioannis Doudalis, Yan Solihin, and Milos Prvulovic. 2008. FlexiTaint: A programmable accelerator for dynamic taint propagation.. In *HPCA*. IEEE Computer Society, 173–184.

[33] Ryan Whelan, Tim Leek, and David Kaeli. 2013. Architecture-Independent Dynamic Information Flow Tracking. In *Proceedings of the 22nd International Conference on Compiler Construction (CC'13)*. Springer-Verlag, Berlin, Heidelberg, 144–163.

[34] Carsten Willems, Thorsten Holz, and Felix Freiling. 2007. Toward Automated Dynamic Malware Analysis Using CWSandbox. In *IEEE Security and Privacy.*

[35] Heng Yin, Dawn Song, Manuel Egele, and Engin Kruegel, Christopher a nd Kirda. 2007. Panorama: capturing system-wide information flow for malware detection and analysis. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. ACM, New York, NY, USA, 116–127. https://doi.org/10.1145/1315245.1315261

[36] David Yu Zhu, Jaeyeon Jung, Dawn Song, Tadayoshi Kohno, and David Wetherall. 2011. TaintEraser: Protecting sensitive data leaks using application-level taint tracking. *ACM SIGOPS Operating Systems Review* 45, 1 (2011), 142–154.