Global Information Assurance Certification Paper

Passing the Sniff (Snort) Test

GIAC (GCIA) Gold Certification

Author: Matthew Hansen, matthew.j.hansen@hotmail.com
Advisor: Chris Walker
Accepted: September 30th 2015

Template Version September 2014

Abstract

They go by several names: Bloatware. Trialware. Pre-installation-ware. Some of them are completely innocuous. Many are designed to automate harvesting of information from the user. The line between these "unwantedware" and malware is thinning. Whether they arrive in our networks from a less-than-perfect supply chain, or as a natural result from Bring-Your-Own-Device (BYOD) policies, or even as an aggressive customer support "service" from the manufacturer, unwantedware shall exist. On the best of days, network defenders will identify, mitigate, and remove said software from their organization in the hopes that it cannot come back. Unfortunately, these herculean efforts are not enough. Users will ignore warnings from the security administrators. Users will pay lip service to the security training their organization provides. Users will rationalize intrusions into their devices through a myriad of worthless excuses: "I'm really boring", or "Anyone who wants to spy on me will have a lot of nothing to do", or "I'm really ugly, turning on my webcam would hurt THEM." Time and again users have shown that they are incapable of understanding the risks involved, they must be trained to dislike being spied on. In this paper we will examine unwanted data exfiltrations initiated by software we are told to trust, be it prepackaged software, chatty smartphone apps, or smart television applications. We will also present methods for detecting said exfiltrations, determining what data is being sent, and alerting the user in a meaningful way.

1. Introduction

It was a simple process, really. Open the Android App store. Get an app that provides a local shell – bash, csh, even the draconian syntax rules from cmd.exe would be acceptable at this point. Then get the job done. The first app needs quite a few access rights that don't seem to line up with what a shell does - Full Internet Access, the unique keys associated with the phone, Start app on boot, Browser History, Device to Cloud communication and more. The rest were not much better, but the glorious ls command as written by Richard Stallman was required for the task at hand. Within a few minutes the user was clicking through advertisements just so that *he could view his own directory structure on his own phone on a platform made possible only by the ideals of open source and the GPL.*

Viewing advertisements isn't the problem, though. The problem is that advertisements, like other undesired processes, may (although they certainly are not required to) run external applications, observe and record user actions, generate web traffic, and report data in real time so as to generate a more perfect advertisement – one that is likely to result in a sale. (Kaspersky, n.d.) Sales are not necessarily the sole goal of such tools, however, as recorded usage habits of a user are directly marketable by themselves. Even large, established corporations are not morally above covertly spying on their own customers. In late 2014, "…Lenovo installed a self-signing root certificate authority (on Lenovo-branded devices) that has the capability to intercept and hijack internet traffic flowing over SSL and TLS connections -- often used by online stores, banks, and other apps and services to secure send data…" for the explicit purpose of gaining advertisement revenue using the Superfish adware program. (Whittaker, 2015, para. 7) Several months later, Lenovo began using malware techniques such as infecting a system BIOS to maintain unwantedware presence on Lenovo-branded devices. (Whittaker, 2015, para. 1,4) Even security vendors have been corrupted into the spying market, as "…security firm AVG [has recently decided that it] can sell search and browser history data to advertisers in order to make money from its free antivirus software." (Temperton, 2015, para. 1)

Most people don't seem to get upset by this kind of treatment by information technology vendors. The notion of non-identifiable information about us being listed as yet another record or series of records in a veritable sea of the same is quite tolerable. The thought of aggregation of non-identifiable information about us to the point where identity is without question is another thing altogether. As consumers, we rarely see any warnings about the reality of how much information is *continually* collected from us. When faced with an overly intrusive license agreement, most users will either completely ignore it or rationalize the collection of their data as something that simply will not happen or is not important.

The primary barrier to solving these issues is the difficulty involved in successfully communicating to a typical user the reality that most privately-owned computers are anything but private. Some tools, like the Ghostery add-on for Mozilla Firefox (available at https://addons.mozilla.org/en-US/firefox/addon/ghostery/) do this beautifully, presenting a notification in the lower right corner of the screen detailing precisely what was attempted and what succeeded.

Figure 1: Ghostery reporting attempts on user tracking to the user (purple rectangle in the lower-right) (NYTimes 2015)

Matthew Hansen, matthew.j.hansen@hotmail.com

The methodology applied by Ghostery is instrumental in changing the mindset of the user. When busy, the user can ignore the notification, yet still be aware that a tracking event occurred. The notification does not impair any task that the user was performing, and the actions allowed or blocked can oftentimes be ascertained in a fraction of a second with a glance of the eye.

Implementing a similar warning scheme on a personal computer to detect nearly any activity that might be disguised, encrypted, redirected, or just extremely rare is a non-trivial task. Software methods exist that can seamlessly move a single process from one point in system memory to another. (Offensive Security, n.d.) Operating System Virtualization can obfuscate the actions of a process altogether, and usually completely remove it from the reach of security tools on the host. An uncountable number of malicious applications directly attack security software in order to further conceal their activities. The numerous methods in use are simply impossible to track. Presently, one thing is certain: Eventually every single one of these programs will attempt to phone home.

2. Tattler

At its most basic level, Tattler is a system that links software use agreements, specific Snort alerts, and user attention. The result is a somewhat unpleasant experience, as Tattler presents to the user in real-time the reality that today's personal computers are typically little more than expensive spying devices. Consider the following screenshot from Tattler:

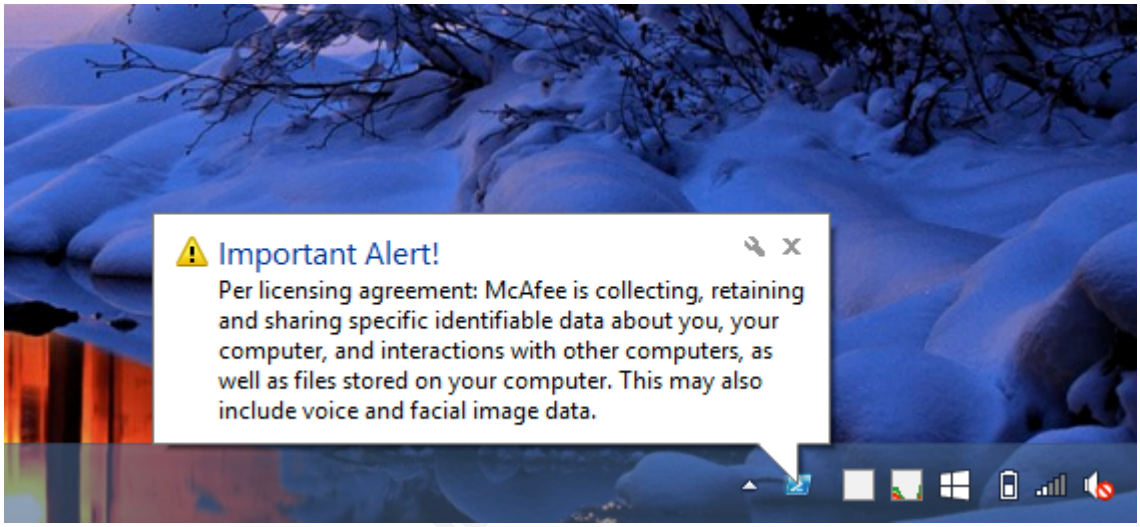

Figure 2 – Tattler reminding the user about the rights of a specific McAfee product.

Tattler's zealousness is highly configurable, as the major element providing information to Tattler is Snort. Tattler itself is a Powershell script that maintains streaming access to Snort's "alert.ids" file, ignoring all alerts (regardless of importance) that are not defined in the "Tattler.rules" file. Alerts that are defined in Tattler.rules are reported to the user via Windows balloon notifications after Snort logs them to alert.ids. Minor modification of Tattler can allow it to report on all Snort alerts, if desired. Configuring Snort not to generate an alert.ids file prevents Tattler from performing any useful tasks. Instructions for installing Tattler can be found in Appendix C.

2.1. Security Considerations Associated with Running Tattler

It is important to understand that Tattler is a tool for *changing user mindsets*, and should not be considered a tool that improves the overall security of the machine that it is running on. For example, Tattler relies completely on the presence of a fully-functional Snort IDS running locally. This configuration implies that at least one Internet-facing network interface is running in promiscuous mode. This fact alone will cause most traffic to/from the device to bypass the host-based firewall.

The process of installing and configuring Tattler requires that Powershell scripts be allowed to run on the local machine. This is not as significant of a vulnerability as the firewall issues noted above, but is an issue of concern nonetheless. It is important to point out, however, that Powershell script execution restrictions do little to impair an attacker anyways, as scripts can still be executed by pasting them into a Powershell console, echoing them into Powershell from standard input, or simply telling Powershell to execute them with "invoke-command". (Sutherland, 2014)

Tattler has no awareness of whether the user at the system console is legitimate or not, and may provide information on what software and services are installed to an adversary with line-of-sight access to the screen.

2.2. Tattler Code

Unlike most Powershell applications that process data in files, Tattler is restricted from using the more common Powershell methods of acquiring data due to the way Snort uses its alert.ids file.

```
# /*========================================================================*/
# Start tapping data from Snort alerts...
$Job = Start-Job -ScriptBlock { Get-Content -Tail 1 -wait  C:\Snort\log\alert.ids }
# /*========================================================================*/
```

- Get-Content: Possibly the most common data gathering tool in Powershell, Get-Content acquires the entire file every time it reads any part of it. This is acceptable on extremely small Snort alert files, but degrades performance quickly when reviewing anything larger than a few dozen megabytes.

- StreamReaders: StreamReaders are an excellent alternative to Get-Content. StreamReaders go to the specified location and wait for more data to arrive on the stream. They are lightweight, quick, reliable and completely incompatible with Snort as they respect Snort's own file use lock.
- -Tail 1 -wait: This method was developed by Microsoft specifically for the purpose of monitoring system log files and reporting new data as it arrives. Unfortunately, the tail/wait method does not give up access to the datastream while it is running, which increases the difficulty of using output data in concurrent tasks.
- A backgrounded, external process using the tail/wait approach: By backgrounding the process performing tail/wait (described above), and grabbing data directly off *its* stream, we can read Snort's alert.ids file in real-time without interfering with Snort's actions.

Powershell pattern matching is powerful, but oftentimes fails within streams for non-intuitive reasons. For example, stream matching does not always consider the hexadecimal sequence "0D0A" as a carriage-return-line-feed, and may dump several lines into an array as if they were a single line. While this does not cause issues for programs that directly output such data to the console, it does cause severe issues for programs that expect to be able to parse individual lines. For this and other reasons, several transformation methods help considerably when attempting to match on the portions of a Snort alert that are meaningful to Tattler. In particular, the rule sid (unique identifier associated with a particular Snort rule) and msg (human-readable message associated with a Snort rule) values are absolutely vital. Reliably acquiring this information knowing that at any time Tattler can be anywhere (including mid-sentence) in a datastream requires that we perform significant modifications to the stream itself before we can begin any kind of pattern matching.

```
# /*======================================================================*/
While (1){
  $data = Receive-Job $Job
  While ($data -ne $null){
```

```
    if($data -ne $null -And $data -ne ""){
      $strGrepAlerts = $data | select-string -pattern "$strAlertPattern"
      if($strGrepAlerts -ne $null){
        # /*=====================================================*/
        # Parse useful info out of whatever Snort just kicked out...
        $strJoinedGrepAlerts = [system.string]::join("`n", $strGrepAlerts)
        # Write-Host "--------------------------------"
        # Write-Host "$strJoinedGrepAlerts"
        ForEach ($line in $strGrepAlerts ){
          $sid = $strJoinedGrepAlerts.Substring(8,7)
          $rest = $strJoinedGrepAlerts.Substring(19)
          $msg = $rest.Substring(0, $rest.IndexOf("`["))
          # Write-Host "     --------------------------------"
          # Write-Host "     SID:  $sid"
          # Write-Host "     MSG:  $msg"
          # /*=================================================*/
```

Once the datastream of Snort's output is in a predictable format, it is a simple matter to perform rule lookup to determine whether Snort has alerted on something that is meaningful to Tattler, and then wait for more data to arrive.

```
          # /*=================================================*/
          # Compare what we just found to whatever is in
          # tattler.rules
          ForEach ($strRule in $arrTattlerRules){
            if($strRule.Contains($sid)){
              NotifyUser $strTitle $msg 'Warning'
            }
          }
          # /*=================================================*/
        }
        # Write-Host "     --------------------------------"
      }
    }
    $data = Receive-Job $Job
  }
  Start-Sleep -s $intPollingPeriod
 }
}
# /*=============================================================================*/
```

Notification of the user is the most exciting part of this entire process, and is made extremely simple by Powershell's excessive capabilities. Much of this code is a direct copy from Microsoft's own "Hey Scripting Guy" column (Microsoft Inc, 2015) regarding this very feature.

```
# /*============================================================================*/
function NotifyUser($strTitle, $strText, $strType){
  Add-Type -AssemblyName System.Windows.Forms
  if ($script:notification -eq $null){
    $script:notification = New-Object System.Windows.Forms.NotifyIcon
  }
  $path = Get-Process -id $pid | Select-Object -ExpandProperty Path
  $notification.Icon            = [System.Drawing.Icon]::ExtractAssociatedIcon($path)
  $notification.BalloonTipIcon  = $strType
  $notification.BalloonTipTitle = $strTitle
  $notification.BalloonTipText  = $strText
  $notification.Visible         = $true
  $notification.ShowBalloonTip(1000)
}
# /*============================================================================*/
```

Minor portions of Tattler have been omitted from this section. The complete script is available in Appendix A below.

2.3. Tattler Rules Generation

The tattler.rules file is generated through meticulous analysis of both software licensing agreements and specific packet captures containing unwantedware activity. Most of the network analysis can be performed within Wireshark using simple display filters and searches for specific strings. Oftentimes, the traffic Tattler is intended to alert on is encrypted, and as such, is difficult to reliably identify with a static Snort rule. In such a case, however, the remote server's public key must be transmitted as cleartext, and Snort can be configured to watch for this public key. The following procedural example details how to create a single Tattler rule for the Toshiba Service Station, an application discussed later in section 3.

1. Study the applicable Toshiba's Software Licensing Agreement. Aggregate the portions that state what types of information are collected, then reword it to use the present participle and condense the text to fit within a 255 character block. This character limit is vital, as standard Windows notifications only display the first 255 characters. The free utility Notepad++ (available at https://notepad-plus-plus.org/) is extremely helpful here (and is extremely useful when manually editing the

tattler.rules file), as it automatically performs character counts of selected text. In this case, our output is: "*Per licensing agreement: Toshiba is collecting and sharing your computer's OS version, software status, model/serial number, and possibly system usage information. This information may be used for business analysis, tech support, or quality assurance.*"

2. Prepare a setup that allows for raw packet capture from the system under test, ideally with an inline Ethernet tap: For purposes of this project, the DualComm DCSW tap (available at http://www.amazon.com/Dualcomm-DCSW-1005-Powered-Ethernet-Mirroring/dp/B002BSF112/) is recommended. The DCSW-1005 is a compact, USB-powered, network switch (ports 1-4), and Ethernet tap (Traffic across port 1 is tapped at port 5). It fits easily in an incident response "jump bag," and is advertised as supporting PoE (Power over Ethernet) pass-through as well. Running a packet capture locally on the system under test is acceptable as well, as this is the intended configuration of Tattler.

3. Select a packet capture tool and configure it to perform a full packet capture without performing any hostname/domainname lookups:
NOTE: Wireshark is known for containing many serious vulnerabilities, and the software configuration that allows for WinPCap to capture raw packets runs at least one layer beneath the host-based firewall. As a result, an otherwise hardened Windows host running a vulnerable version of Wireshark will quickly fall to an attacker *even if the host-based firewall blocks all inbound traffic*. Powerful exploits like the Rapid7 Wireshark LWRES Dissector buffer overflow (available at http://www.rapid7.com/db/modules/exploit/multi/misc/wireshark_lwres_getaddrbyname) should not be overlooked. For these reasons, it is recommended that traffic be captured using another process:

- TCPDump: TCPDump specifics are listed at http://www.tcpdump.org/tcpdump_man.html, but generally, the command will look something like:

  ```
  tcpdump -s 0 -nni <network interface> -w "capture.pcap"
  ```

- netsh: The built-in Windows netsh utility can be configured to perform packet captures as well. (Vandenbrink, 2015) Capture filters are recommended, but not required, and the command will look something like:

  ```
  netsh trace start capture=yes
  netsh trace stop
  ```

  Netsh automatically assigns a filepath and filename, and stores the capture in a proprietary Microsoft format that Snort and Wireshark cannot read. The Microsoft Message Analyzer (available at https://www.microsoft.com/en-us/download/details.aspx?id=44226) can read this file and save it in pcap format, however.

4. Review the packet capture for DNS query responses using the following Wireshark display filter:
   ```
   ip.dst == <local ip> && dns && udp contains toshiba
   ```
5. Review the resultant list, in our case:
   a. devices.toshibaplaces.com (213.161.81.189)
   b. csdsupport.toshiba.com (209.157.69.52)
   c. redirect.toshiba.com (209.157.69.28)
   d. onlineregistration.gedb.toshiba.com (209.157.69.8)
6. Determine based on later traffic from each IP address gained in step 5 above whether each conversation indicates the passage of any significant

data from the system under test: The following Wireshark display filters are useful:

a. **`ip.addr == 213.161.81.189`** – Web browsing traffic from an application identifying itself through HTTP user-agent strings as "Microsoft BITS/77". No obvious exfiltrations of data were noticed.

b. **`ip.addr == 209.157.69.52`** – Web browsing traffic from an application identifying itself through HTTP user-agent strings as "TAIS FDB Pinger". No obvious exfiltrations of data were noticed.

c. **`ip.addr == 209.157.69.28`** – Web browsing traffic from an application identifying itself through HTTP user-agent strings as "ToshibaFBDSend/1.0". No obvious exfiltrations of data were noticed.

d. **`ip.addr == 209.157.69.8`** – Web browsing traffic from an application that does not identify itself through HTTP user-agent strings. It sets up a TLSv1.2 encrypted session and exfiltrates 304 encrypted bytes. 720 encrypted bytes are sent back in an apparent response.
This is a likely candidate, and gives us several things to generate a decent Snort rule from:

   i. The content: "onlineregistration.gedb.toshiba.com"

   ii. A certificate signed by GoDaddy.com, with a hex serial number of "4b1e097f67d629", as well as other useful information, including the public key.

   iii. Direction of traffic flow ($EXTERNAL_NET -> $HOME_NET)

7. Attempt to replicate this traffic by manually executing select processes. A common challenge encountered here is that many of these applications

maintain their own state regarding when to initiate communications. As such, it is difficult to convince said application to phone home simply by running it.

8. Create a Snort Rule based on the information analyzed:
alert tcp $EXTERNAL_NET 443 -> $HOME_NET any (content:"onlineregistration.gedb.toshiba.com"; content:"|3082010a0282010100b8e9e13d84eb4d9866896c25348bc6cc206cb149d299af4b0acaff5bfab6c6e868d89007b35232aeaf9e0a51f8e6b3d49fe42f6ba1bba094276ebc08073e4a9bb283b11acfb59c464fd9adde1b8d8fbbacab68a52840e09709236931d91494234e6d09ff2574558b7d5464917962205d60eb872ad8811e7209beb14510778e12cc864bf407e17267c908cb9b6f685cd0e83daca183f735f9a3d6bf4d2be2f36c91847a8bb2e37e72f640ba822dc77126fee56e75f971c8f5ad3e693cd24157ade93eadeddf03a0bdec74d918ceaddd1618db3c697b50a85962fecc64f360a448177fcd38ca6532983eaa3bbe0c75523bbfa7ca88363ef1a21503a9926fac4ba50203010001|"; msg:"Per licensing agreement: Toshiba is collecting and sharing your computer's OS version, software status, model/serial number, and possibly system usage information. This information may be used for business analysis, tech support, or quality assurance."; sid:7000001;)

3. Tattling on Toshiba

Laptops assembled by Toshiba tend to have the "Toshiba Service Station" (TSS) application installed on them as of at least 2008. TSS "…periodically transmits to [Toshiba's] servers a limited amount of system information required to perform … updates or alerts." (TAIS, 2/2015) According to the license agreement, this information may include "…the Model, Part number, Serial Number, OS Version, Software Status… [and] …system, component and usage information…" (TAIS, 4/2015) It is important to note that Toshiba worded the licensing agreement in such a way that it is not immediately clear that TSS will also send "component and usage information," as such information is collected by a separate utility, Toshiba PC Health Monitor, and then submitted to Toshiba using TSS.

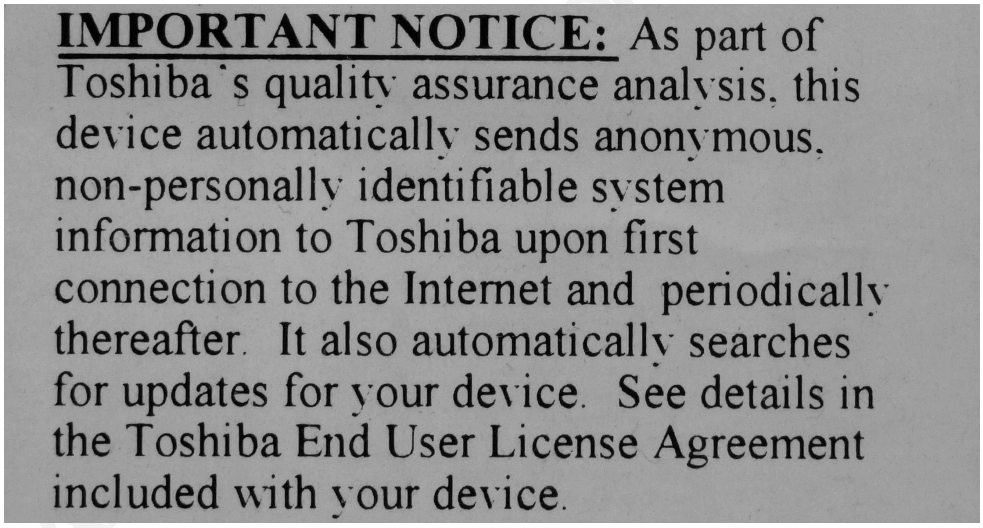

Figure 3 – Toshiba packaging material that provides an incomplete warning regarding Toshiba's data collection practices.

TSS does not have a spotless history when it comes to exploitable vulnerabilities. In early 2015, a privilege escalation vulnerability was discovered within TSS that can be exploited by providing a specially-crafted filepath string to TSS's search capability. (MITRE, 2015)

4. Tattling on McAfee

The claim put forth by the notification in section 2 is fairly serious, but nothing more than that put forth by the McAfee Privacy notice, which states that McAfee will collect:

- Real Names
- Email Addresses
- Usernames
- Passwords
- Physical Address
- Telephone Number
- Lists of websites you visit
- Social Media Identifiers
- Credit Card Billing Information
- Bank Account Information
- Photographs
- Images
- Biometric Information
- Network Usage Details

(McAfee 2014)

Using section 2.3 (Tattler Rules Generation) as a guide, we find appropriate items to search on from McAfee as well, but also an example of McAfee forwarding our information to a third party – Akamai Technologies.

- Mcloud.mcafee.com – approximately 10,000 encrypted bytes exfiltrated. Most of this information is encrypted, so our rule is limited to:
    - Content flow
    - Remote port of 443
    - The public key provided by mcloud.mcafee.com.
- Sadownload.mcafee.com – Attempting to resolve the domain name "sadownload.mcafee.com" results in a chain of new canonical names until finally an IP address belonging to Akamai is returned, corresponding to the domain name e7479.g.akamaiedge.net. This link can be verified easily, as indicated in the following figure:

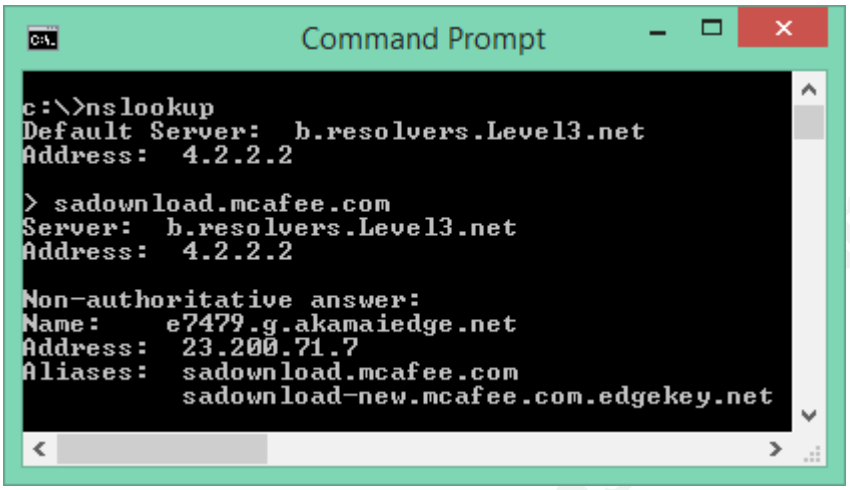

Figure 4 – Demonstrating a link between McAfee and Akamai

Literally dozens of back-to-back encrypted communication sessions occurred with significant amounts of data moving in both directions. An entirely different encryption certificate was used.

5. Tattling on Amazon

Unlike the other organizations reviewed in this project, the system under test never displayed a licensing agreement for Amazon.com. Nowhere in the build process was anything agreed to in terms of what information Amazon can send to or take from the system under test. Nevertheless, within the first few hours that the system under test was connected to the Internet, it generated several sessions to Amazon.com, some encrypted, some not encrypted. In one case Amazon.com initiated an encrypted session to the system under test. No user warnings or notifications were presented, and the activity was not obvious to the user. Although the communications were unexpected (and discovered by simply stumbling across them while reviewing a pcap capture file), they can be easily found within Wireshark simply by applying the filter "tcp contains amazon", or by searching for the public key used (refer to Appendix B for the complete public key).

6. Other Software

Although the scope of software communications analyzed during this project is necessarily brief, Tattler has been designed in such a way that its alerting capabilities can be extended by anyone who possesses the initial distribution of tattler.ps1 and an understanding of how to create Snort rules.

7. Conclusion

Without a significant change to the mindset and forensic talents of the typical user, organizations similar to those depicted here will continue to harvest, use, and sell information about the users of their products. It is unlikely that vague and misleading statements in privacy statements within software licensing agreements will become more honest or clear in the near future. Unfortunately, even if these changes were implemented, users would be unlikely to notice the conditions they agree to at software installation, and are even less likely to care.

A significant gap in user capabilities exists in terms of knowing who our systems are communicating with and what they are saying about us. Tattler partially fills this gap by leveraging existing capabilities in the intrusion detection and systems administration fields. The union of these capabilities is indeed powerful, but requires administration skills beyond the capabilities of most users, and likely further exposes the vulnerabilities on their machines by running their network interfaces in promiscuous mode. These issues may be lessened somewhat by running WinPCap without promiscuous mode, but may of the same issues still apply.

Future advancements will find more success leveraging existing enterprise level Data-Loss Prevention technologies bundled for home users. These tools will likely enjoy greater success if deployed with on-host agents having the capability to precisely link a specific application with a specific network communications session.

References

Kaspersky, (n.d.). What is Adware? Kaspersky Labs. Retrieved from https://usa.kaspersky.com/internet-security-center/threats/adware

Whittaker, Z. (2/19/2015). Until Superfish fix, Lenovo devices can't be trusted for secure work. ZDNet. Retrieved from http://www.zdnet.com/article/superfish-stop-using-your-lenovo-laptop-now/

Whittaker, Z. (8/12/2015). Lenovo used shady 'rootkit' tactic to quietly reinstall unwanted software. ZDNet. Retrieved from http://www.zdnet.com/article/lenovo-rootkit-ensured-its-software-could-not-be-deleted/

Temperton, J. (9/18/2015). AVG can sell your browsing and search history to advertisers. Wired.co.uk. Retrieved from http://www.wired.co.uk/news/archive/2015-09/17/avg-privacy-policy-browser-search-data

The New York Times. (9/26/2015). The New York Times Home Page. The New York Times. Retrieved from http://www.nytimes.com/?action=click&contentCollection=Politics&region=TopBar&module=HomePage-Title&pgtype=article

Offensive Security. (n.d.). Windows Post Manage Modules. Offensive Security. Retrieved from https://www.offensive-security.com/metasploit-unleashed/windows-post-manage-modules/

Sutherland, S. (9/9/2014). 15 Ways to Bypass the PowerShell Execution Policy. The NetSPI Blog. Retrieved from https://blog.netspi.com/15-ways-to-bypass-the-powershell-execution-policy/

Microsoft Inc. (2015). Displaying a Message in the Notification Area. Windows PowerShell Tip of the Week. Retrieved from https://technet.microsoft.com/en-us/library/ff730952.aspx

VandenBrink, R. (3/2015). No Wireshark? No TCPDump? No Problem! SANS Internet Storm Center. Retrieved from https://isc.sans.edu/forums/diary/No+Wireshark+No+TCPDump+No+Problem/19409/

TAIS, Inc. (4/2015). Toshiba End User License Agreement. Retrieved from https://support.toshiba.com/support/navShell?cf=su_eula&pf=true

TAIS, Inc. (2/24/2015). Toshiba Service Station (Vulnerability Update). Retrieved from http://www.support.toshiba.com/sscontent?contentId=4007184

MITRE Corporation. (2/27/2015) CVEDetails – Toshiba Service Station 2.2.13 Security Vulnerabilities. Retrieved from http://www.cvedetails.com/vulnerability-list/vendor_id-2920/product_id-31194/version_id-180217/Toshiba-Service-Station-2.2.13.html

McAfee, Inc (3/20/2014). McAfee Privacy Notice. McAfee.com. Retrieved from http://www.mcafee.com/common/privacy/english/docs/mcafee-privacypolicy.pdf

Appendix A – tattler.ps1

```
# /*=====================================================================*/
function Main(){
  # /*===================================================================*/
  # Initial variable assignment...
  $strTitle = "Important Alert!"
  $strText = "Nothing to report."
  $strSnortAlertPath = "C:\Snort\log\alert.ids"
  $intPollingPeriod = 1
  $strAlertPattern = "\[[0-9]:"
  # /*===================================================================*/
  # Start tapping data from Snort alerts...
  $Job = Start-Job -ScriptBlock { Get-Content -Tail 1 -wait  C:\Snort\log\alert.ids }
  # /*===================================================================*/
  # Figure out what we should be working on...
  $arrTattlerRules = GetTattlerRules "C:\Snort\rules\tattler.rules"
  NotifyUser 'Tattler' 'Tattler is resuming tattling...' 'Info'
  # /*===================================================================*/
  # Get to work...
  While (1){
    $data = Receive-Job $Job
    While ($data -ne $null){
      if($data -ne $null -And $data -ne ""){
        $strGrepAlerts = $data | select-string -pattern "$strAlertPattern"
        if($strGrepAlerts -ne $null){
          # /*=============================================*/
          # Parse useful info out of whatever Snort just kicked out...
          $strJoinedGrepAlerts = [system.string]::join("`n", $strGrepAlerts)
          # Write-Host "--------------------------------"
          # Write-Host "$strJoinedGrepAlerts"
          ForEach ($line in $strGrepAlerts ){
            $sid = $strJoinedGrepAlerts.Substring(8,7)
            $rest = $strJoinedGrepAlerts.Substring(19)
            $msg = $rest.Substring(0, $rest.IndexOf("`["))
            # Write-Host "     --------------------------------"
            # Write-Host "     SID:  $sid"
            # Write-Host "     MSG:  $msg"
            # /*=========================================*/
            # Compare what we just found to whatever is in
            # tattler.rules
            ForEach ($strRule in $arrTattlerRules){
              # Write-Host "Comparing $sid ($msg) to $strRule..."
              if($strRule.Contains($sid)){
                # Write-Host "$sid is in $strRule, so lets notify with: $msg"
                NotifyUser $strTitle $msg 'Warning'
              }
            }
            # /*=========================================*/
          }
          # Write-Host "     --------------------------------"
        }
      }
      $data = Receive-Job $Job
    }
    Start-Sleep -s $intPollingPeriod
  }
}
# /*=====================================================================*/
function NotifyUser($strTitle, $strText, $strType){
  Add-Type -AssemblyName System.Windows.Forms
  if ($script:notification -eq $null){
    $script:notification = New-Object System.Windows.Forms.NotifyIcon
  }
  $path = Get-Process -id $pid | Select-Object -ExpandProperty Path
  $notification.Icon            = [System.Drawing.Icon]::ExtractAssociatedIcon($path)
  $notification.BalloonTipIcon  = $strType
  $notification.BalloonTipTitle = $strTitle
  $notification.BalloonTipText  = $strText
  $notification.Visible         = $true
```

```
  $notification.ShowBalloonTip(1000)
}
# /*=============================================================================*/
function GetTattlerRules($strTattlerRulesPath){
  $arrTattlerRules = Get-Content "$strTattlerRulesPath"
  $arrTattlerRules
}
# /*=============================================================================*/
function Close-Script {
  Stop-Job $Job
    exit
}
# /*=============================================================================*/
Main("go")
# /*=============================================================================*/
```

Appendix B – tattler.rules

```
# Toshiba Privacy Issues...
alert tcp $EXTERNAL_NET 443 -> $HOME_NET any
(content:"onlineregistration.gedb.toshiba.com"; content:"|
3082010a0282010100b8e9e13d84eb4d9866896c25348bc6cc206cb149d299af4b0acaff5bfab6c6e868d8900
7b35232aeaf9e0a51f8e6b3d49fe42f6ba1bba094276ebc08073e4a9bb283b11acfb59c464fd9adde1b8d8fbb
acab68a52840e09709236931d91494234e6d09ff2574558b7d5464917962205d60eb872ad8811e7209beb1451
0778e12cc864bf407e17267c908cb9b6f685cd0e83daca183f735f9a3d6bf4d2be2f36c91847a8bb2e37e72f6
40ba822dc77126fee56e75f971c8f5ad3e693cd24157ade93eadeddf03a0bdec74d918ceaddd1618db3c697b5
0a85962fecc64f360a448177fcd38ca6532983eaa3bbe0c75523bbfa7ca88363ef1a21503a9926fac4ba50203
010001|"; msg:"Per licensing agreement: Toshiba is collecting and sharing your computer's
OS version, software status, model/serial number, and possibly system usage information.
This information may be used for business analysis, tech support, or quality assurance.";
sid:7000001;)

alert tcp $HOME_NET any -> $EXTERNAL_NET 80
(content:"|486f73743a20786d6c2e746169732e757064617465732e746f73686962612e636f6d0d0a|";
msg:"Per licensing agreement: Toshiba is searching for software updates that are
applicable to your machine."; sid:7000002;)

alert tcp $HOME_NET any -> $EXTERNAL_NET 80
(content:"|474554202f5570646174657357532f557064617465732e61736878|";
content:"sNumPartMktg"; content:"sNumSer"; content:"sNumLang"; content:"sVerOS";
content:"VerMajorSP"; content:"sDateBootFirst"; content:"sDateBootFirstSource";
content:"sNumPart"; content:"sCultureInfo"; msg:"Per licensing agreement: Toshiba is
collecting and sharing your computer's OS version, software status, model/serial number,
and possibly system usage information. This information may be used for business
analysis, tech support, or quality assurance."; sid:7000003;)

# McAfee Privacy Issues...
alert tcp $EXTERNAL_NET 443 -> $HOME_NET any
(content:"|3082010a0282010100c000d371939a8f739c22b2ab02601c83c6f68fbef6cec6fee8f48e712675
a93b562eff967e8fbeaaa8791bc07dbc2440cc19727c854895106bbcc8b92094be4f62ee833fa3dbc35a50cf7
56e2017b0787ff356b4501adcda71cb9603232a21845f234257ee0145359027aef19429eba3fa290df86d50cc
7313d7442733ca52db7b5a4840217f101fe2fb3c5089321087361dd99f0a1bd10753e5ec41af9687b8559442c
dc2d944df125dc28df7e526b765656dc739c6f373512f67c8bf5ecec1431bac1c0c5b91f88ac073724410ad08
ce869596216e8252a19abf004cf4c5745c111618e730024c9d52baeba62f8065fd59d5b963ff0807313913463
9adb1650203010001|"; msg:"Per licensing agreement: McAfee is collecting, retaining and
sharing specific identifiable data about you, your computer, and interactions with other
computers, as well as files stored on your computer. This may also include voice and
facial image data."; sid:7000010;)

# Akamai on behalf of McAfee...
alert tcp $EXTERNAL_NET 443 -> $HOME_NET any
(content:"|3082010a02820101009a7d98681140c15f72ec55b3b163f332227291c61605bb088231b4f6eed4
1839112f2eda47fe51316e5bf2a90aeb2fbbf56159655702cd80ffc770325489fddbae9972d44f0c26b92e633
07dde145b6ad7527821f9bfbc50d5541259d8b536d92147b83f6a581d8c72e19795d3e145a8f15ae5befee353
7ca5f052e0cf39940c1971f2c02507487d1ce6f139252f987943e81872f46586985a000447da4b581e7c86b14
b35a620001ccd1b3b225dd193283312239408aac33af5d1c68c7e99d318a0ad9d18cf49ad1003f79933268646
9a2fa0ba6c6ec88802b76efa7a9e984aee9a317d1914600cec8f20233cda9726b6ea806c8a579e20ee6f17254
a32ad350203010001|"; msg:"Per licensing agreement: Akamai is collecting, retaining and
sharing specific identifiable data about you, your computer, and interactions with other
computers, as well as files stored on your computer. This may also include voice and
facial image data."; sid:7000011;)

# Amazon Unrequested HTTPS
alert tcp $EXTERNAL_NET 443 -> $HOME_NET any
(content:"|3082010a028201010086878d7b30d65436e72f82aeca21f89e334bcb62cf09cc3bdcbf188d38b8
fa57b3f4a24f4860762af14663e34ff354b5bec417c898fac185692cb9a1729ab49d77670374e077959de1111
2809238a163882e374d07023635fc804c90bd6a055e37187e6b06ae122a2c22bd9e14c087e4bbc27d21f582d7
b14d28540735a21ce362631a922a0f902d273f4dd43bc39cd01c974aa2384a507d7248c7741e80318d6dfd311
711ab8aedf5afa4df0384b9c4a2f5a474e09ea66d37b7b09183ddf9e5708e4f4caf84eabdf69fdbedb26ce2dd
7ce46ad46cd85e6b4e1f8ebcc9a186da9de550ea6189c2ccf25c5a5d374e71a3c96db6dbdf819249e6f3375a9
3e60a530203010001|"; msg:"Amazon.com is establishing an encrypted data transfer session
with your machine."; sid:7000020;)
```


Matthew Hansen, matthew.j.hansen@hotmail.com

Appendix C – Tattler Installation Instructions

1. Install Snort...
    a. Install Snort for Windows as you normally would.
    b. Ensure that WinPCap starts at boot.

2. Import Tattler Rules...
    a. Copy the "tattler.rules" file to C:\Snort\rules\tattler.rules.
    b. Open the snort.conf file (Usually C:\Snort\etc\snort.conf)...
        i. Add the line "include $RULE_PATH/tattler.rules" to the rules section.
        ii. Repair references to Linux filesystems:
            1. /usr/local/lib/snort_dynamicpreprocessor/ -> C:\Snort\lib\snort_dynamicpreprocessor
            2. /usr/local/lib/snort_dynamicengine/libsf_engine.so -> C:\Snort\lib\snort_dynamicengine\sf_engine.dll
            3. /usr/local/lib/snort_dynamicrules -> C:\Snort\rules
        iii. Either disable the reputation preprocessor or ensure that C:\Snort\rules\white_list.rules exists
        iv. Either disable the reputation preprocessor or ensure that C:\Snort\rules\black_list.rules exists
        v. Remove references to rules files that do not exist.
        vi. Perform any other tuning on snort rules as desired or required.
        vii. Save and close the snort.conf file.

3. Verify the Snort configuration is valid...
    a. Open cmd.exe as an administrator (elevated shell)
        i. cd C:\snort\bin
        ii. ipconfig /all
        iii. snort -W
        (Determine what interface you want Snort to listen on and substitute it for "-i 2" below)
        iv. snort -l C:\Snort\log -h 192.168.0.0/24 -i 2 -c C:\Snort\etc\snort.conf
        v. If the Snort configuration is valid, Snort will appear to "hang" after echoing "Commencing packet processing (pid=####)" If Snort errors and quits, then something is wrong with Snort's configuration – troubleshoot and repair as necessary.
        vi. CTRL+C to end the Snort process. Snort should exit.

4. Make Snort run as a service...
    a. From the same command shell in the previous step (and again replacing "-i 2" with the network interface you prefer)...
        i. snort /SERVICE /INSTALL -l C:\Snort\log -h 192.168.0.0/24 -i 2 -c C:\Snort\etc\snort.conf (NOTE: Substitute your subnet as appropriate.)

Verify that the command completed successfully.
ii. exit
b. Open the services.msc console...
i. Start the "Snort" service and verify that it succeeds.
ii. Set the "Snort" service to start automatically.
iii. Close the services.msc console.

5. Install Tattler...
a. Copy the tattler.ps1 powershell script to C:\Snort\bin\tattler.ps1
6. Reconfigure powershell to allow the execution of scripts...
a. Open Powershell as an administrator (elevated shell)
i. Set-ExecutionPolicy unrestricted
ii. Enter "y" when prompted.
iii. Close Powershell

7. Configure Tattler to run as a scheduled task...
a. Open cmd.exe as an administrator (elevated shell)
i. schtasks /create /TN "Tattler" /SC ONLOGON /TR "Powershell.exe -noprofile -windowstyle hidden -Command C:\Snort\bin\tattler.ps1" /IT
ii. exit
b. Open the taskschd.msc console.
i. Expand "Task Scheduler (Local)"
ii. Select "Task Scheduler Library"
iii. Open the "Tattler" scheduled task
iv. Go to the "Settings" tab
v. Ensure "Stop the task if it runs longer than:" is not checked.
vi. Click OK
vii. Close the taskschd.msc console

8. Reboot the machine.
9. Logon to the machine.
10. A powershell window will briefly appear, ignore it.
11. A notification should appear stating that "Tattler is resuming tattling..."