Mass Malware Analysis:
A Do-It-Yourself Kit

www.cert.at <team@cert.at>

Christian Wojner <Wojner@cert.at>

October 14th, 2009

Version 1.0

Table of Contents

1 Introduction

This paper outlines the relevant steps to build up a customizable automated malware analysis station by using only freely available components with the exception of the target OS (Windows XP) itself. Further a special focus lies in handling a huge amount of malware samples and the actual implementation at CERT.at. As primary goal the reader of this paper should be able to build up her own specific installation and configuration while being free in her decision which components to use.

The first part of this document will cover all the theoretical, strategic and methodological aspects. The second part is focusing on the practical aspects by diving into CERT.at's automated malware analysis station closing with an easy to follow step-by-step tutorial, how to build up CERT.at's implementation for your own use. So feel free to skip parts.

NOTE: Do not treat any of the components of the finally built malware analysis station as trusted! Escapes can always happen, so position your malware analysis station wisely and handle it with care!

2 Motivation

Thinking about the triggers that made me building my own type of an "automated malware analysis station" (AMAS) it occurred to me that many others – especially in my business – might be in a comparable situation. So I decided to publish my best practice as a way which can be easily followed by everyone that is in the same need.

My trigger for such an AMAS is a project that needs some specific analysis of a not manually manageable quantity of malware samples. Regarding this my solution had to comply with the following requirements:

- *logging of registry-keys being read*
- *logging of files being read*
- *cheap*
- *fast*
- *as far as possible autonomic*
- *keep the analysis internal*

Actually the basic concept of my solution for these requirements – which I got running after just three days of work (including research) – meets much more needs. In details an AMAS following my concept meets the following requirements:

- *cheap*
- *runnable on a standard desktop system*
- *fully customizable*
- *target-system freely choose able*
- *easily evaluate able*
- *easy to set up*
- *ready for use as fast as possible*
- *as fast as possible*
- *as autonomic as possible*
- *runtime-unpacked/-decrypted samples must not be a hurdle*
- *data/analysis/samples/… kept confidential*
- *infections must be kept isolated from the public*

3 Fundamental Approach

The fundamental approach to solve a quantity-problem with machine-power is to analyze what you would do manually to fulfill the "exercise", bringing the resulting activity-chain to an automatable concept afterwards. That's just the way I started, so let's bring this manual-activity-chain into a big picture in the assumption of doing this for just one sample:

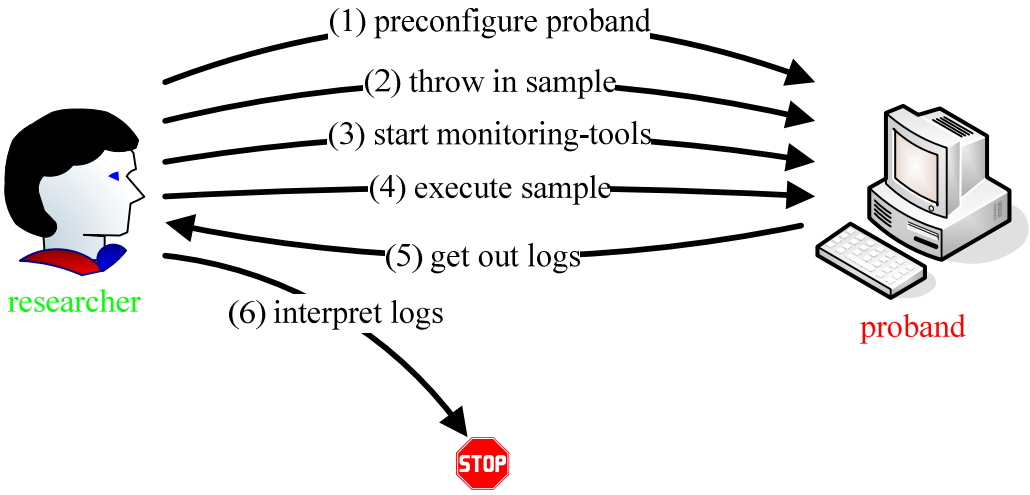


Figure 1

So the primary steps of one malware sample to be analyzed are quite easy to sum up:

1. *The proband[1] has to be preconfigured to meet all our desires (i.e. monitoring).*
2. *The sample to be analyzed has to be transferred to the proband.*
3. *The monitoring tools have to be started and confirmed to work properly.*
4. *Once monitoring is running the sample is being executed.*
5. *All logs that have been obtained need to be transferred back to the researcher.*
6. *The monitoring logs can now be interpreted in any desired way.*

Let's add the loop for analyzing more than one malware sample into the picture:

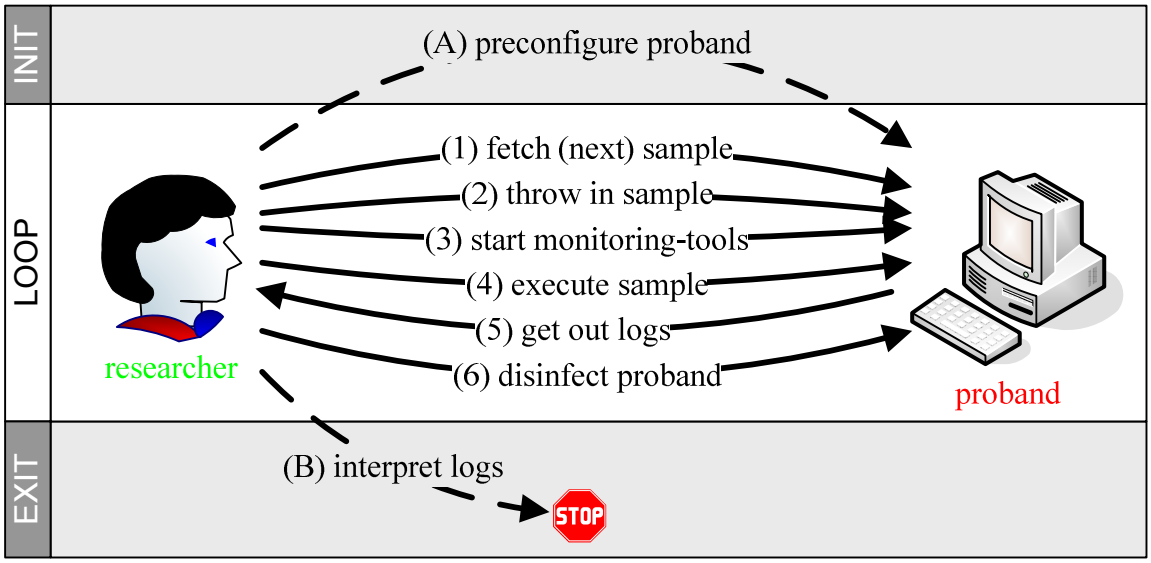


Figure 2

[1] Proband ... eq. guinea pig; but as I like guinea pigs I preferred calling the test machine „proband".

At a closer look we see that we had to split our working-steps in three phases:

- INIT … *activities that have to be done before looping starts*
- LOOP … *steps that are part of the core-functionality of analyzing one sample*
- EXIT … *actions that happen after all samples ran through monitoring*

At this point it should be noted that the interpretation of the monitoring logs can either happen **after the whole looping** until all samples have been monitored or as **last activity of each loop-cycle.**

The way *you* should do it depends on your case's circumstances. If you have a bulk of samples that you want to have monitoring logs for, be well advised to do the interpretation on exit; just as it's shown in our picture. But on the other hand, especially if you have an unpredictable quantity of samples (because of them being uploaded by people using your website i.e.) or you are in the need of instantaneous reports, the right way for you is to do the interpretation as last action of each loop-cycle, regardless of the periodicity of samples available.
However, as already noted, the approach in this paper (as in CERT.at's AMAS) focuses on studies/research and therefore statistics, which are logically done after all of the available malware samples have been monitored. This gives you full freedom in decision which evaluation should be next without the need to repeat monitoring over and over again every time you're interested in something new. So the automation of monitoring thousands of samples is our primary goal. But even if you need some kind of i.e. web based business-service the relating adjustments should be easy.

As Figure 2 shows, the action of preconfiguring the proband moved up to the initial phase of our whole working process, because it does not make sense doing this over and over again. Therefore, a new action item got added, the disinfection of the proband. At the end of every loop cycle our proband has to be disinfected, otherwise the next samples could execute in an improper way and our logs would probably become meaningless. A reliably method of disinfecting our proband is shown in the next section.

4 Identifying Components

Now that we have a distinct view on all the steps our AMAS will be able to perform, let's take a close look at the components it needs to have to work properly.

What firstly occurs to us is that we have two "subjects" or in better words "locations" where actions will take place:

- *the researcher's side and*
- *the proband's side.*

On the researcher's side there are all of the steps regarding the management of the samples, the receipt of the monitoring-logs and the control of the proband's state (i.e. disinfection).
On the proband's side there are the steps of receiving, executing and handling a sample. Furthermore we have a step that pushes (saves) the monitor-logs back to the researcher.
As combining these two locations in one physical location simplifies our pursuit of building an AMAS, the primary characteristics of virtual machines come in handy for that. I would recommend using a virtual machine (guest) for the proband's side itself and the native machine (host) for the researcher's side.

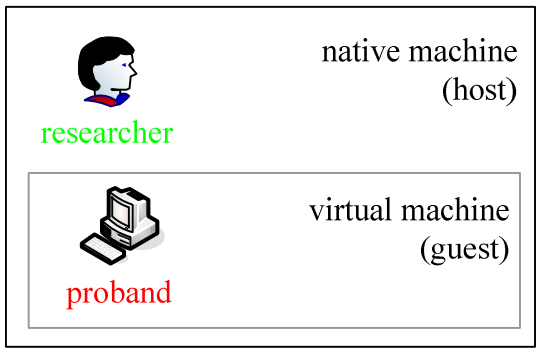


Figure 3

Further observations of Figure 2 lead to the discovery that any of the currently declared locations has some "autonomous entity" that is able to manage, transfer, etc. things. So let's add control processes (cpr = control process researcher, cpp = control process proband):

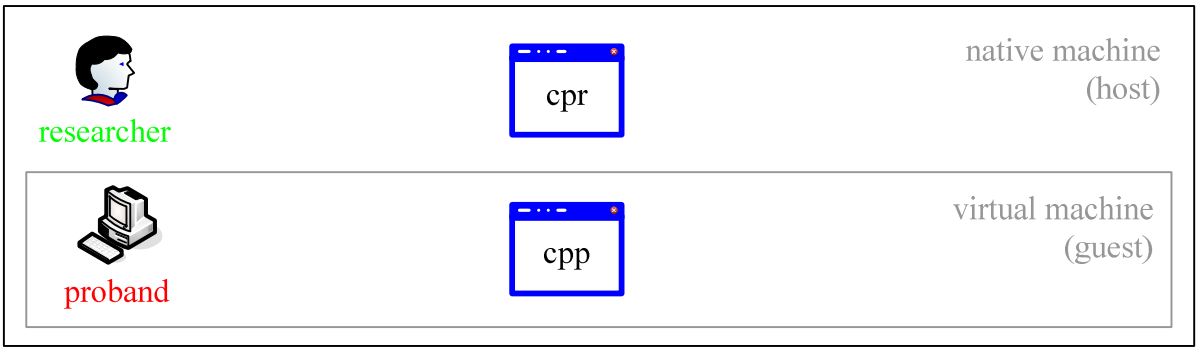


Figure 4

As our latest conclusion made its way to the picture there are still components that need to be declared. The control processes need a way to communicate with each other. Therefore - to avoid a reinvention of the wheel - we'll choose one of the easiest protocols around … FTP. As FTP (just like other protocols) always likes to have a server and at least one client in the game let's pick the native machine for server- and the virtual machine for client-purpose (you can also use the proband as server and the researcher as client, but as Windows XP has no ftp-server included by default the way we plan it just needs fewer thing to do).

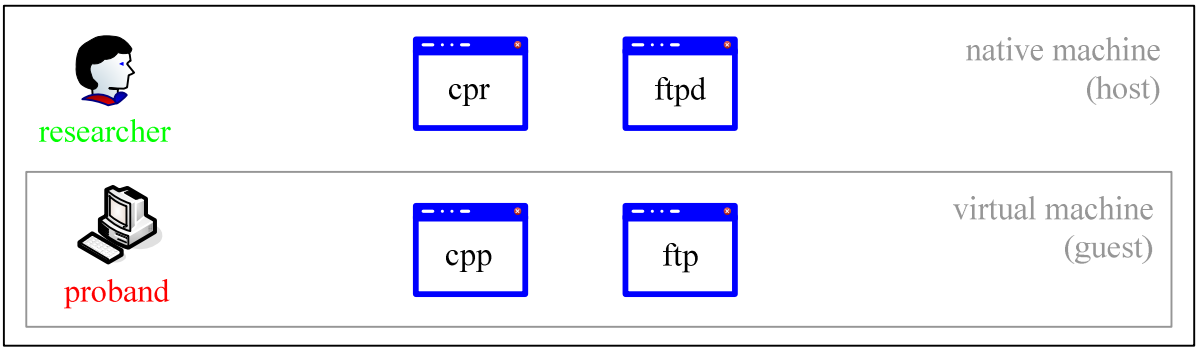


Figure 5

For saving reports and providing data in terms of samples there's a need for two collections:

- *the malware Samples and*
- *the monitoring logs archive.*

Adding them to the picture will lead to the following:

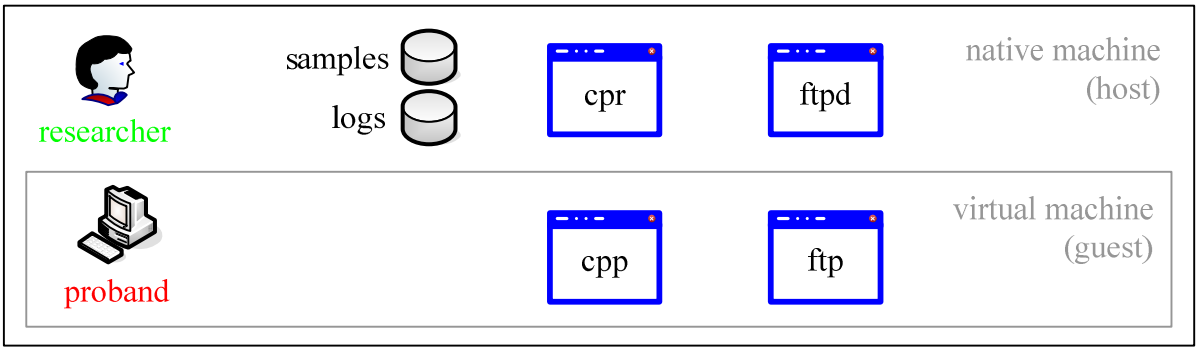


Figure 6

4.1 Defining communications

Finally, the identified components need to communicate with each other:

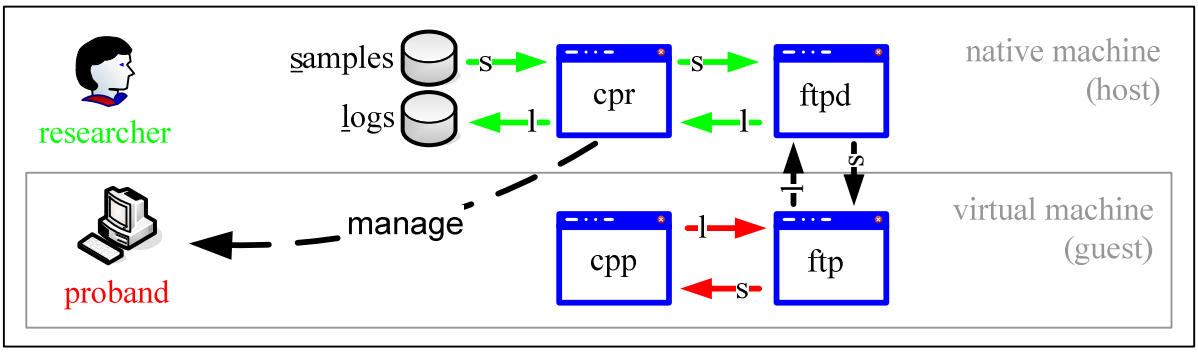


Figure 7

As Figure 7 shows, FTP is used for both transfer-goals:

- *the samples and*
- *the monitoring-logs.*

5 Problems of Automation

A typical problem of automation is synchronization between the host control process and the proband control process.

"Timing" is the magic word. Most of the problems you will encounter (or at least I did) have more or less a nexus to timing. So in other words, there's always one process waiting for the other. And when we talk about waiting, it's "how long" which comes to our minds. Having said this you can trust me when I say, that you can't define events (that trigger the next step) in every possible kind of situation. Sometimes you just have to have timeouts. And actually it's the decision of the timeout values that have direct impact to efficiency, speed and even quality/integrity regarding the monitoring-logs. So the timeout values are the real brains of any AMAS you're planning to build, because the smaller your timeouts are the shorter the time per sample and the higher the risk for missing malware activity.

Besides that you have to realize that these two processes we want to synchronize aren't on the same machine – remember, one runs on the native machine and the other in the virtual machine. Let's reveal a solution for that problem. What again comes in handy here is FTP between those two machines (and processes). The existence of files on the server side is used to synchronize.

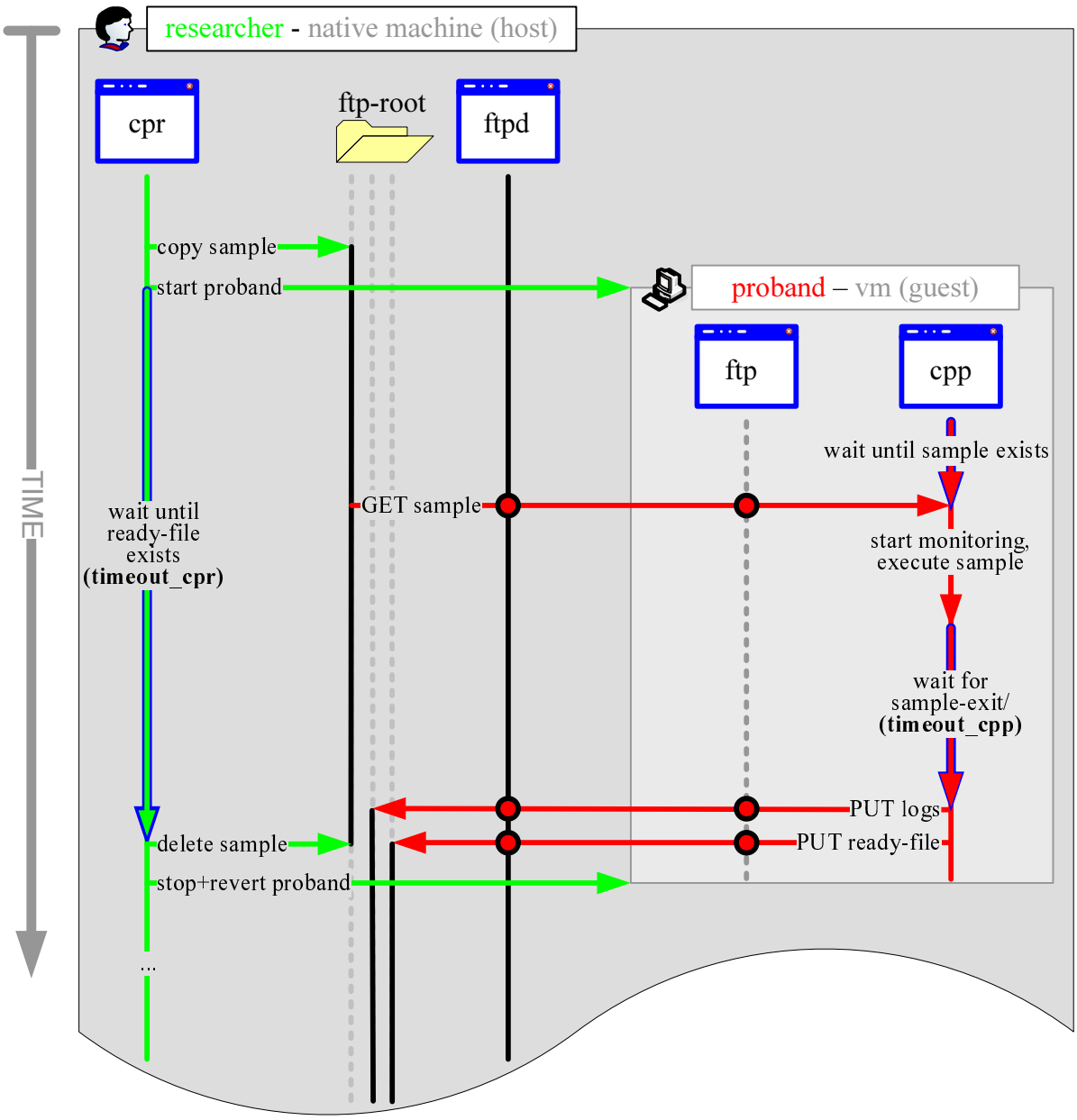


Figure 8

Looking at this diagram and giving the first visual shock some time to disappear we see, that we have two positions where we should define timeouts (coherently named **timeout_...**).

Let's focus on the timeout of the researcher's control process (timeout_cpr). This timeout is really some kind of safety-break. It exists only for saving our automatism from endless looping if something weird happens in the virtual machine. That could be i.e. when the executed malware sample crashes the OS of the virtual machine, shuts it down or reboots it. But actually there are a lot more things that can navigate the virtual machine in an unpredictable state. In all these cases our safety-break (timeout_cpr), will save our day by acting as if the expected trigger-condition had occurred.

The timeout of the proband's control process (timeout_cpp) exists for letting the executed malware sample have a maximum amount of time until the monitoring stops and the researcher's control process (cpr) of the native machine is being sent its (triggering) files.

So, as I said before, the values of the timeouts are of heavy influence and setting them in an unwise way could in the worst case ruin our whole automation.

6 Controlling the Proband

Since I haven't told anything about the magic "controlling the proband on the whole", or in other words, how to deal with the virtual machine('s state) itself, we will now take a close look behind these scenes.

Usually any virtual machine product has the ability to manage the virtual machines via command-line tools. Actually there are so many gears and switches that can be altered via command-line tools that it can sometimes confuse you. But, the good news is that the things we need are relatively elementary and should be easily figured out. In CERT.at's implementation I used Sun's VirtualBox which is one virtual machine, which the most of malware defense-techniques don't care about. However, feel free to choose any other alternative of your taste (i.e.: VMware, QEMU, …), but for now I'll just focus on VirtualBox.

VirtualBox has a handful of decent command-line tools though the mightiest one of them is "VBoxManage". Actually VBoxManage is all we need for our AMAS – as long as we chose VirtualBox. Let's take a look what VBoxManage can do for us and how it's told to …

```
VBoxManage list vms
```

You always start your command with a heading "VBoxManage" getting more specific with every word that follows. So this example causes all virtual machines managed by VirtualBox being listed. Each of the reported virtual machines has a unique identifier which is called UUID. Anytime you want to handle "your" virtual machine you do this using the UUID, so you should firstly identify yours.

```
VBoxManage startvm uuid
```

The command above will startup the machine *uuid* if it's not running, otherwise you get an error.

```
VBoxManage controlvm uuid poweroff
```

This removes the power from the virtual machine *uuid* as if you pulled out the power cable out of a physical machine.

```
VBoxManage snapshot uuid discardcurrent -state
```

The one above is already the last command that's of interest to us. It trashes the actual state of the virtual machine and brings the latter back to its last saved state, which should be our clean start point. So, in other words, anything (including infections) that occurred since the last startup of the virtual machine *uuid* is being "forgotten".

7 CERT.at's Implementation

Now that you have all the basic information to build up an AMAS on your own let's take a look at a practical example – the AMAS of CERT.at.

First of all, I know that our (CERT.at's) AMAS could do much more than it actually does, but if you followed this paper closely you know that the abilities of the proband's process could be enhanced at all times keeping efforts low.
CERT.at's AMAS has the name Minibis which I chose because of its lightweight characteristics in correlation to "Anubis" which is one of the big automatic binary analyzers.

7.1 AMAS Setup

7.1.1 Hardware

- *Dell OPTIPLEX 745*
- *Intel Core 2 Duo 6400*
- *2 Gigs of RAM*

Notes:
What we have here is an average Desktop machine.

7.1.2 Researcher's Software

- *Xubuntu 8.04 (hardy)*
- *SUN's VirtualBox*
- *proftpd*
- *zip*
- *minibis-cpr*

Notes:
Of course there is a lot more software around brought by Xubuntu but I just focused the parts that are in direct need for the AMAS.
What is **minibis-cpr**? That's the researcher's process that manages the VM and all the communications with the proband's process. It's a binary executable which is written in Purebasic, my favorite programming language when I want to have results in a minimum of time. Feel free to use your own favorite language.

7.1.3 Proband's Software

- *Microsoft Windows XP SP3*
- *ProcessMonitor (from Sysinternals)*
- *minibis-cpp*

Notes:
As the proband's control process just has to receive a sample and start it, its startup-state (which is gone back to by reverting) has the proband's process **minibis-cpp** already running, asking for a sample over FTP. So just before the proband-VM is started, all files it wants to retrieve have to be in their place. It also manages the contact with ProcessMonitor, catches a screenshot on exit and does the communication-stuff regarding FTP using Window's own FTP-client.
ProcessMonitor from Sysinternals was the best solution for the information we wanted to gather from our samples. It makes Registry-, File- and most relevant API-activities transparent which can be saved as CSV-file.

7.2 Big picture

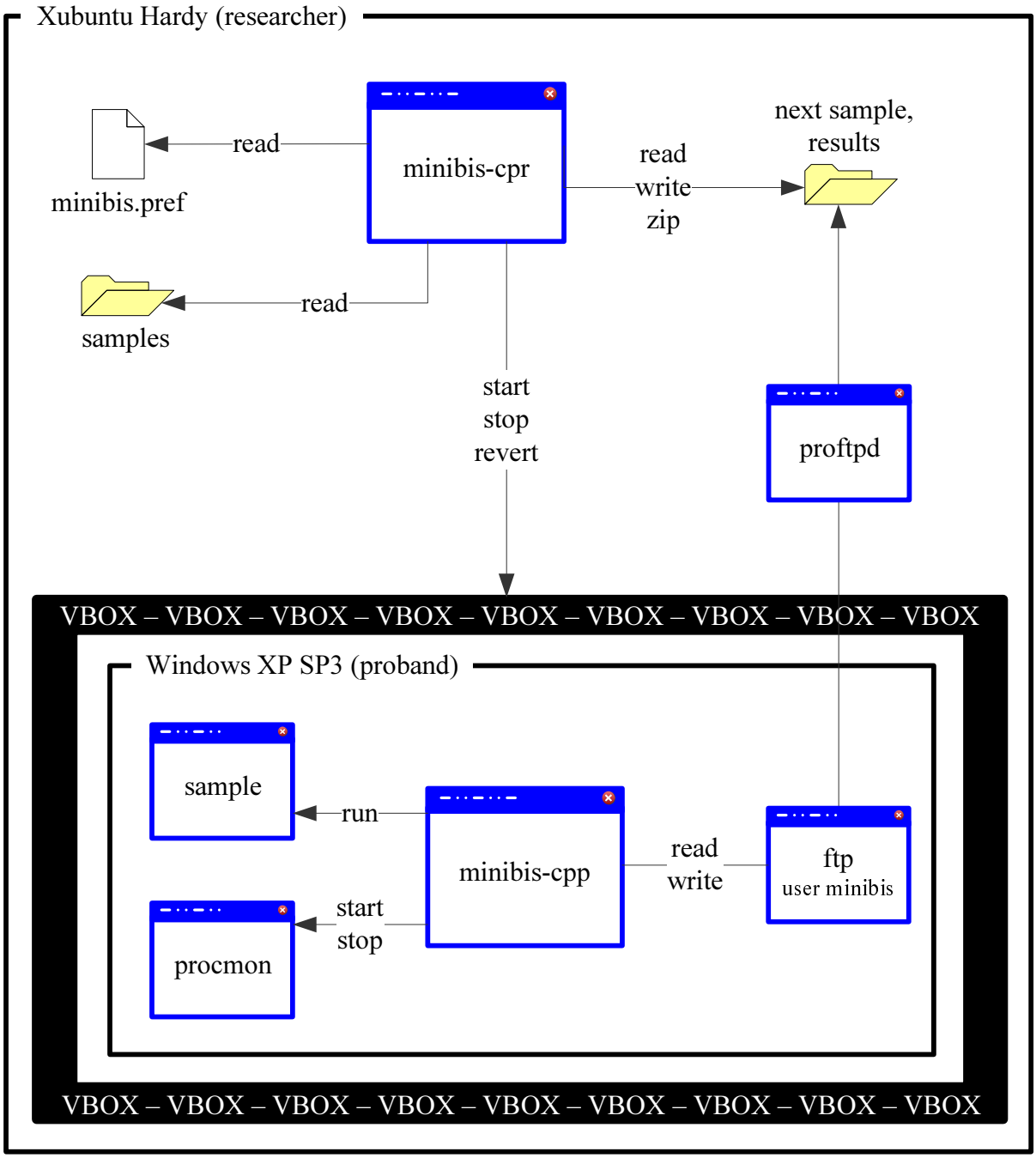


Figure 9

7.3 Executables, Sources and Details

Our researcher's and proband's control processes are all written in Purebasic. Knowing that Purebasic might not be the first decision of everyone I chose it anyway because as I already said, it's my favorite programming language. It's efficient, powerful, crossplattform and also very easy to read and understand, even for people not used to this "Basic-dialect". Actually, compared to scripting languages like Perl for example, I think – besides the fact that Purebasic programs are compiled to binary executables – that someone being not used to a certain programming language but being presented a code-listing of that, might be facing an easier task when this certain language is that readable like Basic-syntax.

7.3.1 Researcher's Control Process (minibis-cpr)

7.3.1.1 Description

minibis-cpr is a Linux-binary. It iterates through all samples transferring one by one to the proband waiting after each transfer for the results of the proband to receive. It also handles starting, stopping and reverting the VM.
minibis-cpr is configured by a file called "minibis.pref" which is an INI-file and is expected to be located in the home-directory of the actual user. However, when it's not existing minibis-cpr creates one using default-values. As minibis-cpr can't predict what the attributes of minibis-cpr's use will be, this default-values – especially the ones regarding paths and VM-id (uuid, remember?) – won't perfectly fit. Though, by having this default-values, you should have an easier job doing the right configuration afterwards.

As "minibis.pref" is also transferred to the proband itself it carries the configuration of all components regarding our AMAS.

Let's take a look at the parameters in "minibis.pref" …

Group [status]

- last … *the last sample that has been analyzed (position to start after on rerun)*
- quit … *switch to halt minibis-cpr in a controlled way*

Group [cpr]

- ftp_path … *ftp-user's home*
- vmid … *this is the uuid of the proband virtual machine*
- find … *the parameters of the find-command that lists all samples*
- dlls … *also use dll-based samples?*
- zip … *which file-extensions of the result-files shall be archived*
- timeout_result … *emergency break for the result-files latency in seconds*
- timeout_stopvm … *emergency break for the vm-stopping latency in seconds*
- timeout_revertvm … *emergency break for the vm-reverting latency in seconds*

Group [cpp]

- timeout_sample … *emergency break for the sample latency in seconds*
- time_after_good_exit … *extra-seconds after sample has exited in case of injections*

<u>Important:</u> As minibis-cpr writes files in the ftp-user's home the user that runs it needs to have the appropriate permissions there!

7.3.1.2 Download

If you are interested in using our implementation of an AMAS feel free to download minibis-cpr from our server:

http://www.cert.at/static/downloads/minibis/minibis-cpr

7.3.1.3 Source Code (minibis-cpr.pb)

```
Global md5.s = ""
Global sample.s = ""
Global ftp_path.s = ""
Global vmid.s = ""
Global find.s = ""
Global dlls.s = ""
Global timeout_result.l = 0
```

```
Global timeout_stopvm.l = 0
Global timeout_revertvm.l = 0
Global zip.s = ""
Global fnd.l = 0
Global last.s = ""

; if preferences-file is not existing create one
If (FileSize(GetHomeDirectory() + "minibis.pref") < 0)
    CreatePreferences(GetHomeDirectory() + "minibis.pref")

    PreferenceGroup("status")
    WritePreferenceString("last", "")
    WritePreferenceString("quit", "0")

    PreferenceGroup("cpr")
    WritePreferenceString("ftp_path", "/home/minibis/")
    WritePreferenceString("vmid", "ff90d65f-f75b-4f70-997f-71e83155348b")
    WritePreferenceString("find", "/media/cdrom0 -type f")
    WritePreferenceString("dlls", "yes")
    WritePreferenceString("timeout_result", "100")
    WritePreferenceString("timeout_stopvm", "60")
    WritePreferenceString("timeout_revertvm", "60")
    WritePreferenceString("zip", "exe,dll,pml")

    PreferenceGroup("cpp")
    WritePreferenceString("timeout_sample", "60")
    WritePreferenceString("time_after_good_exit", "10")

    ClosePreferences()
EndIf

; read preferences to know what I should do and how I should behave
OpenPreferences(GetHomeDirectory() + "minibis.pref")
PreferenceGroup("status")
last = ReadPreferenceString("last", "")
PreferenceGroup("cpr")
ftp_path = ReadPreferenceString("ftp_path", "")
vmid = ReadPreferenceString("vmid", "")
find = ReadPreferenceString("find", "")
dlls = ReadPreferenceString("dlls", "")
timeout_result = ReadPreferenceLong("timeout_result", 0)
timeout_stopvm = ReadPreferenceLong("timeout_stopvm", 0)
timeout_revertvm = ReadPreferenceLong("timeout_revertvm", 0)
zip = ReadPreferenceString("zip", "")
ClosePreferences()

; forward to the next sample in relation to previous run
fnd = RunProgram("find", find, "", #PB_Program_Open | #PB_Program_Hide | #PB_Program_Read)
While (last <> "" And last <> sample)
    sample = ReadProgramString(fnd)
Wend

; compute samples iterative
While (ProgramRunning(fnd))
    ; retrieve sample-path
    sample = ReadProgramString(fnd)

    ; check filetype of sample
    prg.l = RunProgram("file", sample, "", #PB_Program_Open | #PB_Program_Hide | #PB_Program_Read)
    type.s = ReadProgramString(prg)
    type = Mid(StringField(type, 2, ":"), 2)
    CloseProgram(prg)

    ; get MD5 fingerprint of sample
    md5 = MD5FileFingerprint(sample)

    ; continuing just for "relevant" samples
    If (FindString(type, "for MS Windows", 1) > 0 And ((Not (dlls = "no" And FindString(type, "(DLL)", 1) >
1)) Or (dlls = "yes")))
        ; continue analyzing the actual sample only if it hasn't been already analyzed, otherwise skip it
        If (FileSize("/home/minibis/" + md5 + ".rdy") <= 0)
            ; copy necessary files to ftp-path and print some information to commandline
            PrintN("********************************************************************************")
            CopyFile(sample, ftp_path + "abcd")
            CopyFile(GetHomeDirectory() + "minibis.pref", ftp_path + "abcd.pref")
            CreateFile(9, ftp_path + "abcd.type")
            If (FindString(type, "(DLL)", 1) > 1)
                WriteStringN(9, "dll")
            Else
                WriteStringN(9, "exe")
            EndIf
            CloseFile(9)
            PrintN("Analysing: " + sample)
            PrintN("MD5:       " + md5)
            PrintN("Filetype:  " + type)
            PrintN("================================================================================")

            ; start up the proband-VM
            PrintN("Starting VM")
```

```
            RunProgram("VBoxManage", "startvm " + vmid, "", #PB_Program_Wait)

            ; ... and enter the loop of waiting for the results including an emergency break
            Print("Waiting for result ")
            steps.l = 0
            Repeat
                Print(".")
                Delay(1000)
                steps + 1
            Until (FileSize(ftp_path + md5 + ".rdy") > 0 Or steps > 100)
            PrintN("")
            If (steps > 100) ; emergency break used?
                PrintN("Something strange happened! ... I won't do this sample again ;-(")
            EndIf

            ; try to stop the proband-VM
            PrintN("Stopping VM")
            stop.l = RunProgram("VBoxManage", "controlvm " + vmid + " poweroff", "")
            j.l = 0
            Repeat
                j + 1
                Delay(1000)
            Until (Not IsProgram(stop) Or j >= 60)
            ; ... with force if necessary
            If (IsProgram(stop))
                KillProgram(stop)
            EndIf
            ; ... and clean up any garbage
            RunProgram("killall", "-9 VirtualBox", "", #PB_Program_Wait)
            RunProgram("killall", "-9 VBoxXPCOMIPCD", "", #PB_Program_Wait)
            RunProgram("killall", "-9 VBoxSVC", "", #PB_Program_Wait)

            ; revert proband-VM
            PrintN("Reverting VM")
            revert.l = RunProgram("VBoxManage", "snapshot " + vmid + " discardcurrent -state", "",
#PB_Program_Wait)
            j.l = 0
            Repeat
                j + 1
                Delay(1000)
            Until (Not IsProgram(revert) Or j >= 60)
            ; ... again with force if necessary
            If (IsProgram(revert))
                KillProgram(revert)
            EndIf

            ; compress spacehungry resultfiles that don’t need to be directly readable in post-activities
            If (steps <= 100)
                PrintN("Zipping")
                DeleteFile(ftp_path + md5 + ".zip")
                i.l = 0
                Repeat
                    i + 1
                    If (StringField(zip, i, ",") = "")
                        Break
                    EndIf
                    zippar.s = md5 + ".zip " + md5 + "." + StringField(zip, i, ",")
                    RunProgram("zip", zippar, ftp_path, #PB_Program_Wait)
                    DeleteFile(ftp_path + md5 + "." + StringField(zip, i, ","))
                ForEver
            EndIf

            ; save the last position in iteration for eventual breaks
            OpenPreferences(GetHomeDirectory() + "minibis.pref")
            PreferenceGroup("status")
            WritePreferenceString("last", sample)
            If (ReadPreferenceLong("quit", 0) = 1)
                End
            EndIf
            ClosePreferences()

            ; clean up last time - sometimes it's necessary, dunno why
            RunProgram("killall", "-9 VirtualBox", "", #PB_Program_Wait)
            RunProgram("killall", "-9 VBoxXPCOMIPCD", "", #PB_Program_Wait)
            RunProgram("killall", "-9 VBoxSVC", "", #PB_Program_Wait)

            ; deleting trigger-files
            DeleteFile(ftp_path + "abcd")
            DeleteFile(ftp_path + "abcd.pref")
            DeleteFile(ftp_path + "abcd.type")
        Else
            PrintN(md5 + ": Already done")
        EndIf
    Else
        PrintN(md5 + ": Wrong filetype (" + type + ")")
    EndIf
Wend
```

7.3.2 Proband's Process (minibis-cpp)

7.3.2.1 Description

minibis-cpp.exe is a Windows-binary. It fetches all files named like "abcd.*" from the FTP-server, brings up ProcessMonitor, runs the sample, handles controlled detaching from it and uploads the result-files. Besides using ProcessMonitor for the monitoring task minibis-cpp.exe also is able to take a screenshot of the last state before it "detaches" from the sample.
minibis-cpp is configured by a file called "abcd.pref" which is just a copy of "minibis.pref" that is transferred through ftp (put there by minibis-cpr).
For information regarding the parameters in "minibis.pref" please take a look at 7.3.1.1.

7.3.2.2 Download

If you are interested in using our implementation of an AMAS feel free to download minibis-cpp from our server:

http://www.cert.at/static/downloads/minibis/minibis-cpp.exe

7.3.2.3 Source Code (minibis-cpp.pb)

```
UsePNGImageEncoder()

Procedure runSample(par.l)
    If (par = 0)
        prg.l = RunProgram("c:\abcd.exe", "", "")
    Else
        prg.l = RunProgram("rundll32.exe", "c:\abcd.dll", "")
    EndIf
    While (IsProgram(prg))
        Delay(1000)
    Wend
EndProcedure

Procedure makeScreenshot()
    CreateImage(0,GetSystemMetrics_(#SM_CXSCREEN),GetSystemMetrics_(#SM_CYSCREEN))
    DC = StartDrawing(ImageOutput(0))
    BitBlt_(DC,0,0,ImageWidth(0),ImageHeight(0),GetDC_(GetDesktopWindow_()),0,0,#SRCCOPY)
    StopDrawing()
    SaveImage(0, "c:\abcd.png", #PB_ImagePlugin_PNG)
EndProcedure

md5.s = ""

OpenConsole()

PrintN("Waiting for sample")

; prepare command-sequence-file for downloading all relevant files over FTP
file.l = CreateFile(#PB_Any, "c:\ftp.txt")
WriteStringN(file, "open 10.0.2.2")
WriteStringN(file, "minibis")
WriteStringN(file, "minibis")
WriteStringN(file, "binary")
WriteStringN(file, "get abcd")
WriteStringN(file, "get abcd.type")
WriteStringN(file, "get abcd.pref")
WriteStringN(file, "close")
WriteStringN(file, "bye")
CloseFile(file)

; try to download until the sample is here
Repeat
    RunProgram("ftp", "-s:c:\ftp.txt", "", #PB_Program_Wait)
    Delay(1000)
Until (FileSize("abcd") > 0)

; open configuration
OpenPreferences("abcd.pref")
PreferenceGroup("cpp")

; get file-type
ReadFile(9, "abcd.type")
type.s = ReadString(9)
CloseFile(9)

; copy file to c-root for easier handling and calculate md5 for later use
CopyFile("abcd", "c:\abcd")
```

```
md5 = MD5FileFingerprint("c:\abcd")
CopyFile("c:\abcd", "c:\abcd." + type)

; start up ProcessMonitor
PrintN("Starting procmon")
RunProgram("procmon.exe", "/Backingfile c:\abcd.log", "")
RunProgram("procmon.exe", "/WaitForIdle", "", #PB_Program_Wait)
Delay(1000)

; run sample
PrintN("Starting sample")
If (type = "exe")
    thread.l = CreateThread(@runSample(), 0)
Else
    thread.l = CreateThread(@runSample(), 1)
EndIf
stamp.l = ElapsedMilliseconds()

; waiting for sample till it exits or emergeny break comes up
Print("Waiting for sample to end or timeout ")
steps.l = 0
While (IsThread(thread) And steps < ReadPreferenceLong("timeout_sample", 0))
    Print(".")
    Delay(1000)
    steps + 1
Wend

; if not emergency break ...
; ... give some extra time for still recording activities in case of injection
If (Not IsThread(thread) And steps < ReadPreferenceLong("timeout_sample", 0))
    steps.l = 0
    While (steps < ReadPreferenceLong("time_after_good_exit", 0))
        Print("+")
        Delay(1000)
        steps + 1
    Wend
EndIf
PrintN("")

; controlled stop of ProcessMonitor
Print("Terminating procmon ")
RunProgram("procmon.exe", "/Terminate", "")
Repeat
    Print(".")
    file = OpenFile(#PB_Any, "c:\abcd.log.PML")
    Delay(1000)
Until (file)
CloseFile(file)
PrintN("")

; catch screenshot
PrintN("Making screenshot")
makeScreenshot()

; convert ProcessMonitor's format to human readable (CSV)
PrintN("Converting pml to csv")
RunProgram("procmon.exe", "/saveas c:\abcd.csv /openlog c:\abcd.log.PML", "", #PB_Program_Wait)

; transfer all result-files back via FTP
file = CreateFile(#PB_Any, "c:\ready.txt")
WriteStringN(file, "Ready!")
CloseFile(file)
RenameFile("c:\abcd.log.PML", "c:\" + md5 + ".pml")
RenameFile("c:\abcd", "c:\" + md5 + "." + type)
RenameFile("c:\abcd.png", "c:\" + md5 + ".png")
RenameFile("c:\abcd.csv", "c:\" + md5 + ".csv")
RenameFile("c:\ready.txt", "c:\" + md5 + ".rdy")
file = CreateFile(#PB_Any, "c:\ftp.txt")
WriteStringN(file, "open 10.0.2.2")
WriteStringN(file, "minibis")
WriteStringN(file, "minibis")
WriteStringN(file, "binary")
WriteStringN(file, "put c:\" + md5 + ".pml")
WriteStringN(file, "put c:\" + md5 + ".csv")
WriteStringN(file, "put c:\" + md5 + "." + type)
WriteStringN(file, "put c:\" + md5 + ".png")
WriteStringN(file, "put c:\" + md5 + ".rdy")
WriteStringN(file, "close")
WriteStringN(file, "bye")
CloseFile(file)
RunProgram("ftp", "-s:c:\ftp.txt", "", #PB_Program_Wait)

PrintN("Ready!")
Input()

CloseConsole()

End
```

7.4 Setting it up Step by Step

Ok, for those being a little bit shy in finding their way through installations and configurations here comes a chronologically step-by-step walkthrough to build up CERT.at's AMAS.
I admit that not all of the following steps are carved into stone regarding their chronological occurrence but … hey, are you shy or not? So, let's dive into this …

1. Select the physical machine that shall become the hull of your AMAS.
2. Install the latest version of Xubuntu on it.
3. Install proftpd (via "`apt-get install proftpd`").
4. Install zip (via "`apt-get install zip`").
5. Create a user "minibis" (password "minibis").
6. Give your own user, with which you will start "minibis-cpr", full permissions to the home of "minibis" and verify that you can write to "`/home/minibis`".
7. Download "minibis-cpr" to your own user's home.
8. Install SUN's VirtualBox (via "`apt-get install virtualbox`").
9. Create a new virtual machine (VM) in it using Windows XP as operating-system. All default settings for the machine and the OS are fine for us. Decline Autoupdate features when you get asked.
10. Start up the VM if it's not already running.
11. Disconnect the CD-Rom via the menu-bar (this is necessary to prevent unwanted popups to occur).
12. Download "http://www.cert.at/static/minibis/minibis-cpp.exe" and save it to the desktop.
13. Download "http://download.sysinternals.com/Files/ProcessMonitor.zip" and unzip it to the desktop.
14. Disconnect from the Internet/Intranet i.e. by unplugging the network cable!
15. Check out if you can connect to the host ftp-daemon by using Window's ftp-client. If any problems occur, solve them.
16. Start procmon once and accept EULA. Close procmon after that.
17. Execute "minibis-cpp.exe" in the VM and answer the firewall question to NOT BLOCK this application.
18. Create a VM-snapshot of this state.
19. Close the VM using the option to revert to the last taken snapshot.
20. Bring your samples into Xubuntu's filesystem (i.e. by mounting a CD-Rom).
21. Figure out the uuid of your just built VM using "`VBoxManage list vms`" in console (Note: Snapshots do also have an uuid so select the one uuid that belongs to the virtual machine itself!).
22. Execute "minibis-cpr" once in the console to get the example preferences-file and cancel it by pressing `CTRL+C`.
23. Adjust the preference-file ("minibis.pref") which you find under your user's home by overwriting the value of the key vmid with the recently found uuid.

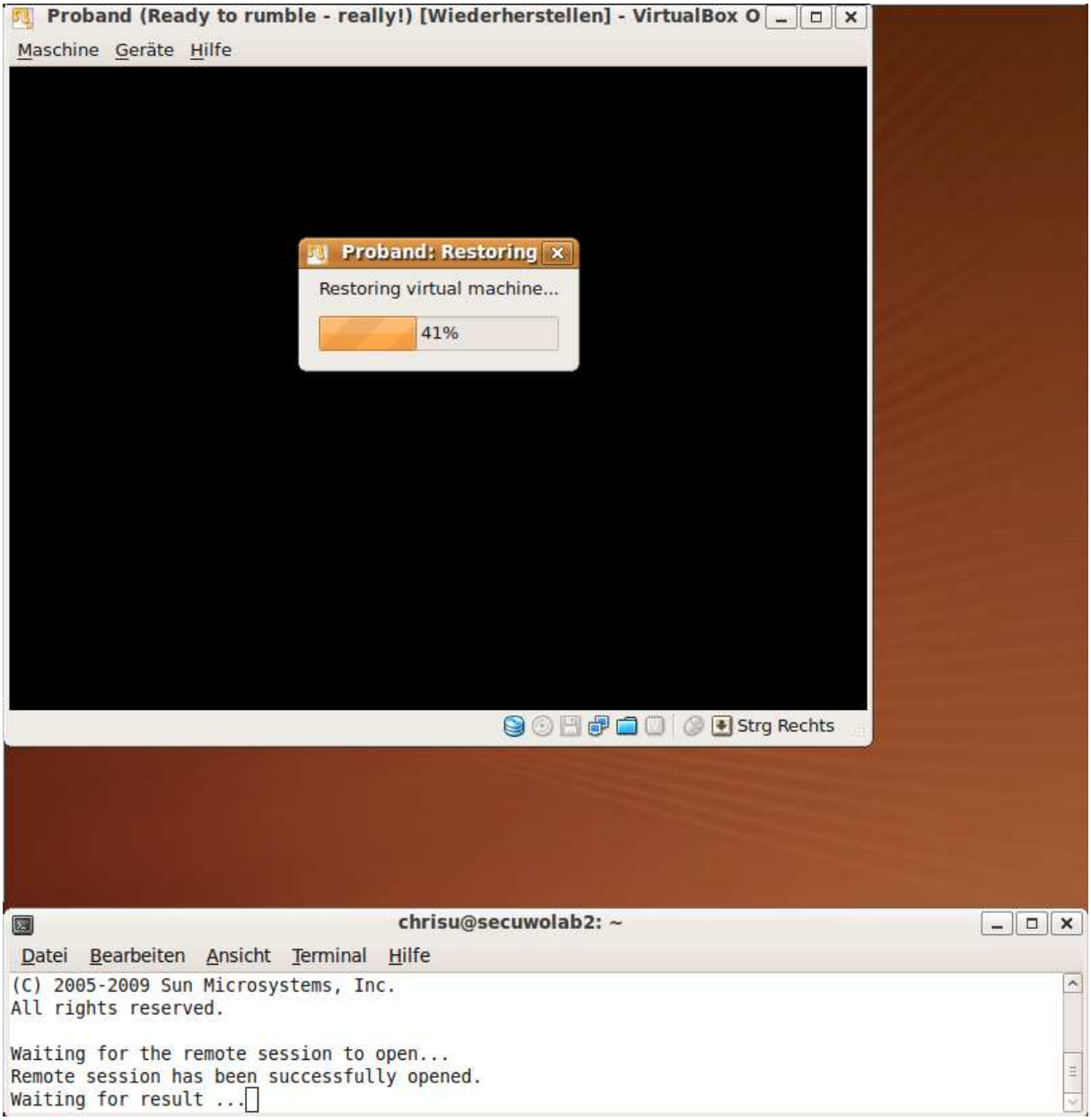
Now we're ready to go!

24. Start up "minibis-cpr" once again in the console.
25. Lay back and enjoy watching the show.

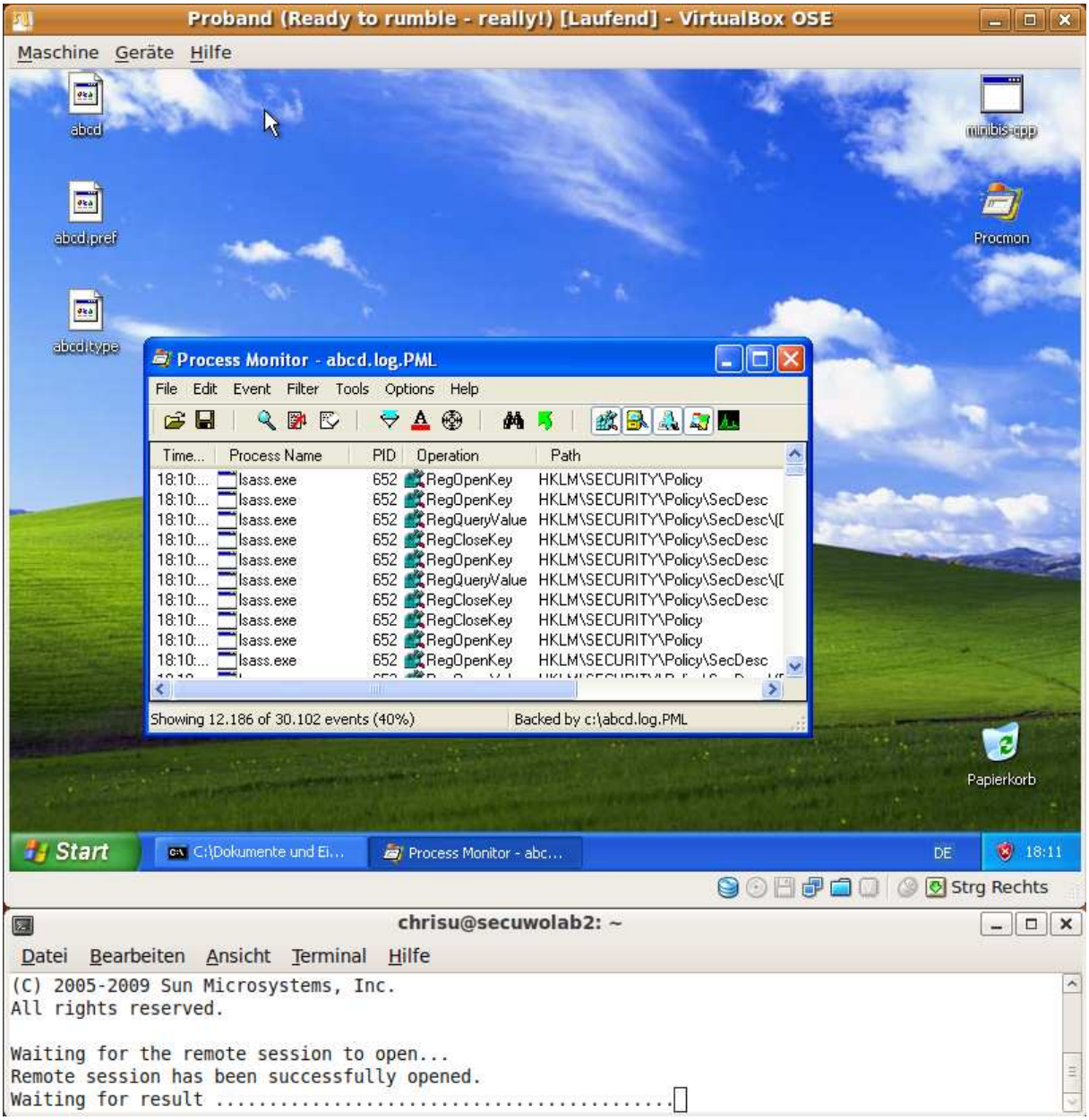
7.5 Problems

The only problems I encountered through analyzing 15.000 samples had something to do with the virtual machine being in some strange state or VirtualBox-daemons zombieing around. But all of them were solvable by me, myself and I … and I'm not a VirtualBox-pro.

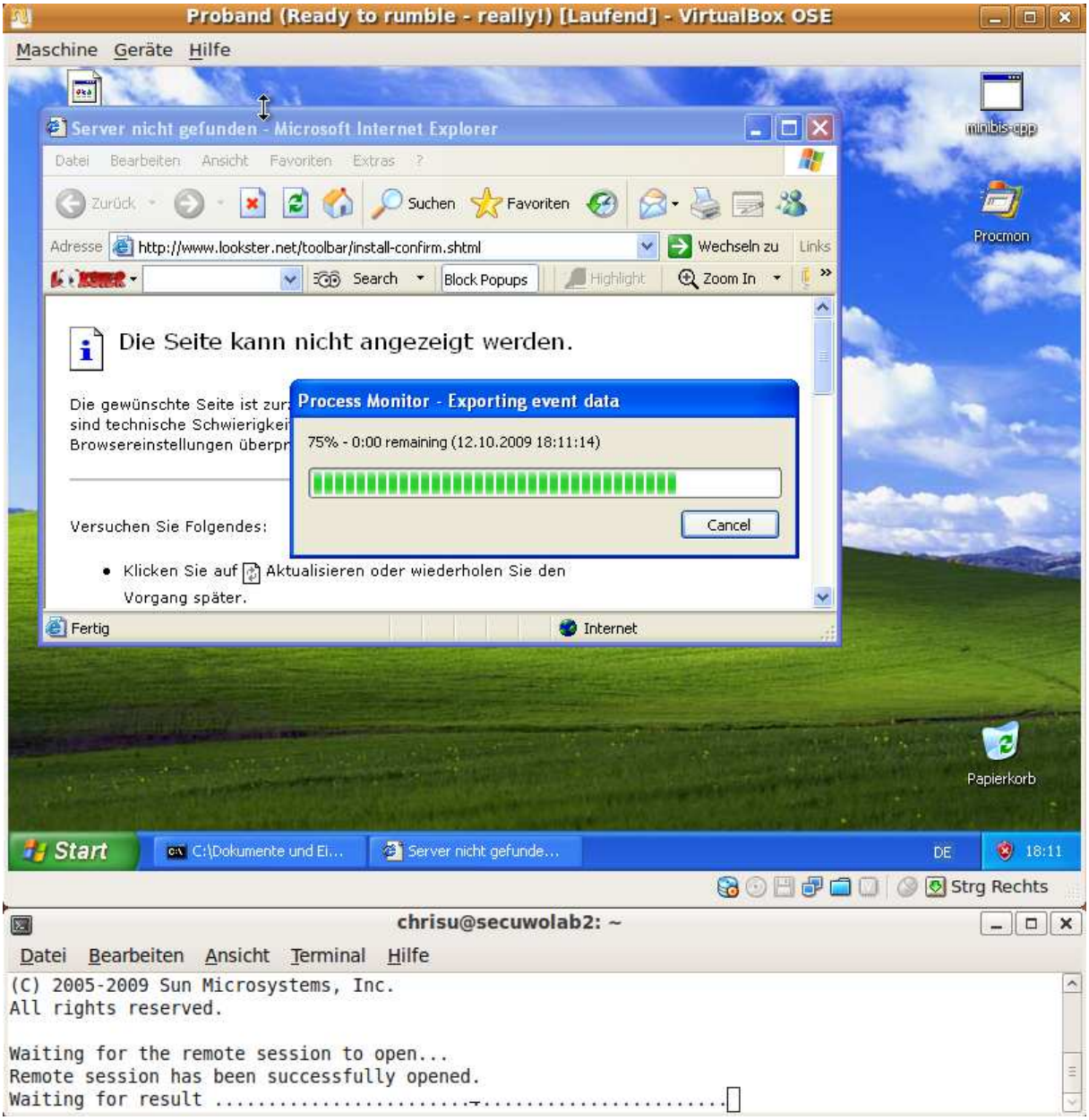
8 Screenshots



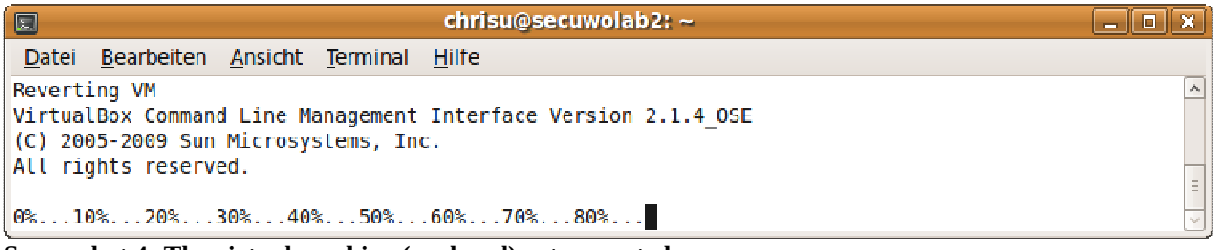
Screenshot 1: The virtual machine (proband) is being started.



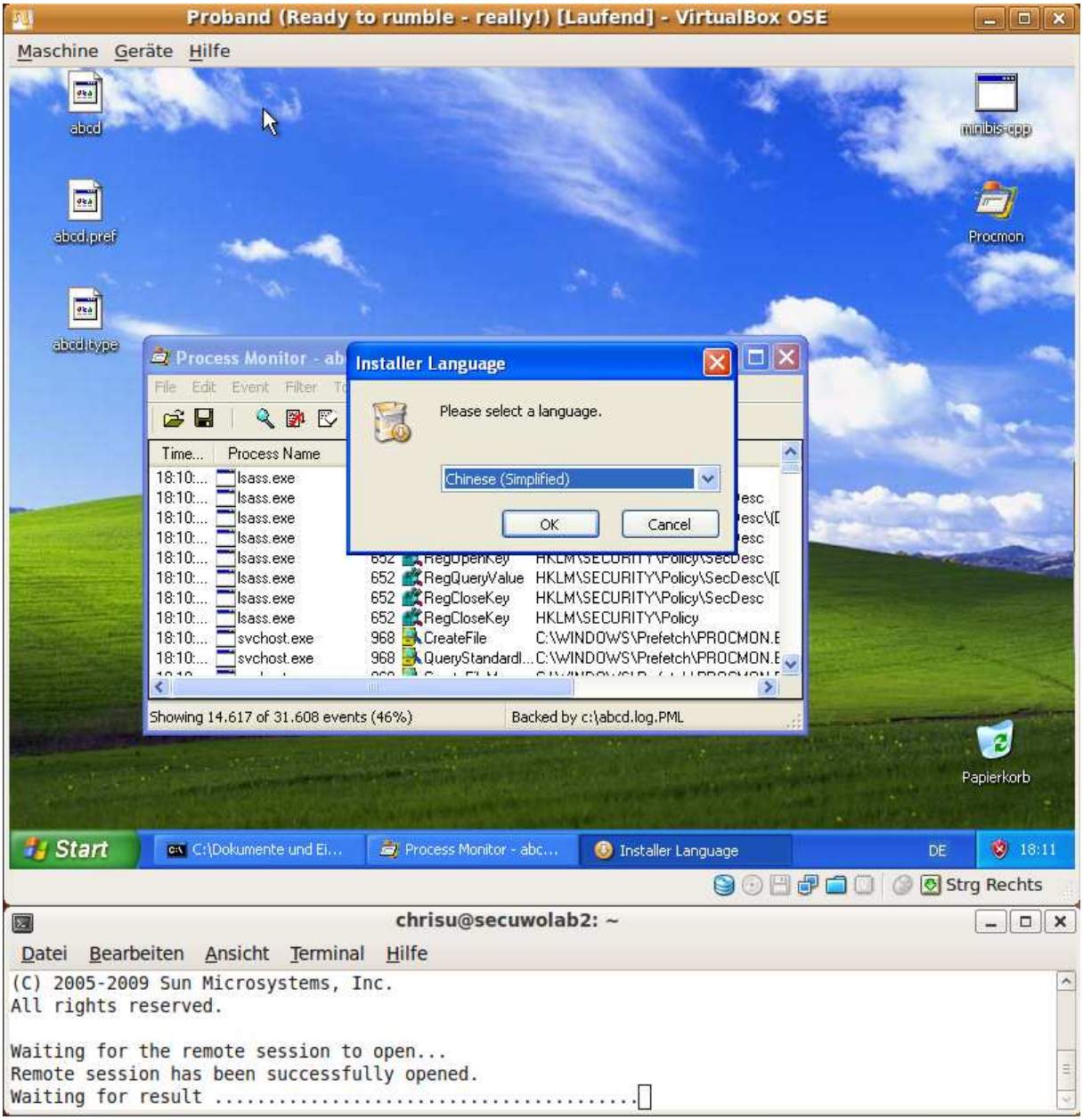
Screenshot 2: ProcessMonitor from Sysinternals has just been started.



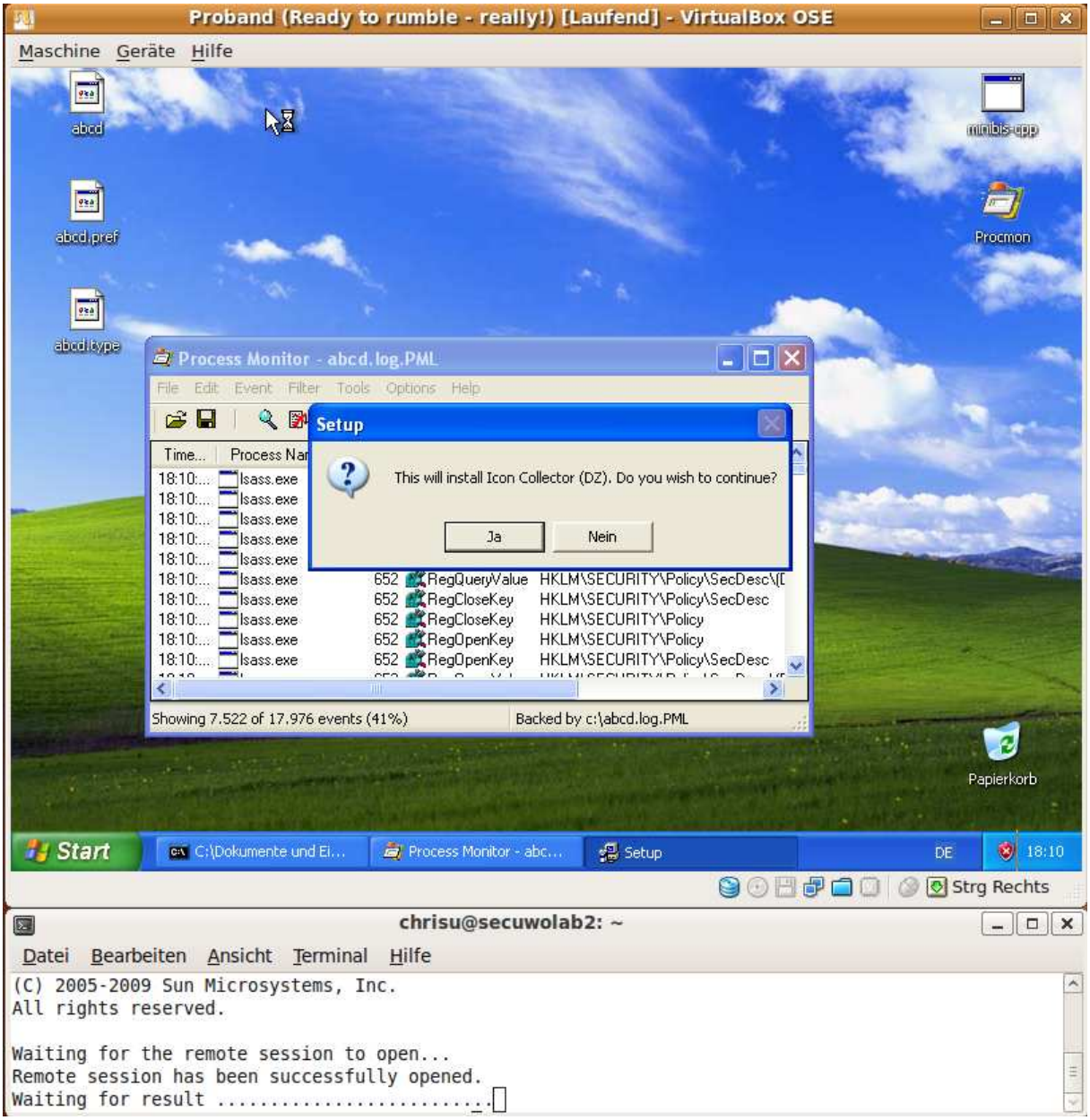
Screenshot 3: The native ProcessMonitor saving format is being converted to CSV.



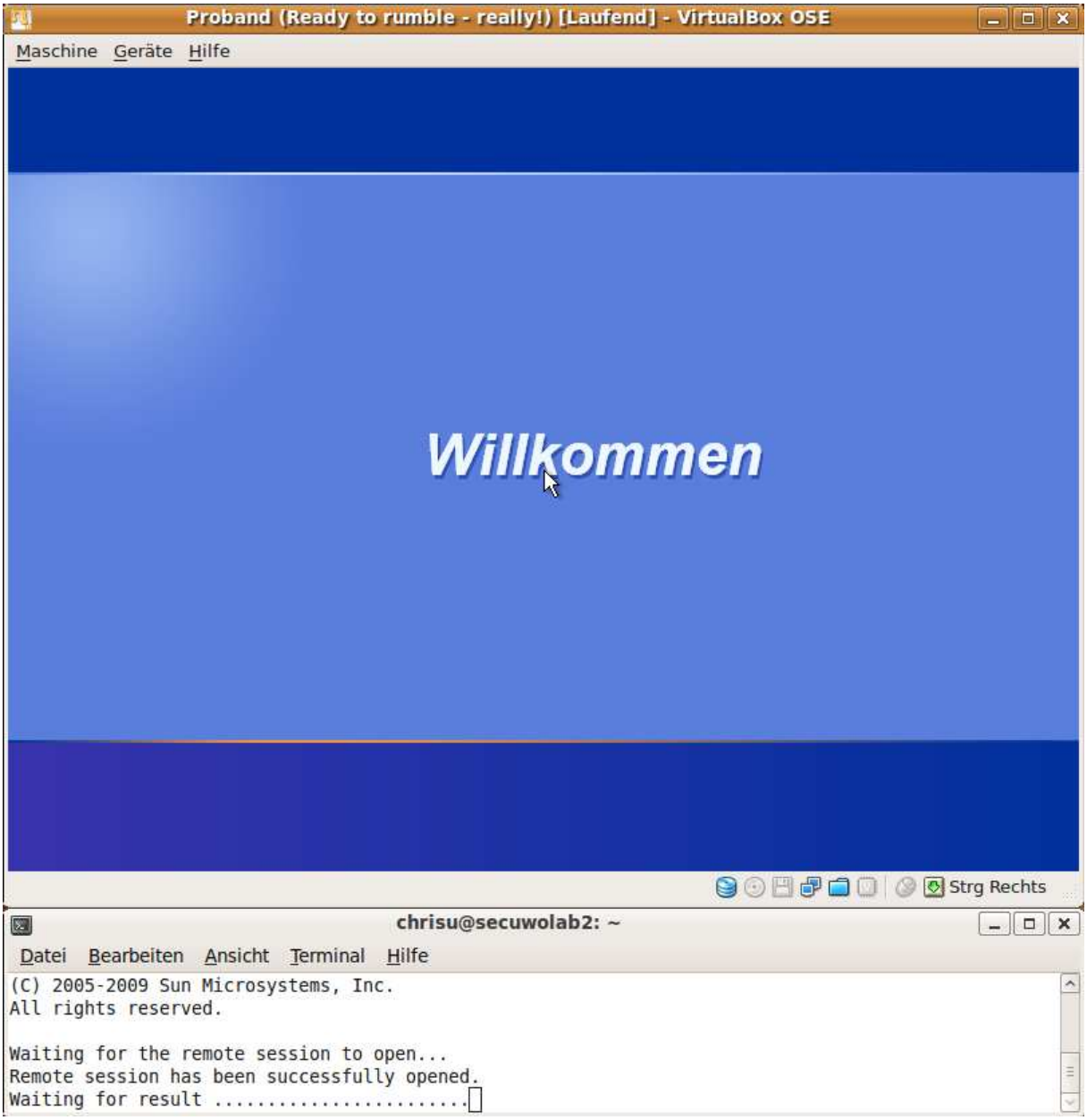
Screenshot 4: The virtual machine (proband) gets reverted.



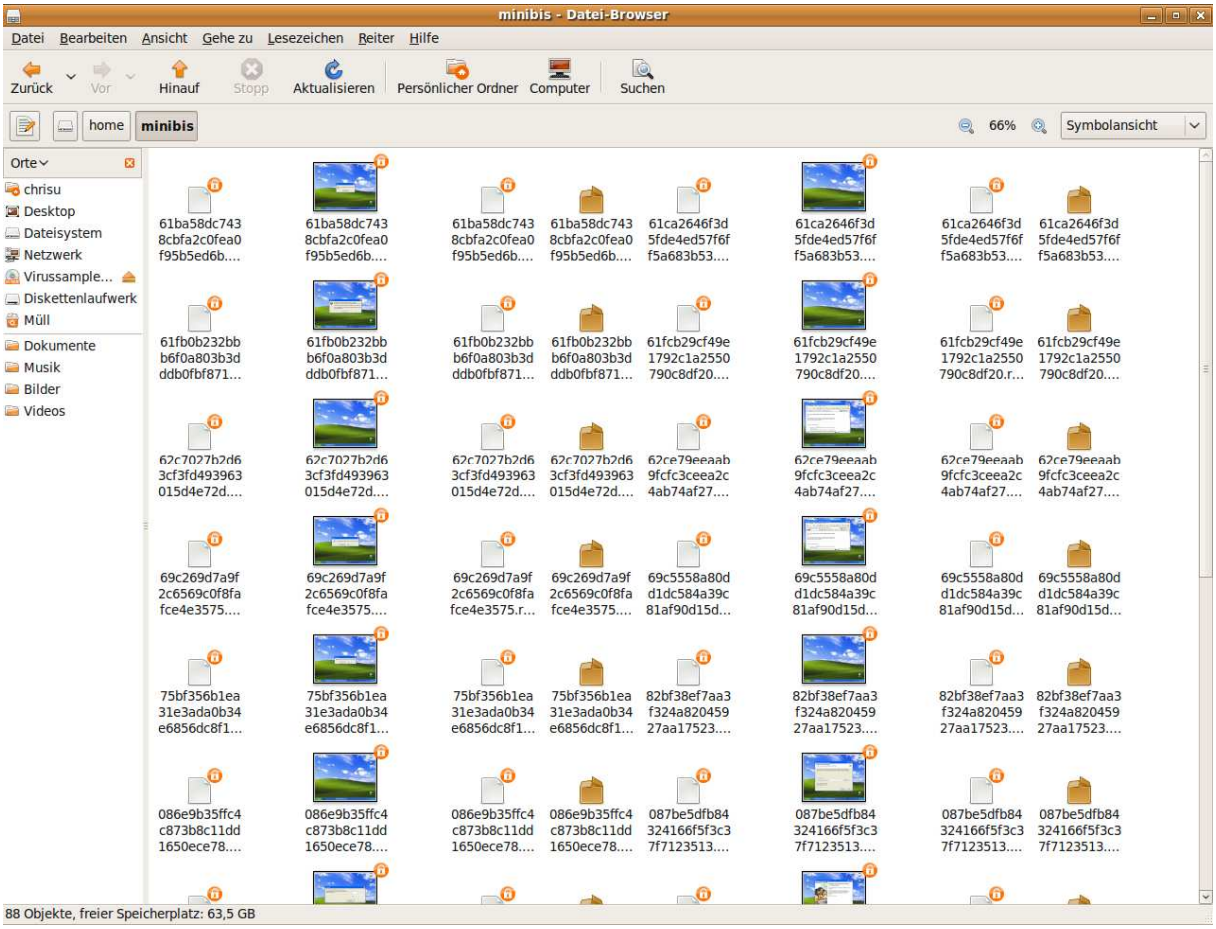
Screenshot 5: One example of active malware.



Screenshot 6: Another example of active malware.



Screenshot 7: Malware caused our proband to restart. That's why we need the outer emergency break.



Screenshot 8: Viewing results after a few samples.

9 Epilogue

So, if everything works fine, sooner or later you should have lots of report-files as well as screenshots (see Screenshot 8). You can now i.e. “grep” over all of the CSV-files to find out how many malwares try to detect VMware by reading certain Registry keys, find out malware that injects itself into other processes, check what’s the favourite auto-run variant of malware ... or even find out which activities are periodicly done by the OS.

As the abilities of our AMAS will definitely grow, this paper will evolve as well. So keep an eye on the latest version of this paper. My future plans are, bringing more flexibility into configuration of CERT.at’s control processes so that they might meet more of your special needs to relieve you of necessarily creating these two processes by yourself.

Now that you made it all through this paper I hope you enjoyed it and I wish you success building your AMAS.

10 Credits

There are some people I would like to thank for reading this paper for correction and verifying purposes:

Klaus Darilion
Tarmo Randel
Lenny Zeltser

Thank you!