Universal Serial Bus forensics

in the pod slurping attack

September 2011

Papadopoulos Konstantinos

(Student ID: 3301100002)
Universal Serial Bus forensics
in the pod slurping attack

September 2011

Papadopoulos Konstantinos (Student ID: 3301100002)

Supervised by prof. V. Katos
Table of Contents

Table of Contents .............................................................................................................. 1
Table of Figures ................................................................................................................ 2
1. Abstract ......................................................................................................................... 3
2. Introduction .................................................................................................................. 3
3. Literature Review ........................................................................................................ 5
   3.1 Live vs. Dead Forensics ......................................................................................... 5
   3.2 Forensics standards and principles ...................................................................... 8
   3.3 USB forensics ...................................................................................................... 9
4. The pod slurping attack ............................................................................................... 11
5. Pod slurping detection application ............................................................................. 15
   5.1 Interface and usage .............................................................................................. 15
   5.2 The Windows Registry ......................................................................................... 18
   5.3 The Detection Algorithm ..................................................................................... 23
6. Testing the application .............................................................................................. 27
   6.1 Scenario 1 ............................................................................................................. 27
   6.2 Scenario 2 ............................................................................................................. 31
7. Analyzing the code ..................................................................................................... 33
   7.1 The algorithm ...................................................................................................... 33
   7.2 The graphical interface ......................................................................................... 40
8. Conclusions ................................................................................................................ 42
9. References ................................................................................................................... 43
Appendix ........................................................................................................................... 45
   List of variables ......................................................................................................... 45
   Code of Program.cs ................................................................................................. 46
   Code of Form1.Designer.cs .................................................................................... 52
   Code of Form1.cs .................................................................................................... 59
### Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fig. 3-1</strong></td>
<td>Traditional (Dead) forensics approach [13]</td>
</tr>
<tr>
<td><strong>Fig. 3-2</strong></td>
<td>Live forensics analysis [15]</td>
</tr>
<tr>
<td><strong>Fig. 3-3</strong></td>
<td>Typical USB storage media</td>
</tr>
<tr>
<td><strong>Fig. 4-1</strong></td>
<td>The fourth generation of the iPod and the first one with USB as a main connection interface.</td>
</tr>
<tr>
<td><strong>Fig. 4-2</strong></td>
<td>An access pattern of a pod slurping attack [5]</td>
</tr>
<tr>
<td><strong>Fig. 4-3</strong></td>
<td>Comparing file access times with a USB’s characteristic transfer rate[5]</td>
</tr>
<tr>
<td><strong>Fig. 4-4</strong></td>
<td>Different characteristic transfer rates of different USB devices [5]</td>
</tr>
<tr>
<td><strong>Fig. 5-1</strong></td>
<td>Different characteristic transfer rates of different USB devices [5]</td>
</tr>
<tr>
<td><strong>Fig. 5-2</strong></td>
<td>USB devices with their corresponding plug in timestamp</td>
</tr>
<tr>
<td><strong>Fig. 5-3</strong></td>
<td>The final report of the detection process</td>
</tr>
<tr>
<td><strong>Fig. 5-4</strong></td>
<td>Structure and items of the Windows registry [10]</td>
</tr>
<tr>
<td><strong>Fig. 5-5</strong></td>
<td>The LastWrite attribute of a registry key</td>
</tr>
<tr>
<td><strong>Fig. 5-6</strong></td>
<td>The USBSTOR registry key and the FriendlyName value</td>
</tr>
<tr>
<td><strong>Fig. 5-7</strong></td>
<td>DeviceClass registry subkeys</td>
</tr>
<tr>
<td><strong>Fig. 5-8</strong></td>
<td>Files in ascending “last accessed time” order</td>
</tr>
<tr>
<td><strong>Fig. 5-9</strong></td>
<td>Files forming a copy sequence</td>
</tr>
<tr>
<td><strong>Fig. 5-10</strong></td>
<td>Transfer rate of each file in the copy sequence</td>
</tr>
<tr>
<td><strong>Fig. 5-11</strong></td>
<td>Time period acceptable to denote a USB device as suspicious</td>
</tr>
<tr>
<td><strong>Fig. 6-1</strong></td>
<td>Files copied by a simulated pod slurping attack</td>
</tr>
<tr>
<td><strong>Fig. 6-2</strong></td>
<td>USB Device data not updated (12/09/2011 while the current date is 14/09/2011).</td>
</tr>
<tr>
<td><strong>Fig. 6-3</strong></td>
<td>Final report detects the bulk copy but fails to detect the USB device</td>
</tr>
<tr>
<td><strong>Fig. 6-4</strong></td>
<td>After the registry is flushed the application reports accurate timestamps.</td>
</tr>
<tr>
<td><strong>Fig. 6-5</strong></td>
<td>Verdict for the first scenario</td>
</tr>
<tr>
<td><strong>Fig. 6-6</strong></td>
<td>Running the application in an intact folder</td>
</tr>
<tr>
<td><strong>Fig. 6-7</strong></td>
<td>Choosing the suspected USBs</td>
</tr>
<tr>
<td><strong>Fig. 6-8</strong></td>
<td>Report shows Kingston DataTraveler as the attacker USB Device</td>
</tr>
</tbody>
</table>
1. Abstract

This essay is part of the Dissertation thesis in the Master’s in Information and Communication Technology of the International Hellenic University. It involves an overview of USB storage technologies and methods of analyzing digital forensics of those portable devices. It mainly focuses on the pod slurping attack and techniques to detect it in a Microsoft Windows environment. Based on these techniques, an application is created that examines a Windows system and assesses the possibility of a pod slurping attack in a system.

The subject is inspired by a paper published by Th. Kavalaris and V. Katos: “On the detection of pod slurping attacks” [5]. The latter is also supervising the Dissertation which is intended to be a follow up thesis on the above paper.

2. Introduction

The Universal Serial Bus is a de facto standard for interfacing between a computer and a mobile device. USB flush drives, USB hard disks, memory cards connecting via memory readers and mobile phones are just a few typical examples of devices that utilize the USB protocol. All of the above are treated by operating systems and computers in a “Universal” way. This makes forensics analysts treat the analysis of those devices in the same way.

After reviewing technologies and issues regarding the USB interface, its forensic traces and ways to discover it, the essay focuses on the pod slurping attack. The pod slurping attack stems its name from the fist device that this kind of attack was introduced, the iPod. The iPod is a device produced by Apple able to reproduce music from digital media. It is portable, small and one can use it to listen to music.
even while he is jogging. iPod has a memory to store the music files which are usually copied from a conventional personal computer. The interface between the two is based on the USB standard. Another interesting fact for USB devices is that they use the same USB port not only to communicate data but to transmit power needed for the device to work and charge its batteries. This feature adds indeed to the “Universal” purpose of the USB interface, which was designed to be a “one for all” solution for mobile devices. The iPod became very popular and many accessories were introduced to the market to facilitate it.

The pod slurping attack is done if the software is altered in a way that instead of just charging or interchanging files, it is able to “slurp” (that is copy to it) files and data from the host PC. Pretending that he wants to charge his iPod, the attacker can copy to the iPod files, photographs and data giving no clue to the victim. Of course, this kind of attack is now done with much more devices than the iPod.

At the second part of the Dissertation an application is introduced which is able to detect a pod slurping attack. The application utilizes a wizard-like environment, collecting data both from the operating system (OS) and the user. It then exploits the traces left over to the OS of a PC by the pod slurping attack. Finally it is able to inform the user whether he is a victim of such an attack with some degree of certainty. The time period between the attack and the investigation is crucial for the reliability of the application’s outcome.

Finally the essay stresses some drawbacks that a pod slurping detection technique might have together with anti-forensic ways that one may use to cover his tracks.
3. Literature Review

The use of computers and its accessories has been growing exponentially in the last decades. While businesses and professionals benefit from the computational power and memorizing capacity of modern computers, outlaws and cons also facilitate new technologies to achieve their goals. It is widely understood among law enforcement agencies around the world that digital forensics is becoming more and more important and a lot of forensics’ scientists are focusing on computer investigation. However, computer accessories have been remarkably neglected from such research.

3.1 Live vs. Dead Forensics

Digital Forensics is defined as “the process of copying data from a computer in a forensic manner” [13]. A more descriptive definition is given by the US-CERT team which defines computer forensics as “the discipline that combines elements of law and computer science to collect and analyze data from computer systems, networks, wireless communications and storage devices in a way that is admissible as evidence in a court of law” [14].

Forensics acquisition includes transferring the data from the crime scene to a safe location and in a safe storage so they can be analyzed without being contaminated. The scope of the forensic analysts stops when interpretation of the gathered data is involved [15].

Digital forensics are divided in “Live” and “Dead” or conventional forensics. Live are the forensics gathered from systems that are in use at the time of the forensic investigation. That includes computers that are in use, mobile phones, live networks etc. Dead forensics are more straightforward and more documented. It regards acquiring forensic data from a non operative system at the time of the investigation.
In both cases the scope of the forensic investigation is to acquire the evidence without altering the original source. Furthermore any investigator should be able to verify that the evidence is the same as the original.

Traditional forensics approach is simple and straightforward. The investigator determines whether the system is on or off. If the system is on then he powers it off with the predefined way or by pulling the plug. Then he removes the hard drive and attaches it to an external drive and copies it contents without modifying the original hard disk.

![Fig. 3-1 Traditional (Dead) forensics approach [13]](image)

The traditional approach has some limitations that are now considered important. Firstly, shutting down or, even worse, pulling the plug leads to data alternation or data loss, specifically data which resides in the main memory. Secondly no network data are being saved. Data such as open network ports which is becoming valuable forensic information is lost when a system is powered down.

When a system is powered on then a different approach should be followed in order to preserve information that would be lost if the system is unplugged. When the investigator investigates a live system he has to select whether he will perform a local or a network analysis. Both options have advantages over the other. He then
chooses whether he wants to conduct the investigation overtly or covertly. After that he attaches an external hard drive to copy the content of the storage media.

![Flowchart of live forensics analysis](image)

**Fig. 3-2** Live forensics analysis [15]

A number of issues have to be dealt with in live forensics. If the current user does not have administrative rights then the work of investigation becomes more complex. Furthermore, a lot of computers have virtual machines running in them which need a special approach since both the host and the guest machine have to be investigated. However, data modification is the most prominent concern when investigating a live system.
3.2 Forensics standards and principles

ACPO’s “Good Practice Guide for Computer Based Electronic Evidence” is a worldwide standard in digital forensics investigation. It is explicitly stated, its purpose is to define methodologies for preserving data mainly in standalone computers, leaving other media such as USB Flash Drives in a second place [1][2]. However, practices and especially principles defined are universally applied to all digital media.

Four principles are defined that have to be followed, so that evidence are preserved and are usable in court. The first principle states that no evidence has to be altered by any law enforcement agency. That practically means that firstly some procedures have to be followed, with respect to the nature of the media, so that evidence will not accidentally be altered. Secondly, some low level technical procedures have to take place in order to copy data in another media and conduct the investigation on that so that the one found in the scene investigation remain intact. It is necessary to mention that the mere startup or connection of a device may alter much of its storage, since all operating systems write a lot of information without the user’s explicit consent. These may include, access times, dates, event logs etc.

The second principle defines the possibility of an exception to the first principle. In exceptional circumstances an investigator might decide that is more important to break the first principle for the sake of evidence preservation. Sometimes, an immediate action has to be taken so that no evidence is destroyed. That might be a working computer, or even worse a computer in the process of deleting valuable data. In such a case the guide allows the alternation of data but it is stated that agents should have a concrete reason for doing so.

The third principle states that all actions should be documented in an audit trail. This trail could later be used to assess the conformity to ACPO’s principles and law restrictions. The series of events are crucial since digital data are easily changed.
The documentation of actions is a profound characteristic of a serious and dedicated law enforcement team.

Lastly, the fourth principle of the ACPO’s Guide refers to a person who has to be in charge of the scene investigation. The case officer should supervise the investigation team and make sure that all actions comply with regulations and principles [1].

3.3 USB forensics

Although there is some research on how to collect evidence from small device with volatile memory such as UFDs (USB Flush Drives), there is no standardized methodology or best practices available. An effort to make a complete guide for collecting forensics evidence from portable devices has been done by the U.S. National Institute for Standards and Technology [3]. The Guide covers the also fast growing market of PDA and mobile phones. A lot of mobile phones have computing capabilities and operate using different Operating Systems, similar to standalone computers. It also includes procedures for the preservation and collection of evidence from PDAs and software tools used for portable devices. Some references include the use of USB Flash Drives, but only in relation to PDAs.

Fig. 3-3 Typical USB storage media

As soon as the vulnerabilities of the USB devices were revealed, countermeasures were also proposed to deal with USB security. Some methods are
introduced by Nick Cavalancia [6]. Data theft is serious incident in a business and it should not be dealt with isolated measures rather than applying a security strategy throughout the enterprise. Workarounds to the problem of data theft by using USB devices include physically disabling or locking USB ports disabling the use of USB ports by the Operating System. A better policy is to install software that monitors and logs USB connection activity, so that forensics data are being preserved.

The risks from the use of USB Devices are summarized in [7] by Marwan Al-Zarouni. USB devices were not able to steal any data from the host computer because no application could run without the consent of the user. However, things changed dramatically with the introduction of the U3 technology which later became a standard for many of the USB devices. U3 implements an autorun feature that tries to make portable not only data and files, but applications as well. When a U3 enabled device is plugged in, an autorun program runs and installs U3 functionality in the host computer just like the CDs autorun feature.

U3 was easily exploited by hackers to be used for non-legitimate uses. The change of the initial .iso file could turn U3 in a pod slurping application [7]. Instead of running the U3 Launchpad it would copy files from the host computer. Ways of dealing with such threats are physically restricting USB ports, informing the employees-users, disabling the autorun feature, restricting access rights and anti-virus protection.
4. The pod slurping attack

The iPod was released in 2001 by Apple was a digital, portable media player able to playback digitally encoded music. The first three generations were connected via FireWire cable. The fourth generation became compliant with the widely adopted USB connectors. Its size and style made it an immediate commercial success.

Soon after its release security concerns were also raised. Security expert, Abe Usher [8] claimed in 2005 that one could manipulate the iPod to unauthorizingly copy files from a host computer to the iPod. He also made a proof-of-concept application called slurp.exe that when copied to the iPod was able to “slurp” files from a host Windows running computer, immediately after its connection to the USB port. The attack got the name that Abe Usher gave to it: “pod slurping”. The attack was generalized to all USB storage devices that were gaining popularity at the time.

In 2010 Th. Kavalaris and V. Katos proposed a method of detecting a possible pod slurping attack [5]. They claimed that a forensics analyst might detect a pod
slurping attack by comparing the last accessed timestamps of the files with the time of insertion of the USB device. This could be done by acquiring the data from various operating system resources like the Windows registry.

Most importantly, they claimed that the forensics analysis might even detect which specific USB device model was the one that conducted the attack. That can be done by comparing the transfer rate of the alleged attacked file to the characteristic transfer rate of the USB device. As seen in summarizing figure 4.2 a number of files are being accessed sequentially. This implies a possible pod slurping attack. This argument is enforced by the time of the insertion of a suspicious USB device.

![Diagram](image)

**Fig. 4-2** An access pattern of a pod slurping attack [5]

To further ground the argument the authors compare the transfer rate of those files to the characteristic transfer rate of a USB device. The speed of read and write for every USB device is specific and is dependent on the technologies utilized (ie. USB version, storage technology etc.). This characteristic transfer rate can be determined by testing.
Figure 4.3 shows a scatter plot of access times of files and the corresponding regression line that defines the transfer rate. If that transfer rate matches the transfer rate of a USB storage device then that would be an additional clue to enforce the claim of the pod slurping attack using this specific USB device [Fig. 4.4]

Figure 4.4 Different characteristic transfer rates of different USB devices [5]
Based on the above criteria an application is proposed to automatically detect a possible pod slurping attack in a Windows system.
5. Pod slurping detection application

5.1 Interface and usage

The application has a simple wizard-like environment similar to many windows applications. The initial screen contains information about the application and how it is used.

![Pod Slurping Detector](image)

**Fig. 5-1** Different characteristic transfer rates of different USB devices [5]

Interaction with the user is needed in order to identify USB devices that do not belong to the legitimate user or are suspicious both based on the type of the device and on the time it was last plugged in. Hence, at the next screen, a list of all the devices ever connected to the system is presented to the user. Next to each USB device name there is a column with the timestamp of the last plug-in of the device.
This information is valuable, since the legitimate user usually suspects that the attack took place on a specific time period. Therefore, the user is prompted to check the suspicious devices.

![USB forensics in the pod slurping attack](image)

**Fig. 5-2** USB devices with their corresponding plug in timestamp

This information might even be enough for the user to conclude that no unauthorized device was logged by the system or no devices were found to be unknown or plugged in during the suspected period.

The application only examines the possibility of an attack only with respect to the USB devices chosen by the user. So the user narrows down the search and calculations made by the application. Someone might of course, choose all the devices and let the application do a complete investigation.

After examining the files and their timestamps and comparing them against the timestamps of the suspected devices [Fig. 5.2], the application examines the system and presents its final report.
The report gives two important pieces of information. The one is the percentage of files that seemed to have been massively copied. This percentage would usually be smaller as more time went by since the alleged attack. That is normally the case since the more a workstation is used the more files are usually accessed. When files are accessed, valuable information about the “chain of events” during the attack are lost. In the case that all files in question were accessed due to a backup procedure or similar function the detection algorithm will most possibly be unreliable [5].

The next information given is weather the copy sequence has started soon after the USB device was plugged in. This is usually the case when an attack takes place. The attacker starts the process manually or the USB devices trigger the autorun procedure that initiates the attack.

It is also mentioned that the tool only intends to indicate possible attacks by presenting justifiable data. In no case are the results presented 100% certain since a
lot of factors affect the examination and the verdict drawn. A list of those factors are
given below:

1. Time span of workstation used between the suspected attack and the
   examination
2. Backup processes taken place
3. Logging procedures being disabled or enabled
4. Clock consistency / modification
5. Registry integrity

It should also be mentioned that a sophisticated attack might also disable
logging of events or even distort them to cover its tracks. Although the latter would
require administrational rights and a better organized attack, however it is not
impossible. The event of change in the status of those logging procedures might also
be checked leading to conclude that a more sophisticated attack could be detected
by a more sophisticated detection algorithm.

5.2 The Windows Registry

The most important aspect of the project is the algorithm used to detect
weather the host PC has been a victim of the pod slurping attack. At first,
information about the USB devices used by the host system has to be mined from
the operating system, in our case Windows. The two main sources of information are
the Windows’ registry and setupapi.log file [9]. While the later contains information
mostly about the installation of a USB device, the registry contains information both
at the first use of the device and on its most recent use.

The Windows’ registry is a hierarchical tree-like database that contains
information about the system, devices attached to the system, applications installed,
parameters of hardware and software, configuration and user details [10]. It was
introduced to replace the many files that holed this information and formed a
central source of configuration information.
The Registry consists of hives which are the equivalent of folders in the file system. Hives categorize registry information. They contain keys which contain values and subkeys. The values hold specific data and are the actual holders of configurational information. Each value is able to have specific data, meaningful for the specific operating system and application [Fig. 5-4].

The Registry information is spread across many files, while some parts of the registry is formed when the system boots. Most of these files reside in the \%SYSTEMROOT\%\system32\config folder. HKEY_LOCAL_MACHINE\SYSTEM hive is held in a system file called “System”, while HKEY_LOCAL_MACHINE\Software hive is held in “Software”. Each users’ preferences and configuration resides in the users’ “Document and Settings” folder in NTUSER.DAT file [10].

The most popular way of accessing the registry is through the build in Windows “Registry Editor”. The Registry Editor has a graphical environment and lets the user browse through all the registry keys [Fig. 5.4].

Apart from the keys themselves, relative data are maintained just like the timestamps and in the file system. Hence, the Registry keys have a value associated with them called the “LastWrite” time that denotes the last time this key was modified. This information is quite valuable in forensics and is heavily used in the
detection algorithm as we will later see. The timestamp is held as a FILETIME object. A FILETIME objects denotes time as the number of 100 nanosecond intervals that passed since 1 January 1601. An easy way to extract the “LastWrite” attribute of a key is to highlight a key in the registry editor, right click and export as a text file [Fig. 5-5].

The registry contains information about USB removable storage devices. A record of all the USB devices connected to the system is kept in HKEY_LOCAL_MACHINE\System\ControlSet00x\Enum\USBSTOR. Below this key there is a list of all the devices ever connected to the system. That includes pen drives, digital cameras, card readers etc. For each device there is a device ID, which is usually the serial number of the device. Since serial numbers are unique, there is an easy way to identify USB devices using this information. Not all USBs however, have a serial number [10].

Apart from the IDs, a lot of parameters are recorded, all of them during the installation of the USB device. One useful parameter, for the interface of the application created, is the “FriendlyName” value that gives a human readable name of the device, particularly useful for user identification.
USB forensics in the pod slurping attack

Fig. 5-6 The USBSTOR registry key and the FriendlyName value
Although the USBSTOR contains a lot of information about the USB devices, it is modified only at the creation of the key, that is at the first plug-in. There is another registry key that is updated every time a USB device is inserted. That is HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\DeviceClasses\{53f56307-b6bf-11d0-94f2-00a0c91efb8b}. This key contains all the USB devices as subkeys and the data of those subkeys are updated every time a USB device is plugged-in. By extracting the “lastWrite” attribute of that key one can retrieve the timestamp of the last plug-in of the USB device.

Fig. 5-7 DeviceClass registry subkeys
5.3 The Detection Algorithm

The main algorithm that detects whether the workstation in question has been a victim of a pod slurping attack consists of two phases. At the first phase the application examines the “Documents” folder to calculate whether files of that folder have been sequentially copied. This is usually the case in a pod slurping attack, i.e., the massive copying of files. When such a sequence is detected, the start of it is being compared with the time of the last plug-in of the suspected USB devices. If the timestamps are close to each other then that is a clear indication of a pod slurping attack.

In the first phase of the detection all the files are read by the file system and are stored in an array with their attributes (file size, time last accessed etc). These files are sorted by the last accessed attribute in ascending order. This way the application will be ready to compare the timestamps.

Fig. 5-8 Files in ascending “last accessed time” order

The size of each file is then divided by the interval time between each pair of files. This amount denotes the transfer rate which has to be almost the same, if files are copied to the same USB device.
Hence, if there is a sequence of files in which the transfer rate is the same or nearly the same, which is typically the case for files copied to the same USB device, then this sequence is considered an evidence of possible mass file copy. However, some time period may have passed since the attack and some files might be accessed again which will break the copy sequence. To incorporate such an event the algorithm calculates the percentage of files that belong to such a sequence and states that to the user for his additional evaluation.

The acceptable transfer rate in our case (in order for a sequence of files to comply to the “copy sequence” criteria) is set to no more than 10 bytes/sec. This limit is loose enough to include all the characteristic transfer rate of any USB device.
At the second phase the sequence is compared against the plug-in of the candidate attacker. The algorithm also sets a loose criterion; A time interval of 30 minutes is the maximum limit so as to include the possibility of a manual pod slurping attack. The minimum limit is set to -5 minutes which might be surprising since the copy sequence cannot start before the plug-in of the USB device. This is defined by design as testing showed that Windows might delay the update of the registry for some time leading to this, at first sight, oxymoron phenomenon.

**Fig. 5-10** Transfer rate of each file in the copy sequence
Fig. 5-11 Time period acceptable to denote a USB device as suspicious
6. Testing the application

6.1 Scenario 1

After explaining basic principles of the application, some testing examples will be presented along with their outcome. The examples are set as scenarios with different parameters. At first a rather trivial example is set up. There are only a few files under examination that are copied to a USB Flash Drive. After that, the application is executed to test the results.

As seen in Fig. 6-1, the “Date accessed” property of the Windows file system (2\textsuperscript{nd} column), shows that all files were copied nearly at the same time. Although the time shown is not accurate, a more accurate timestamp is held. The application examines all the files in the root of the Documents Folder. This includes all files weather they are hidden, system or read-only. An immediate execution will give unexpected results. The screen where the user chooses the suspicious USB devices does not seem accurate. Although the USB device were plugged in right before the beginning of the bulk copy, the USB Device plug-in time is not updated.
Fig. 6-2 USB Device data not updated (12/09/2011 while the current date is 14/09/2011)

This is the case because Windows may postpone the update process of the timestamps for as long as an hour [5]. A cold restart might not be enough for the timestamps to be flushed to the secondary memory.

It should also be mentioned here that Windows 7 do not update the “Date accessed” property by default. It should be manually enabled by the user before the attack takes place. Otherwise, the attack would not be able to be detected with the proposed method [11].
Fig. 6-3 Final report detects the bulk copy but fails to detect the USB device

After the registry is flushed the application reports accurate timestamps.

Fig. 6-4 After the registry is flushed the application reports accurate timestamps.

After choosing the suspected devices and getting the final report, the suspected USB is identified and reported since the beginning of the copy sequence and the plug-in times match. However one can observe that the percentage of the files copied has changed. This is because while using Windows, files are being accessed. In our case hidden file “desktop.ini” is accessed in boot time.
Fig. 6-5 verdict for the first scenario

Secondly, comparing the plug-in timestamp with the “Last accessed” timestamp in Fig. 6-1, one can observe that the plug in seems to have taken place some minutes after the copying of the files has taken place. This phenomenon was explained earlier and is again due to the Windows inaccuracy in logging. Nevertheless, windows logging processes were not designed to fit the needs of a forensic investigation.
6.2 Scenario 2

Testing further we start by running the application in a folder where no deliberate mass copy has taken place. Since this is the case we expect to see a very low “copy sequence” percentage. Indeed at the first execution the application shows a very low percentage of the files seem to have been sequentially copied.

![Final Report: Percentage of files sequentially copied: 1%
Suspected USB Devices: None](image)

**Fig. 6-6** Running the application in an intact folder

Right after that, a pod slurping attack is simulated and the application is executed again to test its operation.

![List of USB devices](image)

**Fig. 6-7** Choosing the suspected USBs
Fig. 6-8 Report shows Kingston DataTraveler as the attacker USB Device
7. Analyzing the code

7.1 The algorithm

In this part, a more deep view of the algorithm is attempted. The parts of the code itself, is more thoroughly explained. The analysis of the code is addressed to researchers wishing to dig into the details of the implementation of the application aiming at extending or modifying the algorithm or using the programming techniques for similar forensics applications.

The application is written in C#. The choice of a .net language was imperative since the forensics analysis is restrained in Windows systems. C# was chosen as an appropriate programming language for implementing low level and system functions, necessary for the forensics investigation. However, since the .net framework utilizes the Intermediate Language, any .net programming language could have been chosen, such as Visual Basic or other.

All the code is part of the podslurping namespace and the main functionality is part of the Program class. The Form1 class is used to incorporate the visual interface functionality.

```csharp
namespace podslurp
{
  static class Program
  {

      At first, some unmanaged code is necessary to access the registry. With the term “unmanaged code” we refer to code bypassing the usual Common Language Runtime Virtual Machine and using directly binary Windows’ procedures. This is necessary to access the registry and retain registry’s parameters as the Last Access timestamps etc.
```
The two functions incorporated are part of the advapi32.dll which includes an API for accessing the registry and security parameters of Windows. The “RegOpenKeyEx” is used to open a registry key and “RegEnumKeyEx” is used to enumerate subkeys of an open registry key.

The application starts by showing the initial screen with instructions to the user. After that, the next screen is prepared. For that a full list of all the USB devices ever connected to the system has to be established. That is done by the populateUSBs() function.

In order to populate the list of the USB devices, data have to be retrieved from the registry. Hence, at first the registry key is defined and opened. A list of all
the subkeys containing the USB devices is made and stored in USBSubKeys string array.

```csharp
String basekey = @"SYSTEM\ControlSet002\Control\DeviceClasses\{53f56307-b6bf-11d0-94f2-00a0c91efb8b}"
    RegistryKey registry = Registry.LocalMachine.OpenSubKey(basekey, false);
    string[] USBSubKeys = registry.GetSubKeyNames();
```

Next the main part of the function takes place. A set of string manipulation commands draws the name of each USB device. Although the name of the USB device is found in the FriendlyName attribute, it is chosen to draw the name this way to avoid pairing the registry keys, which might lead to inconsistencies and errors.

```csharp
foreach (string key in USBSubKeys)
{
    if (key.IndexOf("USBSTOR") >= 0 && key.IndexOf("&Ven_") != 0)
    {
        friendlyKey = key.Substring(key.IndexOf("&Ven_") + 4, key.Length - key.IndexOf("&Ven_") - 5);
        friendlyKey = friendlyKey.Substring(0, friendlyKey.IndexOf("#"));
        friendlyKey = friendlyKey.Substring(0, friendlyKey.IndexOf("&Rev"));
        friendlyKey = friendlyKey.Replace("&Prod_", " ");
        friendlyKey = friendlyKey.Replace("_", " ");
        friendlyKey = friendlyKey.Replace("&", " ");
        if (friendlyKey == "") { friendlyKey = "Unknown"; }
        //allkeys = allkeys + friendlyKey + "\t\t";
        USBdevices[devices] = friendlyKey.PadRight(30);
    }
}
```

After retrieving the human readable name, the registry subkeys is opened in order to retrieve last access time information.

```csharp
if (RegOpenKeyExA(HKEY_LOCAL_MACHINE, basekey + "\" + key, 0, KEY_READ, out hKeyLPIT) == 0)
```
The out parameter hkeyLPIT is a handle for the key in question. Using that handle the RegOpenKeyEx() function is called which enumerates all the keys' subkeys. The subkeys are not actually needed, but the function is called in order to get the last parameter called lpftLastWriteTime. This variable carries the last access time in FILETIME format. After its conversion, the value is stored in the array USBtimes which has all the last access times for every USB device.

```csharp
if (RegEnumKeyExA(hKeyLPIT, 0, bufLPIT, ref dwBufSizeLPIT, (IntPtr)null, (IntPtr)null, (IntPtr)null, out writeTimeLPITs) == 0)
{
    USBtimes[devices] = DateTimeFromFileTime(writeTimeLPITs);
    ++devices;
}
else
{
    System.Console.WriteLine("Error code returned ___" + RegEnumKeyExA(hKeyLPIT, 0, bufLPIT, ref dwBufSizeLPIT, (IntPtr)null, (IntPtr)null, (IntPtr)null, out writeTimeLPITs).ToString() + " " + bufLPIT.ToString());
}
```

The above are executed for each USB device and after that the registry keys are closed to release the memory.

```csharp
if (devices>1) {registry.Close();}
else
{
    MessageBox.Show("No USB Devices Found!");
}
```

The devices stored in USBdevices[] array and the times stored in USBtimes[] are presented in the second screen of the application. As soon as the user chooses the USB devices and clicks on the Next button the readFiles() procedure is called. As
its name denotes it is responsible to read the files of the “My Documents” folder (without the subfolders) and examine them in order to see if there is a sequence of files accessed sequentially.

At first the home directory is defined in MyDocs variable. Next the GetFiles() function is called to get the file paths and names of every file in the “My Documents” folder.

```csharp
public static void readFiles()
{
    String MyDocs = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
    string[] filePaths = Directory.GetFiles(MyDocs);
    int numOfFiles = filePaths.Length;

    The names are later used to get the FileInfo of the file which contains all the file attributes.

    int j,i=0;
    foreach (String fileName in filePaths)
    {
        fileDetails[i] = new FileInfo(fileName);
        ++i;
    }

    The FileInfo are now placed in the fileDetails[] array. Next, the BubbleSort algorithm is used to sort in ascending Last Accessed time order.

    //Sort files in ascending LastAccessTime
    FileInfo temp;
    for (i = 1; i <numOfFiles; i++)
    {
        for (j = numOfFiles-1; j >= i; j--)
        {
            if (fileDetails[j - 1].LastAccessTime.CompareTo(fileDetails[j].LastAccessTime) > 0)
            {
                temp = new FileInfo(fileDetails[j - 1].ToString());
            }
        }
    }
```
The array now has to be examined for files being accessed sequentially. That is going to be done by comparing the Last Accessed times with the size of the file. That is indeed the transfer rate, and it should generally stay below a threshold in order for the algorithm to consider the files in question as sequentially copied. As soon as the application finds two of those files, it denotes the first one as first file of the sequence (firstFile variable) and runs ahead to see where the chain of sequentially accessed files goes. The number of those files in comparison with the total files of the folder (including hidden and system files) denotes the percentage of copied files.

```csharp
int filesCopied = 0;
    const double maxTransRate = 0.1;
```

```csharp
double interval, firstfile_interval;
    DateTime firstFile = new DateTime();
    firstFile = fileDetails[1].LastAccessTime;
    for (i = 1; i < numOfFiles; i++)
    {
        interval = fileDetails[i].LastAccessTime.Subtract(fileDetails[i-1].LastAccessTime).TotalSeconds / fileDetails[i-1].Length;
        firstfile_interval =
            fileDetails[i].LastAccessTime.Subtract(fileDetails[1].LastAccessTime).TotalMinutes;
        if ((interval < maxTransRate) && (firstfile_interval < 240))
        {
            ++filesCopied;
            if (filesCopied == 1)
            {
                firstFile = fileDetails[i].LastAccessTime;
            }
        }
```
percFilesCopied = (filesCopied *100)/ (numOfFiles - 1);

It has to be clarified that the algorithm checks against the possibility of pod slurping attack against the whole MyDocuments folder. Alternation of the algorithm might be more efficient in detecting partial pod slurping attack etc. (See below: Considerations)

At the next phase the Last accessed time of the first file of the sequence is compared against the last plug-in times of the USB devices chosen by the user. The choices of the user are stored in checkedDevices array which is retrieved by the corresponding Form. If the last plug-in time is 5 minutes after or 30 minutes before the copy of the first file then the USB Device is considered suspicious and is reported accordingly.

//Mark as suspected if plugged in at most half an hour before mass coping takes place
foreach (int choice in checkedDevices)
{
    DateTime firstFileClone;
    firstFileClone = new DateTime();
    firstFileClone = firstFile;
    if ((firstFile.Subtract(USBtimes[choice + 1]).TotalMinutes <= 30) && (firstFile.Subtract(USBtimes[choice + 1]).TotalMinutes >= -5))
    {
        suspectedUSBs[choice+1]=true;
    }
}

An array called suspectedUSBs is made to denote which of the USB devices fall under the above conditions.
7.2 The graphical interface

To complete the reference to the code, we continue with a reference to the graphical interface of the application. The GUI consists of a Form that contains three panels, one for each screen. When one panel is active, the other two are not visible. This way the application flows from one screen to the other when the user clicks next button.

```csharp
public Form1()
{
    InitializeComponent();
    this.panel2.Visible = false;
    this.panel3.Visible = false;
}
```

When the start button is pressed at the initial screen the Form calls the populateUSBs() function. The data collected from the populateUSBs() function are being added to a checklist box so that the user put a checkmark to the ones he choses. The visible property is changed to change to the next screen.

```csharp
private void button1_Click(object sender, EventArgs e)
{
    podslurp.Program.populateUSBs();
    int i=1;
    while (podslurp.Program.USBdevices[i]!=null)
    {
        this.checkedListBox1.Items.Add(podslurp.Program.USBdevices[i] + " \t " + podslurp.Program.USBtimes[i]);
        ++i;
    }
    this.panel1.Visible = false;
    this.panel2.Visible = true;
}
```
The next screen works in a similar way. When the “Next” button is clicked the Form passes the values of the checked USB devices to a property of the Program Class. This property is used to apply the detection algorithm incorporated in the readFiles() function, which is called right afterwards. The data calculated by the radFiles() function are then drawn by the Form and presented in the next screen. Those are the suspectedUSBs[] along with the USBtimes[] arrays which contain all the devices that the algorithm found as suspicious and the percFilesCopied public variable which denotes the percentage of the files sequentially copied. Visibility properties are changed accordingly.

```csharp
private void button2_Click(object sender, EventArgs e)
{
    podslurp.Program.checkedDevices =
this.checkedListBox1.CheckedIndices;
    podslurp.Program.readFiles();
    this.textBox5.Text = "Percentage of files sequentially copied:" +
podslurp.Program.percFilesCopied +"%\r\n";
    this.textBox5.Text += "Suspected USB Devices: \r\n";
    int i;
    Boolean atLeastOne = false;
    for (i = 1; i < 50; i++)
    {
        if (podslurp.Program.suspectedUSBs[i] == true)
        {
            this.textBox5.Text += "    " +
podslurp.Program.USBdevices[i] + " \t " + podslurp.Program.USBtimes[i] +
"\r\n";
            atLeastOne = true;
        }
    }
    if (atLeastOne == false) { this.textBox5.Text += "None"; }
    this.panel2.Visible = false;
    this.panel3.Visible = true;
}
```
8. Conclusions

Based on the above analysis and implementation some observations and conclusions have been drawn. Firstly we saw that, a lot of information about the usage of a workstation is kept in the registry and other files of the system. Those data are valuable for the forensics analyst. Having said that, it should also be noted that the usage data are not as refined and as precise as a forensics analyst would need for a complete analysis. Some forensics data, like timestamps etc, are not updated regularly and not precise. It seems that the registry was not designed having the forensics analysis in mind rather than the needs of the operating system itself.

A second point regards the detection algorithm. A lot of discussion could be made about the scope and the strictness of the algorithm. The algorithm proposed in this dissertation tried to detect a pod slurping attack that copied a full directory. Other researches might define the attack in a different way and detect a partial attack, or an attack at a specific type of file, or conduct a more thorough examination both in terms of memory and in terms of timeframe. Such an analysis falls out of the scope of this dissertation.

Overall the aim of this essay was not to cover all aspects of the pod slurping attack, rather to implement an application which is a proof of the concept that such a forensics analysis is possible in an automatic way and conducted by an average user himself.
9. References


[7]. Marwan Al-Zarouni, (2006), ”The Reality of Risks from Consented use of USB Devices”, School of Computer and Information Science, Edith Cowan University


Appendix

List of variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Physical meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkedDevices</td>
<td>Array containing the choices of the user</td>
</tr>
<tr>
<td>devices</td>
<td>Pointer of the USBdevices[] array</td>
</tr>
<tr>
<td>fileDetails[]</td>
<td>File properties such as Last accessed, size, Last modified, owners etc.</td>
</tr>
<tr>
<td>filePaths[]</td>
<td>Path and filenames of all the files in the folder</td>
</tr>
<tr>
<td>filesCopied</td>
<td>Files sequentially copied</td>
</tr>
<tr>
<td>firstFile</td>
<td>The file denoted as the first file of a sequence of sequentially copied files</td>
</tr>
<tr>
<td>firstFile_interval</td>
<td>The timestamp of the first file of the sequence</td>
</tr>
<tr>
<td>firstFileClone</td>
<td>A clone of the first file handle to be used for calculations</td>
</tr>
<tr>
<td>hKeyLPIT</td>
<td>Handle of a registry key</td>
</tr>
<tr>
<td>Interval</td>
<td>Time elapsed between the access time of two files</td>
</tr>
<tr>
<td>maxTransRate</td>
<td>Maximum accepted transfer rate in sec/byte. If a file is copied in time larger</td>
</tr>
<tr>
<td></td>
<td>than this then it is supposed to not be sequentially copied.</td>
</tr>
<tr>
<td>MyDocs</td>
<td>Full path of the “My Documents folder”</td>
</tr>
<tr>
<td>numOfFiles</td>
<td>Number of files in the MyDocs folder</td>
</tr>
<tr>
<td>percFilesCopied</td>
<td>Percentage of files of the folder that was sequentially copied</td>
</tr>
<tr>
<td>suspectedUSBs[]</td>
<td>USBs considered suspected by the algorithm</td>
</tr>
<tr>
<td>USBdevices[]</td>
<td>Array of all the USB devices ever connected to the system</td>
</tr>
<tr>
<td>USBtimes[]</td>
<td>Last plug in time of every USB device in USBdevices[]</td>
</tr>
</tbody>
</table>
**Code of Program.cs**

```csharp
using System;
using System.IO;
using System.Collections;
using System.Text;
using System.Runtime.InteropServices;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using Microsoft.Win32;

namespace podslurp
{
    static class Program
    {
        public static IntPtr HKEY_LOCAL_MACHINE = new IntPtr(0x80000002u);
        const int KEY_READ = 0x20019;
        public static string[] USBdevices = new string[50];
        public static DateTime[] USBtimes = new DateTime[50];
        public static bool[] suspectedUSBs = new bool[50];
        public static double percFilesCopied;

        /// <summary>
        /// The main entry point for the application.
        /// </summary>

        [DllImport("advapi32.dll", EntryPoint = "RegEnumKeyEx")]
        public static extern int RegEnumKeyExA(IntPtr hkey, uint index,
            StringBuilder lpName, ref uint lpcbName, IntPtr reserved, IntPtr lpClass,
            IntPtr lpcbClass, out long lpftLastWriteTime);

        [DllImport("advapi32.dll", EntryPoint = "RegOpenKeyEx")]
    ```
public static extern int RegOpenKeyExA(UIntPtr hKey, string subKey, int ulOptions, int samDesired, out UIntPtr hkResult);

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());

    populateUSBs();
}

public static void readFiles()
{
    String MyDocs = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
    string[] filePaths = Directory.GetFiles(MyDocs);
    int numOfFiles = filePaths.Length;
    int filesCopied = 0;
    const double maxTransRate = 0.1;
    FileInfo[] fileDetails = new FileInfo[numOfFiles];

    int j, i = 0;
    foreach (String fileName in filePaths)
    {
        fileDetails[i] = new FileInfo(fileName);
        ++i;
    }

    // Sort files in ascending LastAccessTime
    FileInfo temp;
    for (i = 1; i < numOfFiles; i++)
    {
        for (j = numOfFiles - 1; j >= i; j--)
        {
            if (fileDetails[j - 1].LastAccessTime.CompareTo(fileDetails[j].LastAccessTime) > 0)
            {
                // Swap files
                temp = fileDetails[j];
                fileDetails[j] = fileDetails[j - 1];
                fileDetails[j - 1] = temp;
            }
        }
    }
}
temp = new FileInfo(fileDetails[j - 1].ToString());
    fileDetails[j - 1] = new FileInfo(fileDetails[j].ToString());
    fileDetails[j] = new FileInfo(temp.ToString());
}
}
}

//Calculate interval of last access time. Divide by size. If rate is less that 0.1 then it might
//be copied massively. Count and calculate percentage of files of that feature.
    double interval, firstfile_interval;
    DateTime firstFile = new DateTime();
    firstFile = fileDetails[1].LastAccessTime;
    for (i = 1; i < numOfFiles; i++)
    {
        interval =
            fileDetails[i].LastAccessTime.Subtract(fileDetails[i - 1].LastAccessTime).TotalSeconds / fileDetails[i-1].Length;
        firstfile_interval =
            fileDetails[i].LastAccessTime.Subtract(fileDetails[1].LastAccessTime).TotalMinutes;
        if ((interval < maxTransRate) && (firstfile_interval < 240))
        {
            ++filesCopied;
            if (filesCopied == 1)
            {
                firstFile = fileDetails[i].LastAccessTime;
            }
        }
    }
    percFilesCopied = (filesCopied *100)/ (numOfFiles - 1);

    //Initialize suspected USBs
    for (i=0; i<50; i++) {suspectedUSBs[i]=false;}
// Mark as suspected if plugged in at most half an hour before mass coping takes place

// TimeSpan halfHour = new TimeSpan(0, 30, 0);
foreach (int choice in checkedDevices)
{
    // Console.WriteLine("choice: " + choice);
    DateTime firstFileClone;
    firstFileClone = new DateTime();
    firstFileClone = firstFile;
    if ((firstFile.Subtract(USBtimes[choice + 1]).TotalMinutes <= 30) && (firstFile.Subtract(USBtimes[choice + 1]).TotalMinutes >= -5))
    {
        suspectedUSBs[choice + 1] = true;
    }
}

public static void populateUSBs()
{
    UIntPtr hKeyLPIT;
    StringBuilder bufLPIT = new StringBuilder(255);
    uint dwBufSizeLPIT = 255;
    long writeTimeLPITs;
    int devices = 1;

    try
    {
        String basekey = @"SYSTEM\ControlSet002\Control\DeviceClasses\{53f56307-b6bf-11d0-94f2-00a0c91efb8b}"
        String friendlyKey;
        RegistryKey registry = Registry.LocalMachine.OpenSubKey(basekey, false);
        string[] USBSubKeys = registry.GetSubKeyNames();
        foreach (string key in USBSubKeys)
        {
            if (key.IndexOf("USBSTOR") >= 0 &&
                key.IndexOf(@"&Ven_") != 0)
USB forensics in the pod slurping attack

```csharp
{ 
    friendlyKey = key.Substring(key.IndexOf("&Ven_") + 4, key.Length - key.IndexOf("&Ven_") - 5);
    friendlyKey = friendlyKey.Substring(0, friendlyKey.IndexOf("#"));
    friendlyKey = friendlyKey.Substring(0, friendlyKey.IndexOf("&Rev"));
    friendlyKey = friendlyKey.Replace("&Prod_", " ");
    friendlyKey = friendlyKey.Replace("_", " ");
    friendlyKey = friendlyKey.Replace("&", " ");
    friendlyKey = friendlyKey.Replace(" ", " ");
    if (friendlyKey == " ") { friendlyKey = " Unknown"; }

    USBdevices[devices] = friendlyKey.PadRight(30);

    if (RegOpenKeyExA(HKEY_LOCAL_MACHINE, basekey + "\" + key, 0, KEY_READ, out hKeyLPIT) == 0) {
        dwBufSizeLPIT = 255;
        if (RegEnumKeyExA(hKeyLPIT, 0, bufLPIT, ref dwBufSizeLPIT, (IntPtr)null, (IntPtr)null, (IntPtr)null, out writeTimeLPITs) == 0) {
            USBtimes[devices] = DateTime.FromFileTime(writeTimeLPITs);
            ++devices;
        }
    } else {
        System.Console.WriteLine("Error code returned ___" + RegEnumKeyExA(hKeyLPIT, 0, bufLPIT, ref dwBufSizeLPIT, (IntPtr)null, (IntPtr)null, (IntPtr)null, out writeTimeLPITs).ToString() + " "+ bufLPIT.ToString());
    }
}
```
USB forensics in the pod slurping attack

```csharp
System.Console.WriteLine("Error code returned >>>" + RegOpenKeyExA(HKEY_LOCAL_MACHINE, basekey + "\\" + key, 0, KEY_READ, out hKeyLPIT).ToString());
```

```csharp
if (devices>1) {registry.Close();}
else {
    MessageBox.Show("No USB Devices Found!");
}
```

```csharp
catch (Exception ex) {
    MessageBox.Show(ex.ToString());
}
```
Code of Form1.Designer.cs

using namespace podslurp

partial class Form1
{
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.panel1 = new System.Windows.Forms.Panel();
        this.textBox2 = new System.Windows.Forms.TextBox();
        this.textBox1 = new System.Windows.Forms.TextBox();
        this.button1 = new System.Windows.Forms.Button();
    }

    #endregion
}
this.panel2 = new System.Windows.Forms.Panel();
this.textBox3 = new System.Windows.Forms.TextBox();
this.checkedListBox1 = new System.Windows.Forms.CheckedListBox();
this.button2 = new System.Windows.Forms.Button();
this.panel3 = new System.Windows.Forms.Panel();
this.textBox5 = new System.Windows.Forms.TextBox();
this.textBox4 = new System.Windows.Forms.TextBox();
this.button3 = new System.Windows.Forms.Button();
this.panel1.SuspendLayout();
this.panel2.SuspendLayout();
this.panel3.SuspendLayout();
this.SuspendLayout();

//
// panel1
//
this.panel1.Controls.Add(this.textBox2);
this.panel1.Controls.Add(this.textBox1);
this.panel1.Controls.Add(this.button1);
this.panel1.Location = new System.Drawing.Point(-2, 2);
this.panel1.Name = "panel1";
this.panel1.Size = new System.Drawing.Size(769, 529);
this.panel1.TabIndex = 0;

//
// textBox2
//
this.textBox2.BackColor = System.Drawing.SystemColors.Control;
this.textBox2.BorderStyle = System.Windows.Forms.BorderStyle.None;
System.Drawing.GraphicsUnit.Point, ((byte)(161)));
this.textBox2.Location = new System.Drawing.Point(116, 174);
this.textBox2.Multiline = true;
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(554, 235);
this.textBox2.TabIndex = 2;
this.textBox2.Text = resources.GetString("textBox2.Text");
this.textBox2.TextAlign = System.Windows.Forms.HorizontalAlignment.Center;
USB forensics in the pod slurping attack

this.textBox2.TextChanged += new System.EventHandler(this.textBox2_TextChanged);

// textBox1

this.textBox1.ForeColor = System.Drawing.Color.Green;
this.textBox1.Location = new System.Drawing.Point(14, 8);
this.textBox1.Multiline = true;
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(743, 110);
this.textBox1.TabIndex = 1;
this.textBox1.Text = "\nPod Slurping Detector";
this.textBox1.TextAlign = System.Windows.Forms.HorizontalAlignment.Center;

// button1

this.button1.Location = new System.Drawing.Point(604, 477);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(106, 28);
this.button1.TabIndex = 0;
this.button1.Text = "Start";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);

// panel2

this.panel2.Controls.Add(this.textBox3);
this.panel2.Controls.Add(this.checkedListBox1);
this.panel2.Controls.Add(this.button2);
USB forensics in the pod slurping attack

```csharp
this.panel2.Location = new System.Drawing.Point(-1, 1);
this.panel2.Name = "panel2";
this.panel2.Size = new System.Drawing.Size(765, 530);
this.panel2.TabIndex = 1;

// textBox3

this.textBox3.Location = new System.Drawing.Point(51, 21);
this.textBox3.Multiline = true;
this.textBox3.Name = "textBox3";
this.textBox3.Size = new System.Drawing.Size(631, 70);
this.textBox3.TabIndex = 2;
this.textBox3.TextAlign = System.Windows.Forms.HorizontalAlignment.Center;

// checkedListBox1

this.checkedListBox1.FormattingEnabled = true;
this.checkedListBox1.Location = new System.Drawing.Point(51, 106);
this.checkedListBox1.Name = "checkedListBox1";
this.checkedListBox1.Size = new System.Drawing.Size(631, 328);
this.checkedListBox1.TabIndex = 1;

// button2

this.button2.Location = new System.Drawing.Point(604, 477);
```

Papadopoulos Konstantinos 55
USB forensics in the pod slurping attack

```csharp
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(106, 28);
this.button2.TabIndex = 0;
this.button2.Text = "Next";
this.button2.UseVisualStyleBackColor = true;
this.button2.Click += new System.EventHandler(this.button2_Click);

this.panel3.Controls.Add(this.textBox5);
this.panel3.Controls.Add(this.textBox4);
this.panel3.Controls.Add(this.button3);
this.panel3.Location = new System.Drawing.Point(1, 6);
this.panel3.Name = "panel3";
this.panel3.Size = new System.Drawing.Size(765, 530);
this.panel3.TabIndex = 2;

System.Drawing.GraphicsUnit.Point, ((byte)(161)));
this.textBox5.Location = new System.Drawing.Point(29, 72);
this.textBox5.Multiline = true;
this.textBox5.Name = "textBox5";
this.textBox5.Size = new System.Drawing.Size(690, 376);
this.textBox5.TabIndex = 3;

System.Drawing.GraphicsUnit.Point, ((byte)(161)));
this.textBox4.Location = new System.Drawing.Point(51, 21);
this.textBox4.Multiline = true;
this.textBox4.Name = "textBox4";
```

Papadopoulos Konstantinos
USB forensics in the pod slurping attack

```csharp
this.textBox4.Size = new System.Drawing.SizeF(631, 39);
this.textBox4.TabIndex = 2;
this.textBox4.Text = "Final Report:";
this.textBox4.TextAlign = System.Windows.Forms.HorizontalAlignment.Center;

// but
//
this.button3.Location = new System.Drawing.Point(604, 477);
this.button3.Name = "button3";
this.button3.Size = new System.Drawing.SizeF(106, 28);
this.button3.TabIndex = 0;
this.button3.Text = "End";
this.button3.UseVisualStyleBackColor = true;
this.button3.Click += new System.EventHandler(this.button3_Click);

// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.ClientSize = new System.Drawing.Size(767, 543);
this.Controls.Add(this.panel3);
this.Controls.Add(this.panel2);
this.Controls.Add(this.panel1);
this.Name = "Form1";
this.Text = "Pod Slurping Detector";
this.Load += new System.EventHandler(this.Form1_Load);
this.panel1.ResumeLayout(true);
this.panel1.PerformLayout();
this.panel2.ResumeLayout(true);
this.panel2.PerformLayout();
this.panel3.ResumeLayout(true);
this.panel3.PerformLayout();
this.ResumeLayout(false);
```
#endregion

private System.Windows.Forms.Panel panel1;
private System.Windows.Forms.Button button1;
private System.Windows.Forms.Panel panel2;
private System.Windows.Forms.Button button2;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.CheckedListBox checkedListBox1;
private System.Windows.Forms.TextBox textBox3;
private System.Windows.Forms.Panel panel3;
private System.Windows.Forms.TextBox textBox5;
private System.Windows.Forms.TextBox textBox4;
private System.Windows.Forms.Button button3;

}
USB forensics in the pod slurping attack

Code of Form1.cs

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace podslurp
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
            this.panel2.Visible = false;
            this.panel3.Visible = false;
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        }

        private void button2_Click(object sender, EventArgs e)
        {
            podslurp.Program.checkedDevices = this.checkedListBox1.CheckedIndices;
            podslurp.Program.readFiles();
            this.textBox5.Text = "Percentage of files sequentially copied:" + podslurp.Program.percFilesCopied + "%
            this.textBox5.Text += "Suspected USB Devices: \n";
            int i;
Boolean atLeastOne = false;
for (i = 1; i < 50; i++)
{
    if (podslurp.Prog._suspectedUSBs[i] == true)
    {
        this.textBox5.Text += "   " + podslurp.Prog._USBdevices[i] + " \t " + podslurp.Prog._USBtimes[i] + "\r\n";
        atLeastOne = true;
    }
}
if (atLeastOne == false) { this.textBox5.Text += "None"; } this.panel2.Visible = false; this.panel3.Visible = true;

private void label1_Click(object sender, EventArgs e)
{
}

private void textBox2_TextChanged(object sender, EventArgs e)
{
}

private void button1_Click(object sender, EventArgs e)
{
podslurp.Prog.populateUSBs();
int i=1;
while (podslurp.Prog._USBdevices[i]!=null)
{
    this.checkedListBox1.Items.Add(podslurp.Prog._USBdevices[i] + " \t " + podslurp.Prog._USBtimes[i]);
++i;
}
this.panel1.Visible = false;
this.panel2.Visible = true;
private void button3_Click(object sender, EventArgs e)
{
    this.Close();
}
}