# Identifying Key Attack Surface Resources with Dynamic Analysis

Frederic Michaud
Thales Canada

Prepared By:
Thales Canada
1405 boul. du Parc-Technologique
Quebec, QC  G1P 4P5

Contractor's Document Number: 2268C.001-REP-01-DA-TA1 Rev. 02
Contract Project Manager: Steeve Cote, 418-651-0606
PWGSC Contract Number: W7714-155991
CSA: Amaya Arcelus, Defence Scientist, 613-993-3831

Contract Report
DRDC-RDDC-2015-C126
June 2015

THALES

# I.T. Security R&D Specialist for the Cyber Capability Development Centre (CCDC)

# Identifying Key Attack Surface Resources with Dynamic Analysis

**Contract No.: W7714-155991**

**Document Control No.: 2268C.001-REP-01-DA-TA1 Rev. 02**

**Date: 23 March 2015**

THALES

# I.T. Security R&D Specialist for the Cyber Capability Development Centre (CCDC)

# Identifying Key Attack Surface Resources with Dynamic Analysis

**Contract No.: W7714-155991**

**Prepared by:**

[illegible]

Frédéric Michaud
Cyber Specialist

**Approved by:**

[illegible]

Steeve Côté
Project Manager

## REVISION HISTORY

| Date | Rev. | Description | Author |
|---|---|---|---|
| 23 March 2015 | 01 | Initial submission | F. Michaud TSCA 004 QA |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## TABLE OF CONTENTS

Page

# LIST OF FIGURES

**Page**

## LIST OF TABLES

# LIST OF ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| **C** | |
| CAPEC | Common Attack Pattern Enumeration and Classification |
| CVE | Common Vulnerabilities and Exposures |
| **E** | |
| ETW | Event Tracing for Windows |

# 1 ON THE CONCEPT OF ATTACK SURFACE

Intuitively, the attack surface of a system is the set of elements an adversary can use to attack a system. The bigger the attack surface, the easier it is to attack a system and do damage. But what kind of element should be considered as being part of the attack surface? What are the relations between these elements? And the most important question: could a surface attack metric really help estimate the security of a software system in practice? Manadhta in (Manadhata & Wing, 2011) tries to answer these questions.

First, it is interesting to consider how far the researchers went in the formalization of their attack surface model. Their complex automaton model is convenient to establish definitions and discuss theoretical properties. However, the usefulness of such an elaborate model for empirical observations is questionable. Section 4 of their article shows a much simpler approach in practice, and even some corner-cutting: for example, how exactly were channels and untrusted data items identified is not told. Hence, it is important to use a model that is workable in practice; otherwise it is just a waste of efforts.

## 1.1 The Two Stages of Compromising a System

Compromising a software system is generally done in two stages. The first stage is to establish a foothold on the system, by exploiting a vulnerability (often remote) that gives the attacker some kind of access. The success of that first stage is closely related to the existence of a vulnerability: without it, the attack cannot succeed. Therefore, an attack surface metric has to take the existence of such vulnerability into account. The existence of vulnerabilities for that first stage is closely related to code quality.

Most of the time, exploiting that first vulnerability is not enough to give the attacker the level of access he needs to complete his mission. In a second stage, he will have to make his way into the system by gaining higher access privilege, until he can steal the information of interest for example. The second stage is very dependent of software design and configuration errors. For example, a vulnerable program running as root (the highest privilege level on UNIX) would give an attacker a very easy time, since the second stage would be unnecessary, the attacker already having all privilege.

## 1.2 Code Quality

The first dimension of an attack surface metric to consider is code quality. Most software vulnerabilities are caused by programming errors and a large part of these errors are a lack of data validation. A good example of this is the well-known buffer overflow vulnerability[1]: the size of the data is not validated against the size of the buffer and it is possible to overwrite memory with code masquerading as data to take control of a process.

Evaluating code quality for a large codebase is really difficult, since the space for potential errors leading to vulnerabilities is extremely large. Examining all possibilities where something bad could happen is simply not calculable. To get an idea of this, consider the Common Weakness Enumeration[2], which is a list of weaknesses, or vulnerability patterns, maintained by Mitre. It contains literally hundreds of vulnerability patterns and some of them are too complex for automated verification. A similar list for attack patterns is the Common Attack Pattern Enumeration and Classification (CAPEC)[3], which shows how vulnerabilities can be exploited to attack a system. Again, there are hundreds of cases to consider. In other words, evaluating code quality directly, from a security point of view, is too difficult to work in practice: anything beyond a primitive analysis is out of reach for the moment.

[1] http://en.wikipedia.org/wiki/Buffer_overflow
[2] http://cwe.mitre.org
[3] https://capec.mitre.org

A better approach would be to indirectly measure code quality by looking at the list of known vulnerabilities for a program. The Common Vulnerabilities and Exposures (CVE)[4] is a list of all known vulnerabilities for all platforms. One can search the CVE to identify past vulnerabilities and get an idea of how vulnerable the software of interest has been. This should be a good estimation of how vulnerable the software still is.

The main problem with the evaluation of code quality for the presence of vulnerabilities is that it will always be incomplete. One could obtain a relatively safe score from a code quality metric, because the software has not been known to be vulnerable in the past and the best programming practices were followed rigorously. However, one morning a security researcher could disclose the existence of a remotely exploitable vulnerability for that software and the code quality metric will not mean much by then. A good example of a mature piece of software that was considered very secure until a remotely exploitable arbitrary code execution vulnerability (the worse kind) was disclosed is OpenSSL with the heartbleed bug[5]. This cryptographic library is used to encrypt communications on the Internet and can be found in web servers, email clients, VPNs, etc. Hence, the impact of the heartbleed bug vulnerability was very important.

An attack surface metric should take into account the code quality and the existence of past vulnerabilities, as they are needed for the first stage of compromising a system.

## 1.3 Damage Potential

The damage potential, as described in Section 3.2 of (Manadhata & Wing, 2011), is strongly related to the second stage of compromising a software system. In essence, it is the assessment of what is possible for an attacker once he has established a foothold on a system by exploiting a first vulnerability.

### 1.3.1 Access Rights and Privileges

Now, it is important to understand the difference between access rights (or permissions) and privileges, including the relation both have with users. A user will need access rights to use a resource. For example, if a user is a member of the “dba” group, he can start a database server process. Being member of the “dba” group is having the access privilege to use the database resource. Once the database server is started, its process will run with all the access rights of the user who started it.

So in essence, access rights and privileges are two sides of the same coin: access rights are the permissions a user needs to use a resource (e.g. starting a process) and privileges are the access rights a process inherits from the user who started it.

### 1.3.2 Effort

Should effort really be considered when assessing damage potential? Again, it is interesting to see the intricate formal definitions of damage potential and effort ratio in Sections 3.2 and 3.3 of (Manadhata & Wing, 2011) that are then severely simplified in empirical calculations of Section 4.2.

The problem with estimating effort is that it is dependent of the level of sophistication of an attacker. A master hacker will be able to execute extremely complex attacks that less gifted individuals would have thought impossible. Besides, complex exploits created by expert hackers, when packaged as simple-to-use tools, tend to migrate to less sophisticated “script kiddies”. These low-level hackers do not understand the inner working of the exploit, but they are able to use the exploit packaged as a tool and then can become a dangerous threat.

[4] http://cve.mitre.org
[5] http://heartbleed.com

A good rule of thumb in cyber security is that if a vulnerability is theoretically exploitable, it will be exploited. This is especially true if the rewards are valuable, like a remotely exploitable arbitrary code execution.

### 1.3.3 Dimensions of Damage Potential

Generally, when an attacker has successfully completed the first stage of compromising a system, he will be in control of a running process. The privilege associated with that process (i.e. the access rights of the user who started the process) will determine what the attacker can do. Of course, there is more to it as the attacker could also use another vulnerability to get higher privileges (privilege escalation[6]). Therefore, it is important to understand that no matter what the estimation of the damage potential is, the unpredicted use of a privilege escalation vulnerability could always completely change the picture for the worse.

Now, what are the dimensions to consider when evaluating the damage potential? Section 3.2 in (Manadhata & Wing, 2011) proposes the following four dimensions: method privilege, channel protocol, data item type, and access rights.

#### 1.3.3.1 Method Privilege

This is the most important dimension, because it determines the privilege that an attacker will inherit when compromising a process. Most of the time, the method privilege will be the same as the process privilege, but this is not always the case.

UNIX –based systems allow a process to downgrade its privilege at runtime, generally after executing sensitive operations when a high privilege is no longer needed. This is done to allow an ordinary user to execute necessary sensitive operations in a controlled way, like changing a password. The programs that change their privilege at runtime are called setuid programs and most of these programs are part of the operating system. Creating setuid programs is strongly discouraged, as it is tricky to implement such a functionally in a secure way that cannot be abused.

Since setuid programs will generally spend a very short time with high privilege, the possibility of taking control of a setuid program before it downgrades its privilege is very slim. Hence, it is recommended to use the process privilege for this measure instead. Considering privilege for methods does not provide significant added value, but require important additional efforts.

#### 1.3.3.2 Channel Protocol

It is true that a channel transport protocol (e.g. TCP, UDP, SSL, Sockets) imposes restrictions on the data exchange allowed using the channel. However, these restrictions are not important enough to be considered, since most network protocols can be used to send raw data by using clever encoding tricks[7]. The damage potential of a channel is only remotely related to the transport protocol. However, the application protocol defined over the transport protocol can have a great impact on the exploitability of a vulnerability. But this impact can only be assessed on a case by case basis, as the assessment is complex and only a deep understanding of the code managing network communications by an analyst can give useful insights.

It is recommended to not take the channel protocol into account for assessing damage potential, as it only has a very small impact. Interestingly, this is exactly what is done in (Manadhata & Wing, 2011): Table 4 shows that all channel types have the same value of 1, which means they have the same impact, or that the impact is not important.

[6] http://en.wikipedia.org/wiki/Privilege_escalation
[7] https://capec.mitre.org/data/definitions/64.html

#### 1.3.3.3 Data Item type

The case for data item type (e.g. file, registry) is very similar to the protocol of a channel: it imposes slight restrictions on the kind of data that can be stored. But again, these restrictions are almost meaningless and it is much more important to analyze how a program uses these data items to assess the damage potential. So it is recommended to not use the data item type as an input to calculate the damage potential.

#### 1.3.3.4 Access Rights

Access rights are the effort side of the damage potential-effort ratio. With method privilege, these are very important as they show what an attacker would be able to do if he had no privilege escalation vulnerability to exploit.

# 2 IDENTIFYING ATTACK SURFACE RESOURCES

The following section describes a few strategies and techniques to identify resources involved in the computation of an attack surface metric, as described in (Manadhata & Wing, 2011). The resources to identify are the entry/exit point methods, the channels, and the untrusted data items. In addition, we have to determine the access rights necessary to access these three sets of resources, and the privilege level that compromising these resources would give to an attacker.

We will try to identify these three kinds of resources with automated tools as much as possible. The objective is to identify the resources from the code or the runtime behavior of the system, and not simply from the documentation or from the opinions of a subject matter expert. In addition to this objective, we would like to be able to work with binary code for which source code is unavailable, as this is the norm. Finally, we are looking for a scalable method that would not only work for an individual program, but also for a whole computer or even an enterprise network.

## 2.1 System Definition and Boundaries

The articles referenced in this task (Manadhata & Wing, 2011; Manadhta & Wing, 2004; Howard, Pincus, & Wing, 2003) have an ambiguous definition of what a system is: it could be a small program made of a single module, a more complex program made of multiple modules such as a web server, a distributed application running on multiple computers, or a whole enterprise network. To keep things simple, we will consider a system as being a process running inside of a computer. Since modern operating systems isolate processes from each other and from the system fairly well, this definition allows us to work with existing system boundaries in a practical way.

Processes inside an operating system are running in their own little sandbox, so that if they malfunction, they cannot affect the other processes too much. The key to this separation is protected memory.

In the early days of computing, every process running in an operating system had the capability to read and write in the whole memory space, including the space used by the operating system and other processes. Of course, this means that programming errors involving errant pointers had the potential to crash the whole computer. To prevent this, protected memory[8] was added to operating systems, and restricted processes memory access to their own memory space only.

[8] http://en.wikipedia.org/wiki/Memory_protection

## 2.2 Entry and Exit Points Methods

The entry and exit points of a process are the methods in its codebase that can interact with the operating system APIs to send or receive information from and to the environment. Should all methods calling APIs be considered entry or exit points? In the case of APIs that interact with the file system, the network, or inter-process communications, it is evident that they should be considered as being entry or exit points. But what about other methods involving USB devices or the GPU for example? Technically, any method that receives data from the environment could be subject to an attack and that would include almost all methods calling APIs. However, to keep things practical, a subset of the most unsafe API calls should be considered and not all of them.

(Manadhata & Wing, 2011) identifies the entry and exit points in a target program by first defining a set of API calls that can send or receive data from and to the environment (let's call them in-out APIs). How exactly this is done, outside of "identifying a set of relevant C library methods", is not explained and it seems that the choice is at least partially arbitrary. They then compute the call graph of the target program using cflow on the source code. This allows them to recognize the methods that call these APIs.

## 2.3 Channels

Channels are communication paths between a process and its environment: anything that could bring data into the memory space of a process is a channel. (Manadhata & Wing, 2011) identifies the channels by "observing the run time behavior of both daemons' default installations". Again, how this was achieved exactly is very vague. There is a clear link between the choice of in-out APIs and what is considered a channel. The process that was followed in (Manadhata & Wing, 2011) seems to be:

(1) Identify manually, from the experience of subject matter experts, the potential channels. The target being IMAP servers, the channels identified were all related to the network: TCP, SSL and Unix sockets. They did not consider inter-process communications mechanisms, such as Linux message queues, as being channels; and

(2) Identify manually the list of APIs that are related to these channels. In the case of the network, anything related to TCP/IP would be of interest.

Choosing the channels arbitrarily has an important impact on the entry/exit points identification later. A better method would identify channels from the code or the runtime behavior of the system.

## 2.4 Untrusted Data Items

Persistent data items are elements such as files, cookies, database records, registry entries, etc. Untrusted data items are persistent data items that are directly accessible via entry/exit points methods. For example, if a method returns the content of a specific file when a webpage is accessed, that would make that file an untrusted data item.

(Manadhata & Wing, 2011) makes a distinction between a data item that is used directly by an entry/exit point method, and a data item used indirectly by other methods. It is not clear why such a distinction should be made, since these persistent data items are not local to the entry/exit point methods. Once a process is compromised, all data items in that process are available to an attacker. Hence, it makes no sense to give more importance to untrusted data items, as all data items can be used equally.

# 3 DYNAMIC ANALYSIS

Dynamic analysis of software is performed by executing a program and observing its behavior. Often, the program or its execution environment will be instrumented with probes, to allow the catching of information at a specific time or place in the program that would not be possible without probes, in the kernel for example.

Compared to other kinds of analysis, on a large software system, dynamic analysis is often the better choice because of its simplicity and scalability. Dynamic analysis does not need the source code, or a thorough understanding of the code. It can even work without any knowledge about the software, being similar to black box testing. The insertion of probes will have an impact on the execution by slowing it down, but modern dynamic analysis tools manage this aspect well and the slowdown is generally not over a 20% penalty.

The main downside of dynamic analysis is that it only sees the parts of a program that are executed. There could be a potential problem in the software that would not be detected because it was never executed during the analysis. This is especially problematic for security analysis, since a single hidden vulnerability can drastically alter the outcome.

Overall, there are two kinds of dynamic analysis tools: snapshot tools and tracing tools. Snapshot tools will take a picture of the state of the program at regular intervals. A good example of a snapshot tool is the Task Manager in Windows: it shows the state of processes and resource usage every five seconds or so. Snapshot tools are relatively lightweight on the execution, but they cannot see what happens between snapshots, which is a severe limitation. On the other hand, tracing tools are like a flight recorder and will accumulate events while the program is running, hence seeing everything. Tracing tools have a much greater impact on the execution speed than snapshot tools.

Tracing tools also have an important limitation, as they cannot work with elements that were created before the recording of events began. This could lead to missing important information in some cases.

For example, let's say we want to detect network sockets that could receive information. Now, a network socket is first created in listening mode, waiting for connections. Once a connection is made, the transfer of data begins. If the recording of events begins after the creation of a listening socket and if no connection occurs during the analysis, the listening socket will not be detected. Hence, for tracing tools the data recording should begin as soon as possible, ideally at boot time. If this is not possible, a combination of snapshot tools and tracing tools could be used: the snapshot tool could obtain the state not available to the tracing tool, such as the list of open listening sockets.

The proposed approach to identify attack surface resources automatically, and in a way that is scalable, is to use dynamic analysis to observe the execution of the system and infer the existence of channels, entry/exit points, and untrusted data items.

# 4 PROCESS MONITOR ON WINDOWS

Process Monitor[9] is a free tool created by the famous Microsoft hacker Mark Russinovich. It is part of the well-known Sysinternals utilities[10] . Process Monitor generates a trace of system events that are related to four classes of resources:

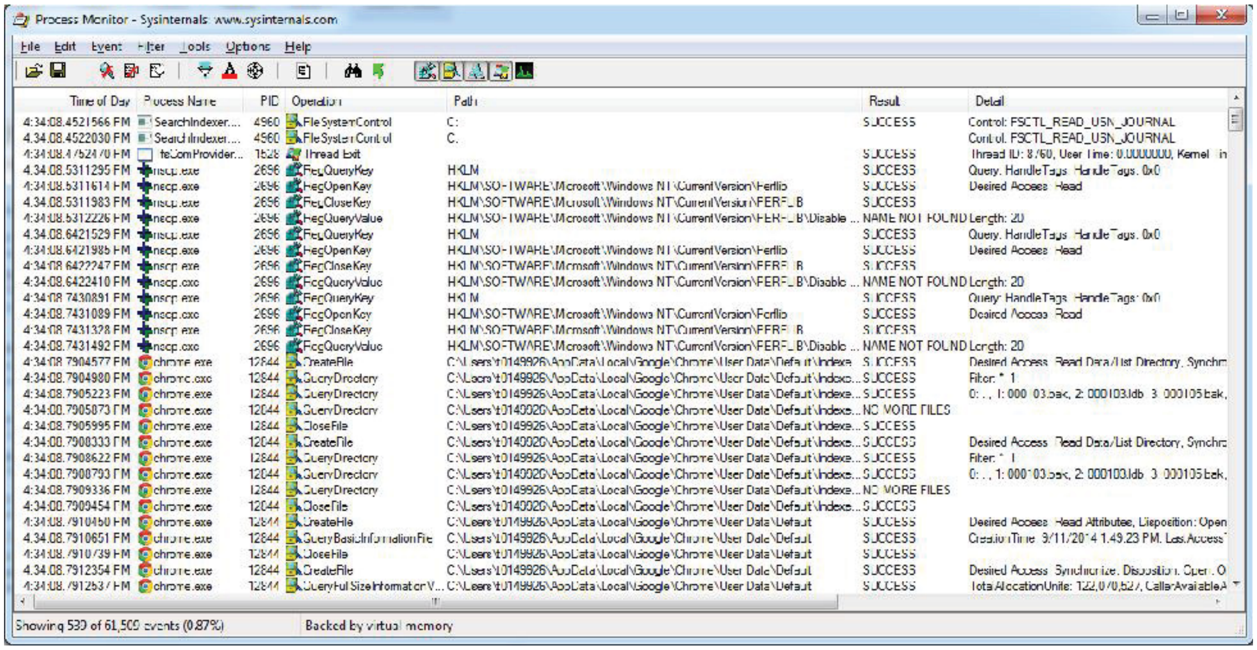
- Registry: all registry operations;
- File system: file system activity for all Windows file systems, local and remote;

[9] https://technet.microsoft.com/en-us/library/bb896645.aspx
[10] https://technet.microsoft.com/en-us/sysinternals/bb545021.aspx

- Network: process monitor records TCP and UDP activity. It does not monitor listening sockets that do not transfer data, which could be a problem; and

- Processes: it tracks all processes and threads creation and exit operations.

Every time a process executes an action that has an impact on these resources, an extensive log entry is generated. Internally, Process Monitor uses both the Event Tracing for Windows (ETW)[11] framework and hooks to collect system events. Process Monitor can also be started at the very beginning of the Windows boot process and start collecting information even before a user session can be opened.



| Time of Day | Process Name | PID | Operation | Path | Result | Detail |
|---|---|---|---|---|---|---|
| 4:34:08.4521566 PM | SearchIndexer.... | 4960 | FileSystemControl | C: | SUCCESS | Control: FSCTL_READ_USN_JOURNAL |
| 4:34:08.4522030 PM | SearchIndexer.... | 4960 | FileSystemControl | C: | | Control: FSCTL_READ_USN_JOURNAL |
| 4:34:08.4752470 PM | teComProvider... | 1528 | Thread Exit | | SUCCESS | Thread ID: 8760, User Time: 0.0000000, Kernel Tim |
| 4:34:08.5311295 PM | nscp.exe | 2696 | RegQueryKey | HKLM | SUCCESS | Query: HandleTags, HandleTags: 0x0 |
| 4:34:08.5311614 PM | nscp.exe | 2696 | RegOpenKey | HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib | SUCCESS | Desired Access: Read |
| 4:34:08.5311983 PM | nscp.exe | 2696 | RegCloseKey | HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\PERFLIB | SUCCESS | |
| 4:34:08.5312226 PM | nscp.exe | 2696 | RegQueryValue | HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\PERFLIB\Disable ... | NAME NOT FOUND | Length: 20 |
| 4:34:08.6421529 PM | nscp.exe | 2696 | RegQueryKey | HKLM | SUCCESS | Query: HandleTags, HandleTags: 0x0 |
| 4:34:08.6421985 PM | nscp.exe | 2696 | RegOpenKey | HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib | SUCCESS | Desired Access: Read |
| 4:34:08.6422247 PM | nscp.exe | 2696 | RegCloseKey | HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\PERFLIB | SUCCESS | |
| 4:34:08.6422410 PM | nscp.exe | 2696 | RegQueryValue | HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\PERFLIB\Disable ... | NAME NOT FOUND | Length: 20 |
| 4:34:08.7430891 PM | nscp.exe | 2696 | RegQueryKey | HKLM | SUCCESS | Query: HandleTags, HandleTags: 0x0 |
| 4:34:08.7431089 PM | nscp.exe | 2696 | RegOpenKey | HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib | SUCCESS | Desired Access: Read |
| 4:34:08.7431328 PM | nscp.exe | 2696 | RegCloseKey | HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\PERFLIB | SUCCESS | |
| 4:34:08.7431492 PM | nscp.exe | 2696 | RegQueryValue | HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\PERFLIB\Disable ... | NAME NOT FOUND | Length: 20 |
| 4:34:08.7904577 PM | chrome.exe | 12844 | CreateFile | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | SUCCESS | Desired Access: Read Data/List Directory, Synchr |
| 4:34:08.7904980 PM | chrome.exe | 12844 | QueryDirectory | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | SUCCESS | Filter: *, 1 |
| 4:34:08.7905223 PM | chrome.exe | 12844 | QueryDirectory | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | SUCCESS | 0: ., 1: 000103.bak, 2: 000103.ldb, 3: 000105.bak, |
| 4:34:08.7905873 PM | chrome.exe | 12844 | QueryDirectory | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | NO MORE FILES | |
| 4:34:08.7905995 PM | chrome.exe | 12844 | CloseFile | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | SUCCESS | |
| 4:34:08.7908333 PM | chrome.exe | 12844 | CreateFile | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | SUCCESS | Desired Access: Read Data/List Directory, Synchr |
| 4:34:08.7908622 PM | chrome.exe | 12844 | QueryDirectory | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | SUCCESS | Filter: *, 1 |
| 4:34:08.7908793 PM | chrome.exe | 12844 | QueryDirectory | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | SUCCESS | 0: ., 1: 000103.bak, 2: 000103.ldb, 3: 000105.bak, |
| 4:34:08.7909336 PM | chrome.exe | 12844 | QueryDirectory | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | NO MORE FILES | |
| 4:34:08.7909454 PM | chrome.exe | 12844 | CloseFile | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default\Indexe... | SUCCESS | |
| 4:34:08.7910450 PM | chrome.exe | 12844 | CreateFile | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default | SUCCESS | Desired Access: Read Attributes, Disposition: Open |
| 4:34:08.7910651 PM | chrome.exe | 12844 | QueryBasicInformationFile | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default | SUCCESS | CreationTime: 9/11/2014 1:49:23 PM, LastAccess |
| 4:34:08.7910739 PM | chrome.exe | 12844 | CloseFile | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default | SUCCESS | |
| 4:34:08.7912354 PM | chrome.exe | 12844 | CreateFile | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default | SUCCESS | Desired Access: Synchronize, Disposition: Open, O |
| 4:34:08.7912537 PM | chrome.exe | 12844 | QueryFullSizeInformationV... | C:\Users\t0149926\AppData\Local\Google\Chrome\User Data\Default | SUCCESS | TotalAllocationUnits: 122,070,527, CallerAvailableA |

Showing 539 of 61,509 events (0.87%) Backed by virtual memory

**Figure 1: Process Monitor on Windows 7**

## 4.1 Identifying Channels with Process Monitor

The network resource watched by Process Monitor already corresponds to the most important channel on the Windows platform. However, there are many inter-process communication mechanisms that have been used to attack Windows processes in the past, such as COM objects, that are not logged by Process Monitor. A dynamic analysis approach will always be limited by the types of event that can be logged by monitoring tools.

### 4.1.1 Channels Related to the Network

Process Monitor creates events for every UDP or TCP activity generated by a process. Here is a sample event logged by Process Monitor on a Windows 7 workstation (Figure 2).

[11] https://msdn.microsoft.com/en-us/library/windows/desktop/bb968803%28v=vs.85%29.aspx

```
<event>
<ProcessIndex>946</ProcessIndex>
<Time_of_Day>4:50:19.3004462 PM</Time_of_Day>
<Process_Name>IEXPLORE.EXE</Process_Name>
<PID>9108</PID>
<Operation>TCP Send</Operation>
<Path>TH3144.THCALJ.CORP:64615 -> 10.94.134.70:8080</Path>
<Result>SUCCESS</Result>
<Detail>Length: 538, startime: 3576884, endtime: 3576885, seqnum: 0,
connid: 0</Detail>
<User>THCALJ\T0149926</User>
<Authentication_ID>00000000:000bc699</Authentication_ID>
<Session>2</Session>
<Integrity>High</Integrity>
<Parent_PID>15368</Parent_PID>
…
```

**Figure 2: Sample Process Monitor Network Event**

We can see that process id 9108 (iexplore.exe - Internet Explorer) sent TCP packets (TCP Send Operation) from the TH3144.THCALJ.CORP IP address on port 64615 to the IP address 10.94.134.70 on port 8080.

Since we want to identify the channels that a hacker could use to attack a process, we are probably not interested by all network communications, but only by those that can be initiated from the attacker. In other words, we are mostly interested by communications that were started by the target process listening on a socket and waiting for a client to connect (inbound communications).

Technically, it is not impossible to use an outbound communication from a process to attack that process. However, that requires an a priori knowledge of the moment when that outbound communication will be initiated and some control over the network to interfere with that communication in a man-in-the-middle style attack. Therefore, using an outbound communication to attack a process is much harder than using an inbound connection and it is recommended that only inbound communications should be taken into account at first.

Identifying inbound connections from a Process Monitor log can be done by looking at the Path element. Process Monitor does an analysis of the flow for every network communication. For example, as is shown in Figure 2, we can clearly see that this is an outbound communication originating from our target computer: TH3144.THCALJ.CORP:64615 -> 10.94.134.70:8080.

By extracting the process id, type of session, and the local port from all receiving events to the computer of interest, one could identify all active channels related to the network.

### 4.1.2 Access Rights

The access rights associated with channels are harder to identify, since we have no information about the TCP or UDP session. If there is an authentication required to use the channel, it is application-specific and it would require an extensive reverse-engineering of the application codebase to assess the existence of authentication mechanisms for every network session observed. It is suggested to use the knowledge about the application to manually specify the access rights related to network channels.

## 4.2 Identifying Untrusted Data Items with Process Monitor

Since data items are not local to a method and could be used anywhere in a compromised process, it is recommended that no distinction should be made between all data items of a process, untrusted or not. Process Monitor can be used to easily identify data items related to a process, as long as these data items belong in the file system or in the registry (which should almost always be the case anyway).

### 4.2.1 Persistent File Data Items

Process Monitor logs many types of operations related to files and only a small subset is of interest to identify persistent file data items: read and write operations. These correspond to the `ReadFile` and `WriteFile` operations in Process Monitor.

```
<event>
<ProcessIndex>1883</ProcessIndex>
<Time_of_Day>2:38:25.5969674 PM</Time_of_Day>
<Process_Name>postgres.exe</Process_Name>
<PID>16624</PID>
<Operation>ReadFile</Operation>
<Path>V:\PostgreSQL\data\postgresql.conf</Path>
<Result>SUCCESS</Result>
<Detail>Offset: 0, Length: 8,192, Priority: Normal</Detail>
<User>NT AUTHORITY\NETWORK SERVICE</User>
<Authentication_ID>00000000:000003e4</Authentication_ID>
<Session>0</Session>
<Integrity>System</Integrity>
<Parent_PID>8656</Parent_PID>
...
```

**Figure 3: Sample Process Monitor Read File Event**

Figure 3 shows that process id 16624, which is a PostgreSQL server (`postgres.exe`), does a ReadFile operation on the file located at the path `V:\PostgreSQL\data\postgresql.conf`. In Figure 4 we can see a similar operation for a file that is written to this time.

```
<event>
<ProcessIndex>1886</ProcessIndex>
<Time_of_Day>2:39:21.7553837 PM</Time_of_Day>
<Process_Name>postgres.exe</Process_Name>
<PID>2352</PID>
<Operation>WriteFile</Operation>
<Path>V:\PostgreSQL\data\pg_log\postgresql-2015-02-
18_143825.log</Path>
<Result>SUCCESS</Result>
<Detail>Offset: 225, Length: 60, I/O Flags: Write Through, Priority:
Normal</Detail>
<User>NT AUTHORITY\NETWORK SERVICE</User>
<Authentication_ID>00000000:000003e4</Authentication_ID>
<Session>0</Session>
<Integrity>System</Integrity>
```

```
<Parent_PID>16624</Parent_PID>
...
```

**Figure 4: Sample Process Monitor Write File Event**

### 4.2.2 Access Rights

Access rights required to access files are not available from Process Monitor and another tool should be used to extract effective permissions on the files identified. A tool such as `AccessChk` could be used, as shown in Figure 5.

```
C:\Program Files Users\SysinternalsSuite>accesschk
V:\PostgreSQL\data\pg_log\postgresql-2015-02-18_143825.log

Accesschk v5.21 - Reports effective permissions for securable objects
Copyright (C) 2006-2014 Mark Russinovich
Sysinternals - www.sysinternals.com

V:\PostgreSQL\data\pg_log\postgresql-2015-02-18_143825.log
  RW Everyone
```

**Figure 5: AcessChk on Files**

### 4.2.3 Persistent Registry Data Items

Identifying persistent registry data items with Process Monitor is very similar to file data items and again, we are only interested in read and write operations. In Process Monitor, these correspond to `RegQueryValue` (Figure 6) and `RegSetValue` (Figure 7).

```
<event>
<ProcessIndex>1895</ProcessIndex>
<Time_of_Day>2:39:19.8813729 PM</Time_of_Day>
<Process_Name>pgAdmin3.exe</Process_Name>
<PID>1908</PID>
<Operation>RegQueryValue</Operation>
<Path>HKLM\System\CurrentControlSet\services\Tcpip\Parameters\Hostname</Path>
<Result>SUCCESS</Result>
<Detail>Type: REG_SZ, Length: 14, Data: TH3144</Detail>
<User>THCALJ\T0149926</User>
<Authentication_ID>00000000:000bc699</Authentication_ID>
<Session>2</Session>
<Integrity>High</Integrity>
<Parent_PID>3908</Parent_PID>
```

**Figure 6: Sample Process Monitor Read Registry Key Value Event**

```
<event>
<ProcessIndex>1895</ProcessIndex>
<Time_of_Day>2:39:19.8817066 PM</Time_of_Day>
<Process_Name>pgAdmin3.exe</Process_Name>
<PID>1908</PID>
<Operation>RegSetValue</Operation>
<Path>HKU\S-1-5-21-3420368142-3320074954-1903974433-
15076\Software\pgAdmin III\Servers\1\Server</Path>
<Result>SUCCESS</Result>
<Detail>Type: REG_SZ, Length: 20, Data: localhost</Detail>
<User>THCALJ\T0149926</User>
<Authentication_ID>00000000:000bc699</Authentication_ID>
<Session>2</Session>
<Integrity>High</Integrity>
<Parent_PID>3908</Parent_PID>
```

**Figure 7: Sample Process Monitor Write Registry Key Value Event**

It is important to only consider successful operations (`<Result>SUCCESS</Result>`), because some programs will test the existence of a registry key by trying to read it, as shown in Figure 8.

```
<event>
<ProcessIndex>1895</ProcessIndex>
<Time_of_Day>2:39:16.4148959 PM</Time_of_Day>
<Process_Name>pgAdmin3.exe</Process_Name>
<PID>1908</PID>
<Operation>RegQueryValue</Operation>
<Path>HKU\S-1-5-21-3420368142-3320074954-1903974433-
15076\Software\pgAdmin III\SuppressGuruHints</Path>
<Result>NAME NOT FOUND</Result>
<Detail>Length: 144</Detail>
<User>THCALJ\T0149926</User>
<Authentication_ID>00000000:000bc699</Authentication_ID>
<Session>2</Session>
<Integrity>High</Integrity>
<Parent_PID>3908</Parent_PID>
```

**Figure 8: Unsuccessful RegQueryValue Operation**

### 4.2.4 Access Right

Again, `AccessChk` can be used to identify the access rights needed to read or write to a specific registry key (Figure 9).

```
accesschk.exe -k
HKLM\System\CurrentControlSet\services\Tcpip\Parameters\Hostname

Accesschk v5.21 - Reports effective permissions for securable objects
Copyright (C) 2006-2014 Mark Russinovich
Sysinternals - www.sysinternals.com

HKLM\System\CurrentControlSet\services\Tcpip\Parameters\Hostname
  R  BUILTIN\Users
     BUILTIN\Administrators
  RW NT AUTHORITY\SYSTEM
  RW NT AUTHORITY\NETWORK SERVICE
  R  NT AUTHORITY\LOCAL SERVICE
  RW BUILTIN\Network Configuration Operators
  RW NT SERVICE\Dhcp
     OWNER RIGHTS
  RW NT SERVICE\WwanSvc
```

**Figure 9: AccessChk on Registry Items**

## 4.3 Identifying Entry and Exit Points with Process Monitor

Here, we have to identify the methods that call in-out APIs. The process to do so suggested in (Manadhata & Wing, 2011) is not applicable in our context, since we are working with binary executables without source code. We could try to replicate the same approach by using disassemblers such as IDA Pro[12] to extract the assembly code from the binaries and then compute the call graphs on assembly code. However, for larger codebases involving many executables and dozens of shared libraries, this would not be practical and would require significant efforts. Again, a simpler approach involving dynamic analysis is suggested.

For each logged event in a process, Process Monitor also extracts the state of the call stack[13] for that process. The call stack shows all method calls from the start of the process until the point when the event was logged. An example of a call stack is shown in Figure 10 for a registry value reading (`RegQueryValue`) that happened in PostgreSQL server.

[12] https://www.hex-rays.com/products/ida
[13] http://en.wikipedia.org/wiki/Call_stack

```
0     ntoskrnl.exe     MmUnmapViewInSessionSpace + 0x7a0
      0xfffff80003423500    C:\Windows\system32\ntoskrnl.exe
1     ntoskrnl.exe     IoRegisterFsRegistrationChangeMountAwareEx +
     0x1bfa6     0xfffff800033bd236    C:\Windows\system32\ntoskrnl.exe
2     ntoskrnl.exe     KeSynchronizeExecution + 0x3a23
      0xfffff80003075153    C:\Windows\system32\ntoskrnl.exe
3     ntdll.dll       NtQueryValueKey + 0xa  0x771b142a
      C:\Windows\System32\ntdll.dll
4     kernel32.dll     LocalAlloc + 0x18c    0x77053e0c
      C:\Windows\System32\kernel32.dll
5     kernel32.dll     RegQueryValueExW + 0xf2     0x77054012
      C:\Windows\System32\kernel32.dll
6     advapi32.dll     RegQueryValueExW + 0x1d     0x7fefe2bf0ed
      C:\Windows\System32\advapi32.dll
7     wxbase28u_vc_custom.dll     wxRegKey::QueryValue + 0x75
      0x7fee7ffd8e5    C:\Program Files\PostgreSQL\9.4\bin\
       wxbase28u_vc_custom.dll
```

**Figure 10: A Sample Call Stack**

Every line shows a method call, the most recent being on the top since this is a stack. Red lines are methods executed in kernel space[14], and blue lines are methods executed in user space. Many APIs called from a program in user space will ultimately be managed by the operating system in kernel space.

The first column identifies the module in which the method call was made, and as one can see, the PostgreSQL Server process (postgre.exe) when loaded in memory is a composite of an executable and many shared libraries, some part of the PostgreSQL package, some being Windows system libraries.

The second column indicates the address of the next instruction to execute upon returning from the sub function (return address). For example, when the call to `MmUnmapViewInSessionSpace in ntoskrnl.exe` will be completed, the execution will return to `IoRegisterFsRegistrationChangeMountAwareEx +` 0x1bfa6 in ntoskrnl.exe (+ 0x1bfa6 is the offset to the specific instruction in that function).

The third column is the absolute return address in the module, a different representation of the same data found in the second column.

The fourth and last column indicates the path in the file system where the module can be found. This path can be used to quickly estimate the origin of the module: if the location points to C:\Windows, it is probably a Windows system library. If however the path points to where the process of interest is installed, as is the case in line 7 of Figure 10 (`C:\Program Files\PostgreSQL\9.4\bin\ wxbase28u_vc_custom.dll`), it is part of the codebase of the monitored process.

Now, a way to identify the in-out methods in the codebase of a process would be to walk the stack from the most recent call until one finds a module that is not a Windows system library. In the case of Figure 10, that would be on line 7: a call to `wxRegKey::QueryValue` in wxbase28u_vc_custom.dll located at C:\Program Files\PostgreSQL\9.4\bin\wxbase28u_vc_custom.dll. Hence, the method wxRegKey::QueryValue is an entry point

[14] http://en.wikipedia.org/wiki/User_space

in the PostgreSQL server on Windows. Figure 11 shows the call stack for a ReadFile operation this time. The method GUC_yyfree in postgres.exe is an entry point.

```
0     fltmgr.sys FltAcquirePushLockShared + 0x907 0xfffff8800107c067
      C:\Windows\system32\drivers\fltmgr.sys
1     fltmgr.sys FltIsCallbackDataDirty + 0x1f3d  0xfffff8800107e82d
      C:\Windows\system32\drivers\fltmgr.sys
2     fltmgr.sys FltDeletePushLock + 0x1e0   0xfffff8800109c630
      C:\Windows\system32\drivers\fltmgr.sys
3     ntoskrnl.exe     NtReadFile + 0x419    0xfffff80003362399
      C:\Windows\system32\ntoskrnl.exe
4     ntoskrnl.exe     KeSynchronizeExecution + 0x3a23
      0xfffff80003075153    C:\Windows\system32\ntoskrnl.exe
5     ntdll.dll  NtReadFile + 0xa 0x771b131a
      C:\Windows\System32\ntdll.dll
6     KernelBase.dll   ReadFile + 0x7a  0x7fefd0c1a7a
      C:\Windows\System32\KernelBase.dll
7     kernel32.dll     ReadFile + 0x59  0x77050a19
      C:\Windows\System32\kernel32.dll
8     msvcr120.dll     read + 0x3eb     0x7fef5969aa7
      C:\Windows\System32\msvcr120.dll
9     msvcr120.dll     fread_nolock_s + 0x18e     0x7fef592f272
      C:\Windows\System32\msvcr120.dll
10    msvcr120.dll     fread_s + 0x7a   0x7fef592f39e
      C:\Windows\System32\msvcr120.dll
11    msvcr120.dll     fread + 0x18     0x7fef592f31c
      C:\Windows\System32\msvcr120.dll
12    postgres.exe     GUC_yyfree + 0x47e    0x14035a50e     C:\Program
     Files\PostgreSQL\9.4\bin\postgres.exe
13    postgres.exe     GUC_yylex + 0x293     0x14035a9c3     C:\Program
     Files\PostgreSQL\9.4\bin\postgres.exe
14    postgres.exe     ParseConfigFp + 0x51a 0x14035005a     C:\Program
     Files\PostgreSQL\9.4\bin\postgres.exe
15    postgres.exe     ProcessConfigFile + 0xdb    0x140350d0b
      C:\Program Files\PostgreSQL\9.4\bin\postgres.exe
16    postgres.exe     SelectConfigFiles + 0x184   0x1403517e4
      C:\Program Files\PostgreSQL\9.4\bin\postgres.exe
17    postgres.exe     PostmasterMain + 0x649      0x1401ed849
      C:\Program Files\PostgreSQL\9.4\bin\postgres.exe
18    postgres.exe     main + 0x2eb     0x14014232b     C:\Program
     Files\PostgreSQL\9.4\bin\postgres.exe
19    postgres.exe     crypt + 0x2327   0x140380bf7     C:\Program
     Files\PostgreSQL\9.4\bin\postgres.exe
20    kernel32.dll     BaseThreadInitThunk + 0xd   0x770559ed
      C:\Windows\System32\kernel32.dll
21    ntdll.dll  RtlUserThreadStart + 0x21   0x7718c541
      C:\Windows\System32\ntdll.dll
```

**Figure 11: Identifying an Entry Point on a ReadFile Operation**

### 4.3.1 Privilege and Access Rights

Unlike Linux, Windows does not have a `setuid()` system call to lower the privilege of a running process and a process will spend its entire lifetime under the privilege it was started with. Hence, to identify the privilege of an entry/exit point method, we can look at the privilege of its running process. An example is shown in Figure 12 for the PostgreSQL database server, where all the entry/exit point methods that can be identified for that process will have the same privilege: NT AUTHORITY\NETWORK SERVICE.

```
<event>
<ProcessIndex>1883</ProcessIndex>
<Time_of_Day>2:38:25.6229234 PM</Time_of_Day>
<Process_Name>postgres.exe</Process_Name>
<PID>16624</PID>
<Operation>ReadFile</Operation>
<Path>C:\Program
Files\PostgreSQL\9.4\share\timezonesets\Default</Path>
<Result>SUCCESS</Result>
<Detail>Offset: 12,288, Length: 4,096</Detail>
<User>NT AUTHORITY\NETWORK SERVICE</User>
<Authentication_ID>00000000:000003e4</Authentication_ID>
<Session>0</Session>
<Integrity>System</Integrity>
<Parent_PID>8656</Parent_PID>
```

**Figure 12: Entry/exit Points Method Privilege**

As for the access rights needed to run the entry/exit point methods, they can be identified by using AccessChk to obtain the permissions associated with the shared library file, as shown in Figure 13.

```
accesschk.exe "C:\Program
Files\PostgreSQL\9.4\bin\wxbase28u_vc_custom.dll"

Accesschk v5.21 - Reports effective permissions for securable objects
Copyright (C) 2006-2014 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Program Files\PostgreSQL\9.4\bin\wxbase28u_vc_custom.dll
  RW NT AUTHORITY\SYSTEM
  RW BUILTIN\Administrators
  R  BUILTIN\Users
```

**Figure 13: AccessChk Checking File Permissions**

## 4.4 Summary

With Process Monitor, we have established a relatively simple and scalable method to identify channels, entry/exit point methods, and data items (untrusted or not) on the Windows platform. Process Monitor also gives us the privilege for entry/exit point methods, and with AccessChk we can get access rights for all data items.

Process Monitor is a nice and easy to use tool, but it has a limited scope: only four classes of events can be monitored, and it is not a scriptable or automatable tool, even if it has a basic command line interface. Using the ETW framework directly could be a less limiting approach, as it is not limited to only four classes of events and it can be used programmatically. For scenarios where one would like to assess the attack surface of a system automatically by installing a software agent, a custom program using the ETW framework would have to be created.

The level of effort required to build a custom program using the ETW framework could be important. The exact capabilities of the ETW framework would have to be assessed, since Process Monitor uses system hooks in addition to the ETW framework to do its monitoring. One would have to make sure that no required functionality identified in Process Monitor depends on these hooks, since they are not part of the ETW framework. If all the functionalities required are part of the ETW framework, the development of such a tool should be relatively straightforward and require around 4 months of work.

## 5 SYSTEMTAP ON LINUX

Sadly, there is no equivalent tool on Linux for Windows Process Monitor that is as easy to use and as powerful. However, there are many lower-level tracing tools similar to Event Tracing for Windows, such as strace[15] , ltrace[16], dtrace[17], LTTng[18], and SystemTap[19].

After reviewing the documentation of these tools, it is clear that many of them could be used to implement the monitoring needed to identify attack surface resources. However, only SystemTap has the right mix of power (anything can be monitored), ease of use (monitoring is done with a simple script), and versatility (the scripting language is almost a full-scale programming language).

SystemTap allows users to write simple scripts for investigating and monitoring a wide variety of kernel functions, system calls, and other events that occur in the kernel space. For example, the script shown in Figure 14 will print the name of the program and its process id every time the system function open is called.

```
probe syscall.open
{
  printf ("%s(%d) open\n", execname(), pid())
}
```

**Figure 14: A Sample SystemTap Script**

A SystemTap script is a mix of probe handlers that will be called each time the event associated with the probe happens, and user defined functions. The user defined functions are used to do custom processing that goes beyond what the probes can do. When called with such a script, SystemTap will do the following behind the scenes:

(1) Analyze the script to identify where the probes should be added in the system;

(2) Translate the script into C;

(3) Compile the script into a loadable kernel module; and

[15] http://en.wikipedia.org/wiki/Strace
[16] http://en.wikipedia.org/wiki/Ltrace
[17] http://en.wikipedia.org/wiki/DTrace
[18] http://lttng.org
[19] https://sourceware.org/systemtap

(4) Load the module and start collecting data.

## 5.1 Installation

SystemTap being mostly a RedHat project, it works well and is easier to install on RedHat-based Linux distributions such as Fedora. Using yum, one can install all the software needed, in addition to the debug symbols necessary to identify where to install the probes (Figure 15).

```
$ yum install gcc systemtap systemtap-runtime systemtap-testsuite
kernel-devel
$ sudo yum --enablerepo=*-debuginfo install kernel-debuginfo kernel-
debuginfo-common
```

**Figure 15: SystemTap Installation Commands**

The documentation about SystemTap is sparse, but the project continues to be actively developed. The current version is 2.7 and the most up to date documentation is the SystemTap Beginners Guide[20]. However, even this guide is incomplete and a good source of information is the large list of examples installed with the software at `/etc/share/systemtap/testsuite/systemtap.examples/`.

SystemTap supports boot-time probing with the onboot command, which would be necessary for whole-computer analysis if one wants to make sure that no event could be missed before the logging mechanism is online. However, even with significant efforts, the author was not able to make it work in the time allowed. The documentation mentions that this functionality can only work on Dracut[21]-based systems, but even on Fedora with Dracut installed it didn't work.

## 5.2 Running SystemTap

A SystemTap script can be launched by using the stap command (Figure 16). The user launching the command must be the superuser, or a member of the stapdev or stapusr groups. Once a script is started, it can be stopped by typing CRTL+C. The data collected by the script is displayed only after a script has finished its execution.

```
[thales@localhost systemtap]$ sudo stap -v test.stp > log.txt
Pass 1: parsed user script and 117 library script(s) using
227396virt/51180res/5732shr/46176data kb, in 130usr/40sys/305real ms.
Pass 2: analyzed script: 10 probe(s), 24 function(s), 78 embed(s), 3 global(s)
using 364724virt/186648res/3740shr/183504data kb, in 1510usr/590sys/4224real
ms.
Pass 3: using cached
/root/.systemtap/cache/f0/stap_f0b44a0519ff7727911bd35bdbb91b62_57257.c
Pass 4: using cached
/root/.systemtap/cache/f0/stap_f0b44a0519ff7727911bd35bdbb91b62_57257.ko
Pass 5: starting run.
^CPass 5: run completed in 0usr/60sys/15623real ms.
```

**Figure 16: Running a SystemTap Script**

[20] https://sourceware.org/systemtap/SystemTap_Beginners_Guide
[21] http://fedoraproject.org/wiki/Dracut

## 5.3 The Complete Script

A SystemTap script to identify channels, data items, and entry/exit point methods all at once was created and is shown in Figure 17. The script outputs data in the CSV format for convenience. It uses 3 global variables (associative arrays[22]) for collecting information: `filehandle` is the list of opened files, `fileresources` is the list of accessed file resources and `networkresources` is the list of incoming network connections. Functions `formatfile` and `formatnetwork` format the output for CSV, and the multiple probes collect information on events to identify channels, data items, and entry/exit point methods.

```
#!/usr/bin/stap

//-----------------------------------------------------------------------------
// Script to extract attack surface resources with SystemTap
//-----------------------------------------------------------------------------

global filehandles
global fileresources
global networkresources

// Returns the path of a previously opened file
function getfdpath:string(pid, fd)
{
      if ([pid, fd] in filehandles)
            return filehandles[pid, fd]
      else
            return sprintf("Unknown file descriptor: %d", fd)
}

// Formats the file resource output
function formatfile:string(fd)
{
      return sprintf("file, %s, %d, %d, %d, %s, %s",
            execname(), pid(), uid(), gid(), usymdata(ustack(0)),
getfdpath(pid(), fd))
}

// Formats the network resource output
function formatnetwork:string(port)
{
      return sprintf("network, %s, %d, %d, %d, %s, %d",
            execname(), pid(), uid(), gid(), usymdata(ustack(0)), port)
}

// Delete global variables in the end
probe end
{
      foreach ([pid, fd] in fileresources)
            printf("%s\n", fileresources[pid, fd])
      foreach ([pid, port] in networkresources)
            printf("%s\n", networkresources[pid, port])
      delete filehandles
      delete fileresources
      delete networkresources
```

[22] https://www.sourceware.org/systemtap/SystemTap_Beginners_Guide/associativearrays.html

```
}

// Registers all file open operations to get the path
probe syscall.open.return
{
  filename = user_string($filename)
  if ($return != -1)
    filehandles[pid(), $return] = filename
}

// File read, write
probe syscall.read.return, syscall.write.return {
      if (!([pid(), $fd] in fileresources))
      {
            fileresources[pid(), $fd] = formatfile($fd)
      }
}

// Network receive
probe kernel.function("tcp_accept").return?,
kernel.function("udp_accept").return?,
kernel.function("inet_csk_accept").return?
{
      sock = $return
      if (sock != 0)
            port = inet_get_local_port(sock)
            if (!([pid(), port] in networkresources))
            {
                  networkresources[pid(), port] = formatnetwork(port)
                  printf("%s\n", formatnetwork(port))
            }
}
```

**Figure 17: The Complete SystemTap Script**

## 5.4 Identifying Channels with SystemTap

Identifying network channels can be done by monitoring incoming connections. There are many ways to do so with SystemTap, since a number of APIs are involved in network communications on Linux. For example, network communications can be probed at the Ethernet device level (layer 2), the TCP/UDP stack level (layers 3), and the socket level (layer 4). To determine the best location in the code to probe for network communications is not trivial and additional experimentations would have to be done to get a definitive answer.

A good API location to probe must be a bottleneck: all network communications for all protocols would have to go through this API. Moreover, this API would need to be at a level where concepts such as an IP address and a listening port are available, which is not the case for lower-level APIs managing Ethernet devices.

Figure 18 shows a way to do so by probing a few kernel functions that are called when a TCP or UDP connection is accepted. Again, it is important to note that only active channels that have transferred data will be detected by probing APIs that accept data connections. We would need to probe for the creation of listening sockets to catch all channels, even inactive. However, the right kernel APIs involved in socket creation has not been identified at this time.

```
probe kernel.function("tcp_accept").return?,
      kernel.function("udp_accept").return?,
      kernel.function("inet_csk_accept").return?
{…}
```

**Figure 18: SystemTap Probes for Monitoring Incoming Connections**

### 5.4.1 Access Rights

Again, it is suggested to use the knowledge about the application to manually specify the access rights related to network channels. However, with a logging framework such as SystemTap, it would theoretically be possible to identify calls to crypto libraries involved in authentication mechanisms. Hence, this would allow the automatic discovery of authentication mechanisms and related access rights.

## 5.5 Identifying Untrusted Data Items with SystemTap

### 5.5.1 Persistent File Data Items

Logging read and write operations on files with SystemTap is very easy: we can monitor calls to the read and write system APIs (Figure 19).

```
probe syscall.read.return, syscall.write.return
{…}
```

**Figure 19: SystemTap Probes for Monitoring Read and Write File Operations**

However, the path of the file being read or written to is not available from these APIs. The read and write system calls use file descriptors to identify files, which are integer indexes on a list of open files. To get the path, one has in addition to probe the file open operation, as shown in Figure 20.

```
probe syscall.open.return
{…]
```

**Figure 20: SystemTap Probe for Monitoring Open File Operation**

By using these two probes, it is possible to monitor read and write operations on files, in addition to identify which file is involved in the operation.

### 5.5.2 Access Rights

File permissions could probably be available directly in SystemTap via a user-defined function, but the author has not found such a function yet. However, it is possible to extract file permissions on Linux with the well-known ls command (Figure 21).

```
$ ls -la /root
total 24
drwxr-xr-x  2 root root 4096 2009-12-16 01:10 .
drwxr-xr-x 23 root root 4096 2010-02-18 10:14 ..
-rw-------  1 root root  123 2010-01-21 15:49 .bash_history
-rw-r--r--  1 root root 2227 2007-10-20 11:51 .bashrc
-rw-r--r--  1 root root  141 2007-10-20 11:51 .profile
-rw-------  1 root root  868 2009-12-16 00:47 .viminfo
```

**Figure 21: The ls Command to Extract File Permissions**

## 5.6 Identifying Entry and Exit Points with SystemTap

Figure 22 shows the output of the script, which is formatted with the following columns:

[type of resource], [executable name], [process id], [user id], [group id], [entry/exit point], …

- Type of resource: file or network;
- Executable name: the name of the executable, often the command that started the process;
- Process id: the process identifier;
- User id: the user that started the process;
- Group id: the primary group of the user running the process;
- The entry/exit point function (only an approximation for the moment, see below); and
- Additional information related to the type of the resource. For files, the full path of the file. For network resources, the local port.

Identifying the entry/exit point function is achieved by calling `usymdata(ustack(0))`. First, the call to `ustack(0)` returns the top function on the user stack. Now, this location is only an approximation of the real entry/exit point and most of the time it will point to a shared library running in the userspace context of the process (e.g. libc). As was shown with Process Monitor, we would have to walk the stack back until we identify a code module that is part of the program of interest and not from the system. Any piece of code in `/usr/`* is a system shared library. There may be a bug in SystemTap `ustack`() function, since only the last two elements were available most of the time. Hence, it was impossible to walk the stack until the real entry/exit point was found. This will have to be further investigated. The `usymdata()` function returns the debug information related to the address , such as [`/usr/lib64/libpthread-2.20.so+0xf400/0x21c000]`.

```
network, python, 2799, 1000, 1000, 0x3d1500f400 [/usr/lib64/libpthread-
      2.20.so+0xf400/0x21c000], 8000
file, systemd-udevd, 528, 0, 0, 0x7f6226426e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3b9000], /sys/fs/cgroup/systemd/system.slice/systemd-
      udevd.service/cgroup.procs
file, systemd-journal, 480, 0, 0, 0x7f5b43afde50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3b9000], /proc/1602/cgroup
file, systemd, 1, 0, 0, 0x7f1666956e6d [/usr/lib64/libc-
      2.20.so+0xf0e6d/0x3b9000], /proc/480/cgroup
file, vmtoolsd, 634, 0, 0, 0x7f82dcbe0e6d [/usr/lib64/libc-
      2.20.so+0xf0e6d/0x3b9000], /proc/devices
file, vmtoolsd, 634, 0, 0, 0x7f82dcbe0e50 [/usr/lib64/libc-
```

```
      2.20.so+0xf0e50/0x3b9000], /etc/mtab
file, vmtoolsd, 634, 0, 0, 0x7f82dcbe0e6d [/usr/lib64/libc-
      2.20.so+0xf0e6d/0x3b9000], /proc/net/if_inet6
file, sedispatch, 588, 0, 0, 0x3d154f0e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3b9000], Unknown file descriptor: 0
file, systemd-cgroups, 2798, 0, 0, 0x7f0b56722757, /lib64/librt.so.1
file, systemd, 1, 0, 0, 0x7f1666c321cd [/usr/lib64/libpthread-
      2.20.so+0xf1cd/0x218000], /dev/urandom
file, systemd, 1, 0, 0, 0x7f1666956e6d [/usr/lib64/libc-
      2.20.so+0xf0e6d/0x3b9000], /sys/fs/cgroup/systemd/system.slice/avahi-
      daemon.service/cgroup.procs
file, python, 2799, 1000, 1000, 0x3d154f0e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3bd000], /usr/lib64/python2.7/site.pyc
file, python, 2799, 1000, 1000, 0x3d154f0e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3bd000], /usr/lib64/python2.7/os.pyc
file, python, 2799, 1000, 1000, 0x3d154f0e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3bd000], /usr/lib64/python2.7/posixpath.pyc
file, python, 2799, 1000, 1000, 0x3d154f0e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3bd000], /usr/lib64/python2.7/stat.pyc
file, python, 2799, 1000, 1000, 0x3d154f0e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3bd000], /usr/lib64/python2.7/linecache.pyc
file, python, 2799, 1000, 1000, 0x3d154f0e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3bd000], /usr/lib64/python2.7/_weakrefset.pyc
file, DNS Resolver #1, 2225, 1000, 1000, 0x3d154f0e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3b9000], /etc/hosts
file, Cache2 I/O, 2225, 1000, 1000, 0x3d1500f1cd [/usr/lib64/libpthread-
      2.20.so+0xf1cd/0x218000],
      /home/thales/.cache/mozilla/firefox/ncsah2hs.default/cache2/entries/884
      B4EDE90824D38033103F4018788CE49EB37BE
file, crond, 1137, 0, 0, 0x7f4ca0142e50 [/usr/lib64/libc-
      2.20.so+0xf0e50/0x3b9000], /etc/passwd
file, gvfsd-metadata, 2042, 1000, 1000, 0x3d1500f16d [/usr/lib64/libpthread-
      2.20.so+0xf16d/0x218000], /home/thales/.local/share/gvfs-
      metadata/home.NBC1UX
file, pool, 2194, 1000, 1000, 0x3d154f0e6d [/usr/lib64/libc-
      2.20.so+0xf0e6d/0x3b9000], /proc/self/task/2800/attr/current
file, gnome-shell, 1755, 1000, 1000, 0x3d154f0e6d [/usr/lib64/libc-
      2.20.so+0xf0e6d/0x3b9000], /proc/self/stat
network, python, 2799, 1000, 1000, 0x3d1500f400 [/usr/lib64/libpthread-
      2.20.so+0xf400/0x21c000], 8000
```

**Figure 22: Log of SystemTap Script**

The output of the script in Figure 22 shows a Linux system executing for a few seconds. To get more interesting results containing network events, a simple http server was created with Python by launching the command shown in Figure 23. A web browser was then used to access the web server twice (first and last event of Figure 22).

```
$ python –m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 …
127.0.0.1 - - [06/Mar/2015 14 :56 :00] GET / http/1.1 200 –
127.0.0.1 - - [06/Mar/2015 14 :56 :00] GET / http/1.1 200 -
```

**Figure 23: Python SimpleHTTPServer**

## 6 WINDOWS COMPUTER DEFINITION

The following section presents an attempt to apply the technique previously shown to capture key attack surface resources on a Windows 7 desktop computer. The computer used for this experiment was a workstation on the Thales corporate network, a realistic setup. Process Monitor was set to "Enable Boot Logging" and the computer was restarted. Then, a user session was opened, Process Monitor was started (which stopped logging) and the event log was saved to disk.

Now, using dynamic analysis for a whole computer generates a lot of data. The raw data captured was around 2.3 GB for 4 612 592 events, and this is only for around five minutes of execution. However, Process Monitor handles event captures of this size without any problem.

It is important to appreciate the size and complexity of such an analysis for a whole computer. The size of the data itself requires the use of tools like databases, since it is impossible to work on such a big dataset manually (forget about Excel). The author of this document suggests SQLite[23], a small and free database. Presenting the results is also a challenge, because there are literally thousands of resources involved in the attack surface. It is clear that such an approach could only be practical if it is fully automated. However, automating such an approach should be relatively easy, since it is mostly integration and database management work.

### 6.1 Identifying Network Channels

Process Monitor failed to capture any network event during this experiment, even after multiple attempts. It seems that the Windows 7 workstation on the Thales corporate network had a special Cisco VPN software installed that interfered with standard network functions. Hence, the network APIs monitored by Process Monitor were not used anymore and were replaced by custom Cisco drivers and APIs.

This shows one limitation of dynamic analysis: if an operating system is modified by the installation of software that makes changes in the inner workings of the system, monitored location may not be available or work as expected anymore. This is especially true for security software such as an antivirus, that have to install themselves deep into the operating system to monitor and protect sensitive low-level functions in the kernel.

### 6.2 Identifying File Data Items

There were 90 652 events of the type `ReadFile` or `WriteFile`, but these events included access to shared libraries (.dll) and executable files (.exe). It seems that Process Monitor captures the action of reading a file to load executable code in memory when launching a process. Should this information be included in our analysis? Yes, because clearly it is possible to attack a system by replacing code modules that will be executed; it fits the definition of what a data item is in (Manadhata & Wing, 2011).

[23] http://www.sqlite.org

Many of these files were accessed multiple times and when the data was grouped by process and path, there were still 7825 unique events. In these 7825 events, there were 6244 unique files accessed by 105 processes (some processes accessed the same files) and these processes were started by 5 different users.

Table 1 shows the list of processes accessing file data items: for each entry there is the process name and the number of individual file data items that were read from or written to.

**Table 1: The List of Processes and the Number of Files Data Items Accessed**

| | | | |
|---|---|---|---|
| System | 3407 | RunDll32.exe | 16 |
| svchost.exe | 838 | shstat.exe | 16 |
| Explorer.EXE | 579 | vpncli.exe | 16 |
| OUTLOOK.EXE | 223 | GoogleUpdate.exe | 15 |
| SearchIndexer.exe | 186 | igfxsrvc.exe | 15 |
| csrss.exe | 169 | mfeann.exe | 15 |
| services.exe | 148 | FMAPP.exe | 14 |
| smss.exe | 102 | TGitCache.exe | 14 |
| vpnagent.exe | 89 | dcusb64.exe | 14 |
| nvSCPAPISvr.exe | 84 | dcswmeter.exe | 13 |
| FrameworkService.exe | 78 | dcuninstallsw.exe | 12 |
| spoolsv.exe | 77 | naPrdMgr.exe | 12 |
| wininit.exe | 75 | rundll32.exe | 12 |
| LogonUI.exe | 74 | Reader_sl.exe | 11 |
| lsass.exe | 74 | WUDFHost.exe | 11 |
| SearchProtocolHost.exe | 70 | CloseVPN.exe | 10 |
| taskhost.exe | 67 | dcinventory.exe | 10 |
| Eraser.exe | 65 | dcondemand.exe | 10 |
| TfsComProviderSvr.exe | 54 | dcusbsummary.exe | 10 |
| vmware-authd.exe | 54 | vmware-usbarbitrator64.exe | 10 |
| vpnui.exe | 51 | OSPPSVC.EXE | 9 |
| cmd.exe | 47 | gpscript.exe | 9 |
| dcagenttrayicon.exe | 47 | igfxpers.exe | 9 |
| dcconfig.exe | 46 | wscript.exe | 8 |
| DllHost.exe | 41 | Check-Intranet.exe | 7 |
| wmiprvse.exe | 40 | RAVBg64.exe | 7 |
| WScript.exe | 36 | lsm.exe | 7 |
| nvvsvc.exe | 34 | mfevtps.exe | 7 |
| MSOSYNC.EXE | 33 | regedit.exe | 7 |
| easynetswitch.exe | 33 | taskeng.exe | 7 |
| runonce.exe | 29 | RAVCpl64.exe | 6 |
| ONENOTEM.EXE | 28 | armsvc.exe | 6 |
| nvxdsync.exe | 26 | cscript.exe | 6 |
| winlogon.exe | 26 | emedtray.exe | 6 |
| SearchFilterHost.exe | 23 | sdclt.exe | 6 |
| dcstatusutil.exe | 23 | atbroker.exe | 5 |
| explorer.exe | 23 | gpupdate.exe | 5 |
| dcagentservice.exe | 22 | hkcmd.exe | 5 |
| Check-Intranet-Boot.exe | 21 | iusb3mon.exe | 5 |
| Dwm.exe | 21 | vmnat.exe | 5 |
| VsTskMgr.exe | 21 | GoogleCrashHandler64.exe | 4 |
| enssvc.exe | 21 | ipconfig.exe | 4 |
| jusched.exe | 21 | userinit.exe | 4 |

| | | | |
|---|---|---|---|
| McTray.exe | 20 | vmnetdhcp.exe | 4 |
| AUDIODG.EXE | 18 | GoogleCrashHandler.exe | 3 |
| MmReminderService.exe | 18 | RAServer.exe | 3 |
| UdaterUI.exe | 18 | Set-English.exe | 3 |
| MfeServiceMgr.exe | 17 | ibmpmsvc.exe | 3 |
| OcsService.exe | 17 | whoami.exe | 3 |
| PrintIsolationHost.exe | 17 | nscp.exe | 2 |
| communicator.exe | 17 | regini.exe | 2 |
| conhost.exe | 17 | autochk.exe | 1 |
| AdobeARM.exe | 16 | | |

## 6.3 Identifying Entry and Exit Point Methods

There is clearly too much data to present the entry/exit point methods analysis for all data items. Hence, a single process was chosen to continue the analysis: MfeServiceMgr.exe, which is a component of the McAfee antivirus (McAfee Service Manager). Table 2 shows the list of file data items for the McAfee Service Manager.

**Table 2: File Data Items for McAfee Service Manager**

| Process | Operation | Path | User |
|---|---|---|---|
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\0409\AgentRes.Dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\AppLib.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\Dispatcher.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\MfeServiceMgr.exe | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\cryptshim.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\mfeCmnLib71.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\mfecryptc.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\mfediscovery.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\msvcp100.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\msvcr100.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\naCmnLib3_71.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\nailog3.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Program Files (x86)\McAfee\Common Framework\naxml3_71.dll | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\ProgramData\McAfee\Common Framework\Mesh\SvcMgr_TH3144.log | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\ProgramData\McAfee\Common Framework\Mesh\SvcMgr_TH3144_error.log | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\ProgramData\NVIDIA Corporation\Drs\nvdrssel.bin | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | ReadFile | C:\Windows\Prefetch\MFESERVICEMGR.EXE-9A98F1C0.pf | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | WriteFile | C:\ProgramData\McAfee\Common Framework\Mesh\SvcMgr_TH3144.log | NT AUTHORITY\SYSTEM |
| MfeServiceMgr.exe | WriteFile | C:\ProgramData\McAfee\Common | NT |

| | | | |
|---|---|---|---|
| | | Framework\Mesh\SvcMgr_TH3144_error.log | AUTHORITY\SYSTEM |

For each data item presented in Table 2, the stack was walked back until a call in user space was found. Since we are doing the analysis for the whole computer (we were not interested in only one specific process), system shared libraries were included in the analysis. Table 3 shows the entry/exit point methods for MfeServiceMgr.exe: there are only 3 entries, as the same function is used to read or write different files.

**Table 3: Entry/ExitPoint Methods for McAfee Service Manager**

| Component | Method |
|---|---|
| `C:\Windows\System32\ntdll.dll` | `RtlUserThreadStart` |
| `C:\Windows\SysWOW64\ntdll.dll` | `LdrInitializeThunk + 0x10` |
| `C:\Program Files (x86)\McAfee\Common Framework\MfeServiceMgr.exe` | `mcafee_com::cma::servicemgr::ServiceMgr::~ServiceMgr + 0x664b` |

### 6.3.1 Access Rights

To be able to call these methods, a user will need the execute permission on these components. Figure 24 shows `accesschk` being used to extract the permissions on `ntdll.dll`. It is important to call `accesschk` with the –l parameter to show the execute permission, otherwise a simplified list of permissions without `execute` is shown.

```
C:\Program Files Users\SysinternalsSuite>accesschk.exe -l
C:\Windows\System32\ntdll.dll

Accesschk v5.21 - Reports effective permissions for securable objects
Copyright (C) 2006-2014 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Windows\System32\ntdll.dll
  DESCRIPTOR FLAGS:
      [SE_DACL_PRESENT]
      [SE_SACL_PRESENT]
      [SE_DACL_PROTECTED]
      [SE_SACL_PROTECTED]
      [SE_RM_CONTROL_VALID]
  OWNER: NT SERVICE\TrustedInstaller
  [0] ACCESS_ALLOWED_ACE_TYPE: NT SERVICE\TrustedInstaller
        FILE_ALL_ACCESS
  [1] ACCESS_ALLOWED_ACE_TYPE: BUILTIN\Administrators
        FILE_EXECUTE
        FILE_READ_ATTRIBUTES
        FILE_READ_DATA
        FILE_READ_EA
        SYNCHRONIZE
        READ_CONTROL
  [2] ACCESS_ALLOWED_ACE_TYPE: NT AUTHORITY\SYSTEM
        FILE_EXECUTE
        FILE_READ_ATTRIBUTES
        FILE_READ_DATA
        FILE_READ_EA
        SYNCHRONIZE
        READ_CONTROL
  [3] ACCESS_ALLOWED_ACE_TYPE: BUILTIN\Users
        FILE_EXECUTE
        FILE_READ_ATTRIBUTES
        FILE_READ_DATA
        FILE_READ_EA
        SYNCHRONIZE
        READ_CONTROL
```

**Figure 24: Permissions for ntdll.dll**

The permissions model on Windows uses access control lists (ACL) [24], which is different from the "owner-group-world" permission model used on Linux. The key difference between these models is the number of entries possible for a file permission. On Linux, a file permission can only be set for one user or group, but on Windows there is a variable list involved that can contain more than one user or group. The ACL approach is more versatile, but can also be much more complex if not used with moderation (i.e. large lists of users for a file permission). Table 4 shows the access rights needed to execute the entry/exit point methods found in the MfeServiceMgr.exe process modules.

**Table 4: Windows File Data Items and Access Rights**

| File Data Item | Access Right for Execute Permission |
|---|---|
| C:\Windows\System32\ntdll.dll | BUILTIN\Administrators |
| C:\Windows\System32\ntdll.dll | NT AUTHORITY\SYSTEM |
| C:\Windows\System32\ntdll.dll | BUILTIN\Users |
| C:\Windows\SysWOW64\ntdll.dll | BUILTIN\Administrators |
| C:\Windows\SysWOW64\ntdll.dll | NT AUTHORITY\SYSTEM |
| C:\Windows\SysWOW64\ntdll.dll | BUILTIN\Users |
| C:\Program Files (x86)\McAfee\Common Framework\MfeServiceMgr.exe | Everyone |

## 6.4 Privilege

The MfeServiceMgr.exe process ran under the `NT AUTHORITY\SYSTEM` privilege and all entry/exit point methods inherit this privilege.

Assigning Numeric Values

As shown in section 4.2 of (Manadhata & Wing, 2011), assigning numeric values to estimate the damage potential related to a privilege cannot easily be automated. However, establishing a total order between them is much easier and recommended as the first stage of this analysis.

[24] http://en.wikipedia.org/wiki/Access_control_list

On UNIX, there is only one built-in account: the superuser (`root`). The superuser can access everything and has all privilege. Then, there are user accounts with varying level of privilege. These user accounts are related to installed applications and are given the necessary privilege required for the application to function (e.g. a postgre account to run the PostgreSQL database). Hence, this must be analyzed on a case by case basis and there is no generic answer beyond a primitive order such as:

```
[superuser] > [authenticated users] > [unauthenticated users]
```

On Windows things are much different, since there are many built-in accounts and groups, in addition to a very large list of access rights that goes way beyond file system security. For example, the Windows Service Control Manager (SCM), which is the module that manages services (daemons) in Windows, has a specific list of access rights shown in Table 5.

**Table 5: Access Rights for the Windows Service Control Manager**

| Access Right | Description |
|---|---|
| SC_MANAGER_ALL_ACCESS | Includes STANDARD_RIGHTS_REQUIRED, in addition to all access rights in this table. |
| SC_MANAGER_CREATE_SERVICE | Required to call the CreateService function to create a service object and add it to the database. |
| SC_MANAGER_CONNECT | Required to connect to the service control manager. |
| SC_MANAGER_ENUMERATE_SERVICE | Required to call the EnumServicesStatus or EnumServicesStatusEx function to list the services that are in the database.<br>Required to call the NotifyServiceStatusChange function to receive notification when any service is created or deleted. |
| SC_MANAGER_LOCK | Required to call the LockServiceDatabase function to acquire a lock on the database. |
| SC_MANAGER_MODIFY_BOOT_CONFIG | Required to call the NotifyBootConfigStatus function. |
| SC_MANAGER_QUERY_LOCK_STATUS | Required to call the QueryServiceLockStatus function to retrieve the lock status information for the database. |

Many built-in users and groups have default access rights for specific OS functions. Moreover, access rights can contain other access rights, groups can contain other groups, and in the end having a clear picture of the exact impacts on security is difficult. The UNIX security model is much simpler and easier to manage.

Without trying to assess the effect on security that each access right or group membership can have (that would be a huge task), there are a few common users and groups that should be mentioned.

- **Administrators Group:** any user member of this group has full administrative rights, a close equivalent to the superuser in UNIX.

- **Users Group**: a member of this group inherits standard access rights for all users.

- **LocalSystem:** this is a special account that has extensive privilege on the computer and is used to run many Windows services. This is the true equivalent to the superuser in UNIX.

- **NetworkService:** this account is also used to run Windows services, but it has less privilege on the local computer than LocalSystem. This account can access the network.

- **LocalService:** this is a third account used to run Windows services and it has the same level of privilege on the computer as NetworkService. The difference is that it cannot access the network.

Now, we can establish an order between these users and groups in term of access rights and potential damage.

(1) The LocalSystem account, having complete and unrestricted access to the computer, is the most dangerous account;

(2) The Administrators group is a close second, but it cannot do some tasks such as closing a system process (LocalSystem can do that);

(3) The Users group has normal access to the computer;

(4) NetworkService has limited access to the computer, but can access the network; and

(5) LocalService also has limited access to the computer, and in addition cannot access the network.

## 6.5 Conclusion

Discovering resources involved in the computation of an attack surface metric, as described in (Manadhata & Wing, 2011), but for a whole computer, is challenging to say the least. This document showed that we can expect a computer to have at least thousands of resources, of which the large majority will be data items. Because of this, it is almost impossible to identify all these resources without the help of automation.

This document also presented a relatively easy to use technique for discovering resources using dynamic analysis. The main advantage of this technique is that it does not need the source code of a software system to work. The second, non-negligible aspect is the scalability of this technique: it would be entirely feasible to use it for a whole network of computers. Dynamic analysis also has downsides, the most important being that this analysis technique only "sees" what is executed. Attack surface resources not involved in an observed execution would not be detected with dynamic analysis.

The security model of Windows is much more complex than the one used in Linux, or UNIX in general. The complexity arises in part from the very large list of access rights that can be used to manage the security in Windows. That list goes above and beyond file system security and a clear mapping of the access rights and their associated impact on security would have to be assessed. No such list has been found yet and a complete assessment is probably necessary.

Some characteristics of the model shown in (Manadhata & Wing, 2011) to do empirical attack surface calculations could be simplified. First, considering privilege for methods instead of processes is not necessary. Except for setuid programs on UNIX, all methods run using the same privilege as the process. Besides, no distinction should be made between trusted and untrusted data items. Data items such as files are not local to a method and an attacker could use any data items belonging to a process. Hence, that distinction is useless. In fact, considering methods at all is not recommended and an interesting simplification would be to consider modules (executables and shared libraries) instead. Finally, the type of a data item should not be considered at all, because the constraints on the kind of data that could be stored in it (e.g. executable code) are trivial to bypass using encoding tricks. Hence, a data item of a specific type (e.g. file) is not more secure than another of a different type (e.g. registry).

In the end, the author of this document wants to make the case that the attack surface of a software system is not strongly related to the type of software. For example, two web server programs could have very different attack surfaces. Server A could be a simple and limited web server, such as lighttpd[25],, and server B could be a fully-featured Apache or IIS server with tons of configuration files, a remote administration console, support for extensions and scripting, etc. Hence, these two web servers would have drastically different attack surfaces. The attack surface really depends on the implementation and must be assessed on a case-by-case basis.

[25] http://www.lighttpd.net

## **References**

*Howard, M., Pincus, J., & Wing, J. (2003). Measuring relative attack surfaces. Workshop on Advanced Developments in Software and Systems Security.*

*Manadhata, P. K., & Wing, J. M. (2011). An Attack Surface Metric. IEEE Transactions on Software Engineering, vol. 37, no. 3, 371-386.*

*Manadhta, P. K., & Wing, J. M. (2004). Measuring a system's attack surface. CMU-CS-04-102.*