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

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 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 over the network. (iii) We must be able to automatically analyze an application and return the result back to the analyst.

To fulfill 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.
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 to index into a 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.

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 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, but how to handle them in a purely dynamic environment is still an open area of research. 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.

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
To use PIITracker, an analyst needs to set up a Windows 7 VM, start the VM, 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 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.

### 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 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.

---

**Table 1:** Examples of Function and System Calls Used for Each PII Data Point

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Taint Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>134203702</td>
<td>Hard Drive Serial Number, MAC Address, Volume Serial Number, Hard Drive Serial Number</td>
</tr>
<tr>
<td>134203706</td>
<td>Hard Drive Serial Number, MAC Address, Volume Serial Number, Hard Drive Serial Number</td>
</tr>
<tr>
<td>134203710</td>
<td>Hard Drive Serial Number, MAC Address, Volume Serial Number, Hard Drive Serial Number</td>
</tr>
<tr>
<td>134203714</td>
<td>Hard Drive Serial Number, MAC Address, Volume Serial Number, Hard Drive Serial Number</td>
</tr>
</tbody>
</table>

**Figure 4:** PIITracker architecture.

**Figure 5:** A sample output of PIITracker.
### Table 1: Hook placement in PIITracker.

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

### Table 2: Results of analyzing Windows applications using PIITracker.

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

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 heavy-weight, 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 CWSSandbox [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]. Either 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.
<table>
<thead>
<tr>
<th>Application</th>
<th>Number of Instructions</th>
<th>Replay time w/o PIITracker</th>
<th>Replay time w/ PIITracker</th>
<th>X overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>QQ Browser</td>
<td>2281062205</td>
<td>45s</td>
<td>46m</td>
<td>62X</td>
</tr>
<tr>
<td>UC Browser</td>
<td>763167441</td>
<td>18s</td>
<td>21m</td>
<td>72X</td>
</tr>
<tr>
<td>Baidu Browser</td>
<td>825742849</td>
<td>20s</td>
<td>22m</td>
<td>66X</td>
</tr>
<tr>
<td>360 Secure Browser</td>
<td>1085848918</td>
<td>22s</td>
<td>26m</td>
<td>65X</td>
</tr>
<tr>
<td>Tencent QQ International</td>
<td>1551545206</td>
<td>24s</td>
<td>26m</td>
<td>68X</td>
</tr>
<tr>
<td>WeChat</td>
<td>3476153268</td>
<td>32s</td>
<td>37m</td>
<td>69X</td>
</tr>
</tbody>
</table>

Table 3: Performance evaluation: PANDA vs PANDA + PIITracker

REFERENCES


