DUPING THE MACHINE – MALWARE STRATEGIES, POST SANDBOX DETECTION

James Wyke
Sophos, UK

Email james.wyke@sophos.com

ABSTRACT

Sandboxes and automated analysis environments are key tools for combating the exponential growth of malware. There are a huge range of different solutions available, and they are used in a wide variety of situations throughout security companies and large IT departments across the globe. In many cases, sandboxes are used as part of an automated system where data is extracted, fed into other systems, and decisions are made on the nature of the sample under examination. Inevitably, sandboxes can be detected, and malware that does so is left with a choice. The majority of malicious samples that detect that they are executing in an artificial environment will exit immediately, but there is a growing subset of malware families that choose to do something more cunning.

In this paper, we explore the different strategies malicious samples employ once a sandbox has been detected. We present examples of decoy behaviour that ranges from dummy files being dropped to the use of fixed path names, bogus DNS and HTTP requests, and misleading configuration files being delivered. We examine samples of malware families including Andromeda, Shylock, Simda and Vundo.

We classify the techniques involved and assess the motivation for each approach by determining the benefit to the malware author in each case.

We conclude by analysing the consequences of failing to realize we are observing bogus behaviour from the sample, such as false positives, prolonging of the life span of the threat, and embarrassing publications where the authors fail to realize they are describing dummy behaviour. Finally, we explore ways in which we might prevent ourselves from falling victim to the same techniques again.

BACKGROUND

With the number of new malware samples seen every day now exceeding 200,000 [1], it is impossible for any organization to process all the samples they encounter manually. This problem was recognized many years ago and led to the emergence of automated analysis systems that attempt to replicate the work of a human in a fraction of the time. A scalable solution can process many hundreds of thousands of samples each day, and can extract most if not all of the pertinent information that would otherwise take a skilled analyst many hours.

The explosion in malware volume has heightened the need for effective automated malware analysis, with many commercial, open-source and custom in-house solutions in wide use. The majority of these solutions execute the sample in a virtualized environment, as this tends to be a more flexible and scalable strategy than using physical machines, and are usually referred to as 'sandboxes'.

However, since we are executing the sample in an artificial environment, there exists the possibility that the nature of the analysis may be detected by the sample being examined. Indeed, many malware families, and particularly malware 'cryptors' and packers used to obfuscate *Windows* PE files, include some level of virtual machine (VM) or sandbox detection.

The techniques used to detect a VM or automated analysis environment are many and varied: there are simple checks for process names used by components of the VM software, checks for registry keys and values that give away the particular VM manager, techniques such as Red Pill [2] that rely on side-effects of the processor virtualizing certain x86 instructions, checks for user interaction such as mouse movement [3], and techniques that attempt to establish whether the system 'looks real' by checking that certain commonly installed software such as *Microsoft Office* are installed and that tools that are typically used for analysis (such as *Wireshark*) are not installed. Although the analysis environment can be hardened against many of these techniques, new methods are being discovered and it can be argued that no sandbox, or VM in particular, is completely undetectable.

From the perspective of a malware author, the purpose of detecting that execution is taking place in an artificial analysis situation rather than on a genuine victim's machine, is to alter its behaviour to hide aspects of its functionality from those that wish to analyse the sample. The most obvious and common manifestation of this intention is to terminate execution immediately after the VM has been detected. To the automated analysis environment it appears that the sample in question failed to execute correctly, as there will typically be no useful output. This meets the goals of the malware author by concealing the functionality of the sample. This may prevent the sample from being classified as malicious, or perhaps more importantly, may conceal critical information such as command and control (C&C) addresses.

The concept of concealing information such as C&C addresses is an important one, as it highlights the fact that sandboxes increasingly comprise only one part of a larger automated system that processes the output of the analysis and may perform further activities such as extracting actionable items – C&C addresses are a good example – and publishing them to other systems such as a URL blacklist. We can now see that if, instead of simply terminating execution when a VM is detected, the sample contacts a different URL to that which would have been contacted if the a VM had not been detected, the malware author can create problems for those running the sandbox and the secondary systems processing the results of analysis. Let us explore some of the possible approaches that malware can take once a sandbox has been detected by looking at several malware families that have chosen not to simply end execution, and the kinds of activity they exhibit.

DIFFERENT STRATEGIES

Andromeda

Andromeda is a bot that can download a variety of modules and is often used to distribute other malware families such as Gameover Zeus [4]. Through several iterations, Andromeda has often employed some element of VM detection followed by unusual alternative behaviour.

Samples use several methods to detect if analysis is taking place in a sandbox, including checking the list of running processes for names such as vboxtray.exe, wireshark.exe, and checking registry values for giveaway strings such as those found at:

HKLM\SYSTEM\CurrentControlSet\Services\Disk\Enum

Following the analysis environment detection code, there is a branch where further payload code is decrypted and executed. If the malware has detected that it is not executing on a live machine, then bogus payload code is loaded. If the checks all come up negative, and it is thus assumed that the system is a genuine victim machine, then the true payload will be loaded (Figure 1).

```
   cmp     dword ptr [ebp-364h], 'awmv'
   jz      short decrypt_bogus_payload
   cmp     dword ptr [ebp-364h], 'xobv'
   jz      short decrypt_bogus_payload
   cmp     dword ptr [ebp-364h], 'umeq'
   jz      short decrypt_bogus_payload

cause_exception_decrypt_genuine_payload: ; CODE XREF: sub_B1B98+B6↑j
                        ; _1961:000B1DE7↑j ...
   mov     eax, [ebx+3Ch]
   lea     eax, [ebx+eax+18h]
   or      word ptr [eax+46h], 80h

decrypt_bogus_payload:  ; CODE XREF: sub_B1B98+102↑j
                        ; sub_B1B98+15C↑j ...
   push    ebx
   push    402544h
   call    sub_B1F03
```

Figure 1: Andromeda payload decision.

The dummy payload itself has varied slightly over time but has mainly consisted of copying the original sample to a fixed pathname, creating a runkey entry in the registry that points to the copy, and opening up a listening socket on TCP port 8000 which then waits for incoming connections (Figure 2).

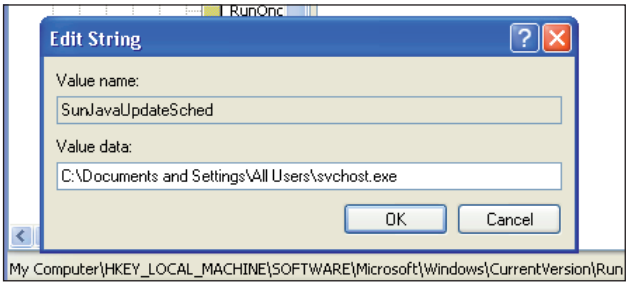

Figure 2: Bogus payload file path and run key.

This is in stark contrast to the genuine payload (Figure 3), where the .exe file is copied to a different location with a randomly generated filename, a different autostart point is created in the registry, and an outbound HTTP request is sent to receive instructions from the C&C server (Figure 4).

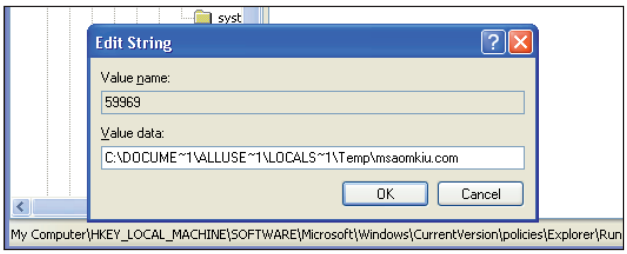

Figure 3: Genuine file path and registry autostart point.

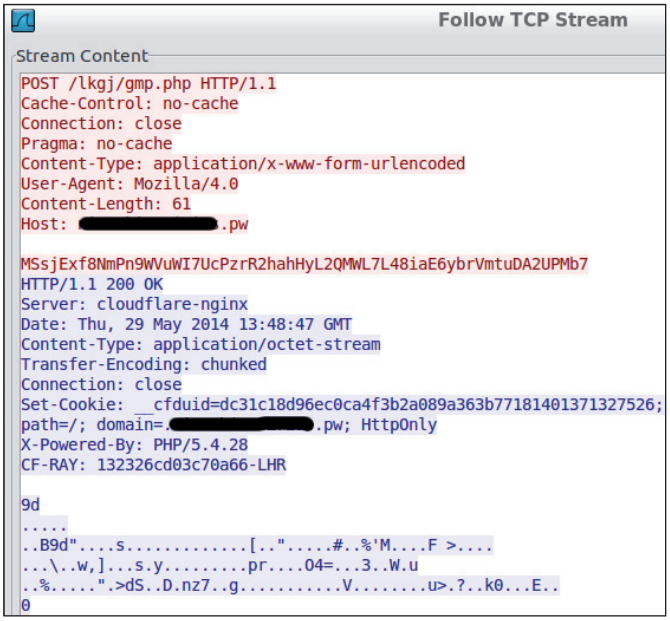

Figure 4: Outbound HTTP request.

One of the key points to note about Andromeda's decoy behaviour is that the behaviour itself is very identifiable. It is easy to establish that a sample is Andromeda because it will always perform the same bogus, but distinguishable, behaviour when executed under a VM that is not hardened against VM detection techniques. This shows that the authors are not primarily interested in preventing the sandbox from determining that the sample is malicious. Behavioural signatures that match on the fixed pathname are easy to write. Rather, it seems that the main goal is to keep the C&C addresses hidden from those analysing the sample, extending the lifetime of those addresses and reducing the administration overhead involved in finding new hosting providers when existing servers are taken down or are blocked by too many network security solutions.

A secondary consequence of this kind of decoy behaviour is that the fake behaviour is often incorrectly assumed to be the *only* behaviour that the sample will ever display. This can lead to miscategorizing the whole malware family as something relatively benign and may result in publications that mistake the decoy behaviour for genuine behaviour [5].

Simda

Simda is primarily a backdoor trojan that is mostly used to steal credentials for a variety of online banking systems [6]. Simda

uses a wide range of techniques to detect the presence of a VM or analysis environment, including checking the *Windows* ProductID in the registry against known values that are found in public online sandboxes, looking for running processes commonly found on analysis machines and for registry entries that indicate that software commonly used by researchers is installed (Figure 5).

```
0x00E0B8: dumpcap.exe
0x00E0F8: ZxSniffer.exe
0x00E138: Aircrack-ng Gui.exe
0x00E178: observer.exe
0x00E1B8: tcpdump.exe
0x00E1F8: WinDump.exe
0x00E238: wspass.exe
0x00E278: Regshot.exe
0x00E2B8: ollydbg.exe
0x00E2F8: PEBrowseDbg.exe
0x00E338: windbg.exe
0x00E378: DrvLoader.exe
0x00E3B8: SymRecv.exe
0x00E3F8: Syser.exe
0x00E438: apis32.exe
0x00E478: VBoxService.exe
0x00E4B8: VBoxTray.exe
0x00E4F8: SbieSvc.exe
0x00E538: SbieCtrl.exe
0x00E578: SandboxieRpcSs.exe
0x00E5B8: SandboxieDcomLaunch.exe
0x00E5F8: SUPERAntiSpyware.exe

0x00E840: SYSTEM\CurrentControlSet\Services\IRIS5
0x00E8C4: Software\eEye Digital Security
0x00E948: SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Wireshark
0x00E9CC: SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\wireshark.exe
0x00EA50: SOFTWARE\ZxSniffer
0x00EAD4: SOFTWARE\Cygwin
0x00EB58: SOFTWARE\Cygwin
0x00EBDC: SOFTWARE\B Labs\Bopup Observer
0x00EC60: AppEvents\Schemes\Apps\Bopup Observer
0x00ECE4: Software\B Labs\Bopup Observer
0x00ED68: SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Win Sniffer_is1
0x00EDEC: Software\Win Sniffer
```

Figure 5: Simda VM detection strings.

Whether the checks come up positive or not, Simda gathers certain information about the victim system including the computer name, the ProductID, the Volume Serial Number from the C:\ drive and, importantly, whether or not the system passed the anti-analysis environment checks. It sends the data back to a C&C server encoded in the URL of an HTTP request (Figure 6).

Follow TCP Stream

Stream Content

```
GET /?uOCE3a31=%96%CB%A9%D0%AB%ABb%D5%97cm%94d%93%98jg%B1kil%93%98q%A0%95%8F%98%A4%A2%A8%B1tk%AA%
D7p%A8%E8%9C%96%E4%DB%D8k%A2%98dl%98Y%9D%A5%A0p%DE%A5f%88%84%5D%A6%D4%95k%96%A3%9D%A8%94v%8By%
96xb%B4%A6%A7%A3%B6%B9g%A9%97wx%A7welki%ABm%7BjW%A9%9E%DEn%5E%93%9F%9D%A5%82tuc%91cb%A2%95%A0%9F%
A2%A4%5E%A1%8D%A7t%91cagid%BCl_%A5%ABtl%9DU HTTP/1.1
Host: report.qg17a31793y793179k1.com
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Trident/4.0; .NET CLR 2.0.50727; .NET CLR
1.1.4322; .NET CLR 3.0.04506.590; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729)
```

Figure 6: Simda HTTP check-in.

If an artificial environment is detected, the sample will enter an infinite loop. However, since the HTTP request has already been sent, the C&C server has already received information about the analyst's machine, including the IP address. The owners of the Simda botnet appear to be using the IP addresses they have collected in this way to blacklist researcher and security company machines.

Through experimentation, we discovered that if a sample that had been executed in a VM and reported back to the C&C server was subsequently executed from a physical machine that passed all the analysis environment checks, it would still get stuck in an infinite loop. We managed to trace this behaviour to the place in the code where the data sent back by the C&C server is checked by the sample. If there is a certain value at a certain offset in the data received, then the sample will enter the infinite loop despite all the client-side checks passing (Figure 7). So it seems that our earlier execution of the sample under a VM had caused all further requests from the same IP address to be denied by the server.

```
loc_4025B4:                             ; CODE XREF: start+409↑j
                mov     ecx, 1Fh
                call    GatherInfoAboutMachineAndSendToServer
                call    GetInfoEncodeSend
                cmp     eax, 2
                jnz     short server_replied_enter_infinite_loop
                push    eax
                push    ebx
                push    1
                lea     ecx, [eax+28h]
                call    GatherInfoAboutMachineAndSendToServer
                call    Sub_403E40
                jmp     loc_402A5A
; ---------------------------------------------------------------------------

server_replied_enter_infinite_loop:     ; CODE XREF: start+426↑j
                cmp     eax, ebx
                jle     short loc_4025F2
                push    eax
                push    ebx
                push    1
                mov     ecx, 29h
                call    GatherInfoAboutMachineAndSendToServer

infinite_loop:                          ; CODE XREF: start:infinite_loop↓j
                jmp     short infinite_loop
; ---------------------------------------------------------------------------
```

Figure 7: Server replies: enter infinite loop.

The Simda authors do not seem to be overly concerned about concealing C&C addresses from researchers. Instead, their goal seems to be to hinder analysis of the threat through both client-side and server-side mechanisms. Even though technologies such as NAT will mean that IP address blacklisting could prevent legitimate infections, the Simda authors are prepared to suffer this loss as long as a greater understanding of how the bot works is prevented.

Vundo

Vundo is a malware family that has been through many guises over the years, most recently being known as Ponmocup [7], but throughout its lifetime the general payload has been to push adware onto victims' systems. Vundo checks for the existence of a VM through a variety of means including checking the SystemBiosVersion value of the HKLM\HARDWARE\DESCRIPTION\System key in the registry.

Vundo's strategy once a sandbox has been detected is most easily demonstrated by observing the network activity under a VM and comparing it to that which takes place on a real machine. In both cases, an initial DNS request is made, the response to which is ignored. Since this initial request is ignored it could be to any domain, but recent samples have been favouring the domain fasternation.net. An HTTP request is then made, but both the URL and the host used are different depending on whether or not a VM is detected.

As can be seen in the example shown in Figures 8 and 9, if a VM is detected a request is made to 12.6.182.165, whereas if a VM is not detected, the request is sent to 93.115.88.220. Vundo is not only attempting to conceal its C&C server addresses but is also providing a decoy address that has no association with the botnet.

This is a clear case of the malware sending a bogus HTTP request when a sandbox is detected. This has the consequence

```
http_requests
request: http://12.6.182.165/adj/Category.aspx
```

Figure 8: Vundo decoy HTTP request.

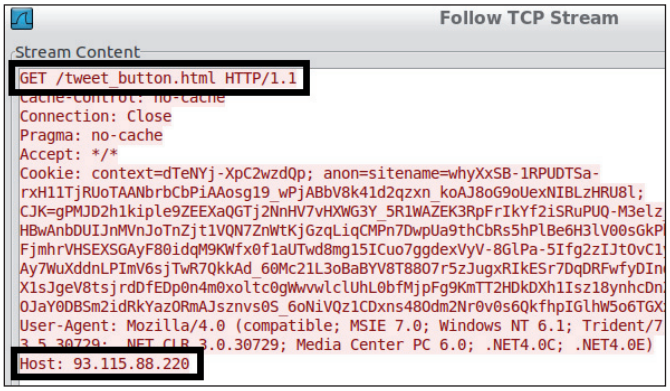

Figure 9: Vundo genuine HTTP request.

that the identity of the genuine C&C server is harder to ascertain, and in a similar way to the Andromeda case, it may survive longer before it is taken down or blocked by network security products. However, the use of the decoy URL is a more sinister development, as this means we cannot trust the data coming out of the automated analysis system. If we are blindly adding all URLs contacted by the malware to network blacklists, then we will have false positives when encountering this kind of malware.

Although in this case the only bogus information is a URL and server address, it highlights the general concept of deliberately attempting to cause security companies to false positive or otherwise publish erroneous data, by changing behaviour once a sandbox has been detected.

Shylock

Shylock is a banking trojan that is notable in that it is not sold as a kit but rather is privately developed and operated by one group [8]. Its configuration architecture is similar to other banking families, such as Zeus, in that the malicious binary holds minimal configuration information – only a URL – from which the full configuration file is downloaded. This file contains the other essential information such as the address to which stolen data is sent and a URL from which to download the web injects file which contains all the extra code that will be added to web pages when specific URLs are browsed to. The configuration file is very important when trying to gain a better understanding of what a particular Shylock sample is aiming to achieve. It contains URLs from which further modules will be downloaded, and the web injects file gives important indications about which financial organizations are being targeted and what extra information may have been elicited from the victim.

When Shylock first checks into its C&C server, it sends a large amount of information about the infected machine (Figure 10). This includes data about the machine itself, such as the CPU speed and amount of RAM installed; data about the OS, such as the *Windows* version, install date and product key; and data about the installed programs, such as the anti-virus software, the browser installed and the programs that are set to run automatically at system startup in the registry.

Figure 10: Shylock check-in data.

Shylock contains VM detection techniques that include looking for common registry entries and processes used by VM manager software. If a VM is detected, extra data gets included in the initial outbound request: a VirtualMachine=Yes field is added to the data, as shown in Figure 11.

```
||||AntiMalwares=Sophos; VirtualBox|||||VirtualMachine=Yes|||||
```

Figure 11: VirtualMachine field added to data.

The inclusion of this field is the source of Shylock's unusual anti-sandbox strategy. When the initial check-in request has been received from the newly infected machine, the C&C server examines the included data and will make a decision on which configuration file to deliver to the victim based on whether or not the VirtualMachine=Yes field is present. If the VM field is included, then a very basic configuration file is returned which does not include the URLs of any further modules to download and includes a very uniform URL for the web injects: '/files/hidden7710777.jpg' (see Figure 12).

```
<hijackcfg>
        <botnet name="net2"/>
        <timer_cfg success="1800" fail="1800"/>
        <timer_log success="1200" fail="1200"/>
        <timer_ping success="1800" fail="1800"/>
        <urls_server>
                <url_server url="https://bai.su/index.php"/>

                <url_server url="https://sxo.su/index.php"/>

                <url_server url="https://pfh.cc/index.php"/>
        </urls_server>
                <httpinject value="on" url="/files/hidden7710777.jpg"


</hijackcfg>
```

Figure 12: VM detected config file.

This is noticeably different from the configuration file that is returned when the VirtualMachine field is not included in the check-in request. This time, we have a plug-ins section that includes the URLs for several plug-ins – BackSocks, DiskSpread, MessengerSpread, PGP, an archiver URL and a url_update URL. Additionally, we can see that each URL path, including the httpinject URL path from which the web injects

are downloaded, includes a directory that was not present in the VM-detected configuration file, in the example shown in Figure 13, this directory is 010-update-d9hbjz6.

```
      <archiver url="https://lud.su/files/rar.exe" cmd="a -r -dh -ep2 -v500k"/>

      <url_update md5="9fd741c8251fce276dfa587af274e045" url="/files/010-update-

      <httpinject value="on" url="/files/010-update-d9hbjz6/hidden7770777.jpg" m
      <grabemails value="off"/>
<plugins>
      <plugin name="BackSocks" url="/files/010-update-d9hbjz6/bsds.gsm" value="1
      <plugin name="DiskSpread" url="/files/010-update-d9hbjz6/dsp.psd" value="o
      <plugin name="MessengerSpread" url="/files/010-update-d9hbjz6/msg.gsm" val
      <plugin name="PGP" url="/files/010-update-d9hbjz6/pgp.asc" value="on" cmd=
</plugins>
```

Figure 13: Genuine config file.

We can see further evidence that the Shylock authors are trying to deceive researchers in the web injects files that are returned from the URL provided in the fake configuration file and from the URL in the genuine configuration file. The web injects from the fake file are very generic and change little over time. They still look genuine, as they are designed to pass casual inspection. When the web injects from the genuine configuration file are examined we can see that these are much more geographically targeted and contain more advanced JavaScript and HTML code (Figure 14).

```
; ======== COOKIES & SOLS =======
<unit>
<cookies_get domain="retail.santander.co.uk" request="/EBA
<sols_get domain="retail.santander.co.uk" request="/EBAN_A

<cookies_get domain="business.santander.co.uk" request="/E
<sols_get domain="business.santander.co.uk" request="/EBAN

<cookies_get domain="*secure.bankofamerica.com*" request="
<sols_get domain="*secure.bankofamerica.com*" request="*my

</unit>

; ======== SIGNBOTID  =======
<signbotid  value ="YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY" />
<signbotnet  value ="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" />

; ======== ALERTBLOCK =======

; AZ
<alertblock domain="*big-web-svcs.cc" />
<alertblock domain="*safety-for-all.cc*" />
<alertblock domain="*erp-cloud.cc*" />
```

Figure 14: Genuine web injects.

When Shylock detects that it is running in a VM or sandbox, it does not attempt to conceal its C&C addresses or hide the nature of its functionality from the analysis system. The functionality that is hidden is done so at a more subtle level. From the infected machine's perspective there is very little difference in behaviour, merely the extra data added to the HTTP request when a VM is detected. It is only when the data sent back by the server is analysed in depth that we realize we are being fooled. The benefits to the malware authors in this case are that researchers will not be aware of new plug-in files that only appear in the genuine configuration file, and that banks and other financial organizations will not be aware that they are specifically being targeted or aware of what form the web injects that target their web applications will take.

CATEGORIZATION OF TECHNIQUES AND GOALS

Having analysed several real-world post sandbox detection strategies, we can begin to group the techniques used and the intentions and goals of the perpetrators. Table 1 attempts to broadly group the techniques used with a more verbose description, an example, and the goal of the malware author in each case.

CONSEQUENCES

Although we have touched on what the consequences can be for failing to realize that an alternative behaviour is being observed, it is worth going through them in greater detail.

For the more basic techniques, such as being presented with completely different, more benign behaviour as in the Andromeda example, there is the obvious consequence of public embarrassment when material is published detailing the decoy behaviour of the threat without the realization that this is not the way the threat behaves in a real system. The researcher has effectively fallen for the ruse.

Where genuine C&C addresses are hidden, we observe that the domains in question will often remain live for many weeks after they were initially introduced. Typically, domains known to be call-home addresses for malware will have relatively short shelf lives as they are blocked by security products or the hosting provider takes action to remove malicious content. The easiest way to determine the call-home address for a sample is to execute it in a sandbox. So if the sandbox fails to extract the C&C address, then fewer people, systems and companies will be aware that it is malicious, and thus it will live on for longer.

The blacklisting case highlights the need to use a fresh infrastructure when carrying out analysis, as previous use of IP addresses or machines may invalidate future analyses. Once a server outside of our control decides to decline our requests there is little we can do to remedy the situation, except send traffic through a different address.

The case where decoy behaviour is displayed that is designed to induce a negative consequence on the larger analysis system, such as the decoy HTTP request displayed by Vundo, is one that requires careful consideration from companies that process large volumes of samples and take further action on the artifacts of analysis. Evidence has been presented elsewhere that shows this kind of attack is already taking place in other forms, such as against automated detection from AV vendors [9]. This technique can also be used to cause a vendor to assign attribution to an innocent party, such as one cybercrime group seeking to lay blame for an attack on a rival group, or a nation state causing an enemy nation to be blamed for its actions.

More subtle attacks, such as the altered configuration file distributed by Shylock, have consequences further along the

Technique	Description	Example	Goal
Alternative, benign behaviour	The true nature of the sample is hidden along with data such as C&C addresses, to be replaced with different, more benign behaviour	Andromeda's decoy pathname and listening socket	Conceal C&C addresses, extend lifetime of network infrastructure, reduce level of community knowledge about threat
Blacklisting	Artifacts such as IP address are identified as potentially belonging to researchers, normal execution will not take place from these addresses even if other checks pass	Simda reports detected sandboxes to C&C server, subsequent requests from real machines from the same IP are instructed to enter infinite loop by server	Prevent researchers from further understanding the threat, build up list of likely security company IP addresses
Decoy addresses	Alternative C&C addresses are substituted for the genuine value when artificial environment is detected	Vundo sends HTTP request to decoy address when first executed	Conceal genuine C&C address, divert attention to fake address, potentially induce false positives
Fake configuration data	Configuration information returned by C&C servers is adjusted based on whether a sandbox is detected	Shylock serves up dummy config file and dummy web injects if a sandbox is detected	Conceal extra functionality not evident from the sample through server interaction, hide targeted URLs and injected code, hide existence of further modules

Table 1: Categorization of malware techniques and goals after a sandbox is detected.

chain of events that take place when a victim is infected with a sample. Banks and other financial organizations keenly study banking malware and the custom web injects they employ to better understand how their customers may be defrauded by such malware. More advanced constructions, such as automatic transfer systems that can initiate bank transfers entirely through the injected JavaScript, are particularly valuable to the security departments of banks. By ensuring that these features are not evident when the sample is analysed in a sandbox, the malware authors can keep them hidden from the anti-fraud departments that would otherwise find ways to detect and block the activity.

PROTECTION

When attempting to ensure that our sandbox system does not fall prey to these types of attacks, the most simplistic approach is to make our analysis environment look as much like a real system as possible, so that any checks that the malware may make are defeated.

Many of the basic techniques used to detect a VM are easily thwarted. Simple hardening, such as not installing guest additions and masking giveaway strings in the registry, will defeat the majority of VM detection techniques. More advanced methods, such as detecting assembly instruction differences, may be defended against by compiling custom versions of the VM software or by applying custom configuration options.

The only guaranteed method to defeat all VM detection techniques is to use a physical machine, but this can be difficult from a management and automation point of view, meaning it may not be possible to put huge numbers of samples (e.g. 400,000 per day) through such a system. Even then, checks that attempt to establish the machine is a genuine victim rather than an analysis machine may still succeed based on the software installed or other similar checks that try to establish that a real human has been using the system. If an attack is truly targeted, then the sample may check for extremely specific settings on the current system, such as username, language settings, machine name, and atypical software installed. If our sandbox machine fails any of these checks we reveal ourselves to the attacker.

Another protection strategy is to attempt to identify that a sample is looking for evidence of a sandbox or automated analysis system, and only then to send the sample to the physical machine. This reduces the management overhead as we would only be sending the proportion of the samples that require it to a physical machine, and the rest can go through the virtualized setup which would in theory have much greater capacity. However, it then becomes extremely important to be able to identify every single possible sandbox detection technique in existence and new ones as they are developed, as if one technique is missed, the sandbox becomes vulnerable.

A third strategy is always to run a sample in both an unhardened virtual machine and a physical box, comparing any differences. This would flag up instances where the sample is behaving differently under analysis conditions from in the real world, but creates considerable resources overhead as each sample has to be executed twice.

CONCLUSION

Sandbox execution is increasingly touted as a possible solution for detecting 'day-zero' malware. There are a wide range of commercial and publicly available solutions that can be

incorporated into an organization's network defences or used as standalone research tools, and many security companies have their own in-house solutions used to process huge quantities of malware daily. Furthermore, the data that is extracted from analysis is far greater than a simple decision of 'good' or 'bad', as attempts are made to correlate the behaviour of one sample with that of another and to detect and block the common features of both, such as C&C addresses.

Virtualization is also a technology that is becoming increasingly widespread and is commonly seen on both the server and the desktop. Despite this fact, we continue to see more VM-aware malware that refuses to execute or will execute in a different way while virtualized. It seems that malware authors are willing to sacrifice some genuine installs on machines that are virtualized for the sake of the benefits that can be gained from concealing true functionality from researchers and security companies, and indeed, from presenting false and misleading information instead.

The examples shown in this paper indicate that we must be wary of trusting the output from a sandbox analysis, and show how important it is to ensure an artificial environment looks as much as possible like a real machine. Detecting that a sample or a whole malware client/server interaction is behaving differently can sometimes be easy but we have also shown cases where the differences are very subtle and difficult to identify.

REFERENCES

[1] http://www.sophos.com/en-us/support/knowledgebase/119112.aspx.

[2] http://invisiblethings.org/papers/redpill.html.

[3] https://media.blackhat.com/us-13/US-13-Singh-Hot-Knives-Through-Butter-Evading-File-Based-Sandboxes-Slides.pdf.

[4] Xu, H. https://www.virusbtn.com/virusbulletin/archive/2013/05/vb201305-Andromeda-botnet.

[5] http://www.0xebfe.net/blog/2013/03/30/fooled-by-andromeda/.

[6] http://blogs.technet.com/b/mmpc/archive/2013/09/10/msrt-september-2013-win32-simda.aspx.

[7] http://c-apt-ure.blogspot.co.uk/2013/12/ponmocup-hunter-is-re-tired.html.

[8] https://www.baesystemsdetica.com/services/cyber-security/the-shylock-malware/.

[9] https://www.virusbtn.com/conference/vb2013/abstracts/LM7-JiaBatchelder.xml.