# ECE 6747 Advanced Topics in Malware Analysis

# MODULE 8 TRANSCRIPTS

**L1 Automated Malware Analysis Framework**

>> Hello everyone and welcome back to Advanced Topics in Malware Analysis. In this week, I know you're busy working on your labs, so we're gonna take it a little easy and learn about automated malware analysis frameworks. We've already learned a lot about dynamic analysis and building up sandboxes that you can use to automatically extract information about a malware.

In this lesson, we're gonna look at how to compose all of those techniques into an automated sandbox framework that can extract as much information about a malware sample automatically as possible. A sandbox is a combination of hosts which could be virtual machines or physical computers that live just to run malware samples.

You usually have a front end host that's responsible for accepting new malware samples, classifying them, and then delegating, actually executing them to virtual machines or a server farm of physical servers. There's usually an isolated network so that the analysis of these malware cannot escape your sandbox environment. And each of the analysis guests is a clean environment that runs the sample along with some analysis tools that can easily be reset after the malware has executed.

There are a number of automated malware analysis frameworks out there, some of which you can download and play around with yourself. Note that these are never gonna be as deep of an inspection as real reverse engineering like you would do in IDA or in PIN. But it can provide a much quicker analysis that allows you to understand the different behaviors of the malware sample.

By far one of the most common frameworks for this type of analysis was one called Anubis. It has now been taken down. It's no longer freely available, because the authors of the Anubis framework have formed a startup company, Lastline, that is doing very successful malware analysis research. At this point the most popular open source sandbox is Cuckoo.

Many people use Cuckoo to do automated malware analysis, and Cuckoo can be extended to perform even advanced analysis like memory forensics. There are other types of automated frameworks for different types of malware such as document malware that come in PDFs or Microsoft Office files, and even Android applications.

Back when Anubis was available, this is what the home screen would look like. You could see the samples that were recently analyzed by Anubis, and down at the bottom, you could upload your own sample for Anubis to analyze and produce some results. This may look familiar because many of you are familiar with the virus total platform.

This is a common platform use now, which runs a malware sample in a similar sandbox and feeds that malware to a large array of different anti viruses. And then reports back to you on what those different anti viruses reported for the malware sample. Here at Georgia Tech, we have a hometown hero called Apiary.

This is maintained by the Georgia Tech Research Institute, which is a Research Institute affiliated with Georgia Tech. We built APIARY while ago to ingest malware samples and collect as much telemetry data on those samples as possible. To make quick analysis of new malware as painless as could be, before a reverse engineer had to dive in and hand reverse engineer different malware samples.

When you upload a sample to Apiary, it's going to automatically queue some analyses and add this malware to its large corpus of existing malware samples. You can then query different indicators of the malware to understand how that malware behaved and what execution artifacts could have been collected. These are things like API's that were called blacklists that the malware might have landed on, and any URLs on network behavior associated with that malware.

Apiary produces a simple report where it identifies the different suspicion factors, and how dangerous it considers this malware sample based on those suspicion factors. This is very common for most advanced malware sandbox techniques, such as Apiary. You can also see a rundown of different pieces of information about the malware file itself.

And then you can collect information that was found by Apiary while executing the malware sample, such as any DNS to IP resolutions that went on while the malware was executing. This can allow you to check out what websites or remote command and control servers the malware was trying to access.

Apiary keeps this stored so that as those domains or command and control servers get taken down, you can still have a record of which ones were used by the malware. Similarly, Apiary provides network observations, such as what systems were being accessed by the malware when it executed. You can also download a detailed packet capture from the malware that was generated when it was executing.

There's also some simple static analysis to understand the features of the malware binary file itself. And you can query that information from Apiary as well. And similarly, the executable files sections and import tables can be checked to determine perhaps if there's any packing involved or what API's this malware is likely to call.

These are all features that you would find in any robust malware sandbox framework. So I recommend that you go play around with Cuckoo or any other popular malware sandbox that you're interested in.

**L2 Real World Malware Investigation: Employee “Misconduct”**

>> Hello everyone, and welcome back to Advanced Topics in Malware Analysis. In this lesson, I'm gonna talk about how we actually used a malware sandbox framework to identify a malware sample that was hiding from pretty much every investigator that had looked at it. This investigation actually saved an employee their job.

And I'll get into exactly what happened in this lesson. Researchers got called in to investigate this malware sample by the lawyers defending an employee who had been wrongfully terminated from a company. The company alleged that the employee was misusing their computers during the day when they were supposed to be working.

It seemed like an open and shut internet browser forensics case. Look at what websites the employee had been visiting, make sure that they were on the level. And if they weren't, the employee probably should have been fired. But that's not what we found. We found quite a bit of web activity.

Internet Explorer had been used to browse and download thousands and thousands of not safe for work images. It appeared that the user had spent their entire day doing no work except for downloading it. We were actually taken off guard by the sheer amount of web activity for this purpose.

Even stranger, the times that all of these images were downloaded, corresponded directly with the times that the user was logged in and working at their computer. We were taken back of course, the employee denied everything. They said, Surely a virus must have done it. I'm not the type of person who would do this when I'm at work.

This is called the Trojan defense, often used by defendants to try to get out of cyber crimes by claiming that a virus must have done it. This is generally not taken seriously by many cyber investigators. And indeed, most button pushing cyber investigators might have come in, just run an antivirus found nothing.

Assumed that this employee was lying and not had anything to present in court to help this employee out. However, this employee got lucky. Because they didn't call in an unsophisticated investigator, the malware analysis that went on for this case was actually quite robust. It turned out that there was indeed a malware infecting this employee's computer.

It hooked into a DLL used by Internet Explorer, that controls the web browser in the same way that the user interface of the web browser can control, the browsing. Internet Explorer actually calls this same DLL to navigate to different pages without the user needing to click on anything.

So it's generally considered a helpful feature of Internet Explorer. The malware, however, had hooked into this feature, and was using it maliciously to navigate the internet browser silently in the background of the computer. Even stranger the malware seemed to inhibit process monitors such as procmon, because when we installed procmon on the user's computer, it reported nothing suspicious.

The malware binary itself turned out to be quite a nightmare to analyse It was packed with a standard packer that could be identified and unpacked with limited trouble. It was packed with a standard packer called Asprotect, that could be unpacked with just a little bit of trouble. The binary itself turned out to be written in Borland Delphi.

This is a flavor of compilers that really mangles a binary and make static analysis very difficult. We tried to unpack some of the fields in IDA. But it turns out the binary itself was way too complex to look at in raw assembly. And when we try to decompile the binary with existing decompiler tools, they simply crashed whenever they tried to read the binary sample.

It was at this point that we tried the Anubis sandbox. Running the sample in Anubis produced logs that showed the malware sample hooking into the Internet Explorer library and beginning to navigate and download not safe for work content. Whenever the computer was actively being used. When the computer went idle, the malware would pause.

So we have to credit Anubis in this case for getting that employee rehired.

**L3 Anti-VM Techniques**

>> Hello, everyone, and welcome back to Advanced Topics in Malware Analysis. In this lesson, we're gonna talk about tricks malware can play to defeat virtual machines and detect or break out when they're running in one. Anti-VM is the Achilles heel of sandboxes. Because most sandboxes are built from virtual machines, modern malware will sometimes contain measures to detect that they are running in a virtual machine, such as VMware Workstation or Fusion.

And malware may refuse to run or alter their behavior when a VM is detected. Most legitimate applications will not do this. However, some, including testing software or other software that does not want to allow you to reverse engineer it, may refuse to run in a VM. The reasons we use VMs for malware analysis are obvious.

So it's sort of up to reverse engineers to force malware to run in a VM when they need it to. Some common tricks malware will use to detect if they're running in a VM is to look for virtualized devices inside of that VM. VMs provide these virtual hardware devices.

And the operating system often picks up the fact that these are not real physical devices, but instead provided by the Virtual Machine Manager running underneath the virtual machine. For example, in VMware, you will see a VMware mouse, VMware network card, and VMware hard drive hardware when you're running inside of a VMware VM.

This can give away the fact that the malware is running in a VMware VM. Other artifacts, VMware is not a single example, as all virtual machines give away artifacts that you're running inside of a virtual machine. VMware, for example, will place entries into the Windows registry, noting that it's a VMware virtual machine.

Especially if you have some enhancement functionality installed, like VMware Tools, inside of the guest operating system. Malware have even been known to scan a system's memory looking for the string VMware. If you're not running in a VMware virtual machine, it's very unlikely your memory will contain VMware in it.

Other tricks, like looking for the interrupt descriptor table, might also give away that you're not running on bare metal. And there's many more of these. For example, a malware named the HARULF malware has two different anti-VM checks. One that looks at the location of the local descriptor table, which if you're running on Windows will be one value.

But if you're running Windows in a VM will be a different value. The malware then calls the CPUID instruction and checks this values that the CPU sets in the registers. There are bits in that return that could give away the fact that you're running in a virtual machine.

Another excellent piece of research identified the fact that you could use a single sequence of instructions, so small that it fits right here on this slide, to detect if you're running in a virtual machine. By checking the address of the interrupt descriptor table against what you would expect from running on bare metal.

If you're curious, read the whole paper published by Invisible Things Lab on how this attack works. There are similar Red Pill attacks for the global descriptor table and local descriptor table, which you can access using the SGDT and SLDT instructions. These detection techniques are becoming so common, there's actually a library called the ScoopyNG collection that you can download and build a number of different tests if you're running in a virtual machine.

One of the most effective simply checks if there is a host to guest communication channel by basically yelling data down at a Virtual Machine Manager and checking if anything answers. If something answers, you're obviously running on a virtual machine, if nothing answers, you're most likely running on bare metal.

The code looks something like this. There's a special IO instruction called in that you can write data down to a communication channel with hardware. Virtual Machine Managers will set the registers to specific return values if you send them specific commands. This command allows you to check based on the return value what version of VMware the VM is running on.

Otherwise, if you don't get an answer, you must be running on bare metal. So how can you defeat these anti-VM techniques if you come across them in malware? Well, one approach is to simply watch the malware closely as it executes, look for any values that it's reading or writing from the system.

And if you can identify that they are VM detecting values, try to hide those values from the malware. Similarly, if there are side effects of the VM itself, such as shared folders or shared clipboard, etc., those can be often turned off, or you can try to hide them from the malware using dynamic analysis.

An excellent read is a research paper published a while ago on hiding different virtualization techniques from attackers. If you have to go to extremes, we've heard of research that tries to patch the VMware binary itself to change the magic number used for communication, breaking that communication channel detection attack we talked about earlier.

There've also been attempts to create custom Virtual Machine Managers that malware will not detect. But as soon as it gets popular, you can bet malware is gonna start detecting it. A project cited in the paper above also mentioned brute force attempts to change magic numbers everywhere that they occur in the Virtual Machine Manager binary.

This is an interesting approach, but its practicality might be limited. And that's the end of the lesson. I know you're all working on your labs. And I hope that this has been informative, how you can utilize virtual machines to automate the process of dynamic analysis of malware. And we talked about how this can even be extremely helpful in real world malware investigations.