# Pinky*: A Modern Malware-oriented Dynamic Information Retrieval Tool

Paul Irofti

July 12, 2013

## Abstract

A multifacet tool that aids the entire process of antimalware development starting from laboratory work and bulk-analysis, to reverse-engineering investigations of hot samples, to creating mix-and-match dynamic and context aware signatures, all the way up to the in-field deployment process of pro-active behaviour-based detection and universal unpacking safety-net.

## 1 Introduction

When development began on this ambitious project, the antimalware scene was fully developed and had a lot of research put into the problems that arise with dynamic tools and information retrieval techniques. In fact, the research started moving so fast that traditional security companies were a bit behind on the new discoveries and algorithms concerning heuristics, instrumentation and compiler design. This sprinkled a keen interest in developing a new tool from the ground-up with the building-blocks set in post-2010 research advancements.

Being a small team of experienced reverse engineers, very well accustomed to the legacy yet fascinating 1990 technology for emulating samples used in popular antimalware products, we decided that it's time to bring on a new product to the table.

A product that won't resume itself to being just an emulator.

A product that will take technology further and match the needs of modern reverse-engineers and white-hats.

A product that will be able to adjust as it goes to new platforms, new research, and to new analysis techniques by being not just an input - output blackbox as its predecessors, but a blackbox with a feedback loop that allows the system to be observed, controlled and adjusted to properly meet the needs of its masters.

## 2 High Level Solution

Pinky is designed as an opaque tool with a clear and simple interface that allows it to be integrated and controlled by third-party applications in a non-intrusive way.

It is platform independent. The samples

*this is just a project codename and can be changed once a product name is chosen

it analyzes are agnostic of the platform it is ran on. For example you can analyze a Windows 32-bit executable on a Linux distribution running on a MIPS-64 platform. Thus it can be used both on the server market, serving for example as a mail-scanner, and on the client market aiding an existent antivirus solution.

The emulator is very fast due to it's modern virtual machine implementation that coupled with the just-in-time compilation strategy and a performant caching system gives one of the fastest malware-targeted dynamic analysis tools in the industry.

Pinky is also used as a tracer. It can give traces of system calls and native APIs (such as ntdll.dll, kernel32.dll, advapi32.dll and so on) with the extending possibility of also capturing TCP/IP traffic.

Pinky has a generic information retrieval framework that is designed to serve every need it's surrounding environment might have in coming up with a proper verdict on a given sample.

It is designed with ubiquitous dynamic instrumentation in mind. At any point in the emulation process, Pinky can provide feedback on the state it's in. Examples like mapped memory, registers, stack, text section, filesystem state come to mind.

As an added bonus, the instrumentation framework has no performance impact on the emulation process.

The emulator has a callback system setup that permits context-aware signatures and, through the instrumentation framework, behaviour-based detection that can aid in pro-active comparative tests ratings.

Also aiding both basic and pro-active detection is the ability to act as a universal unpacker that helps to cope with new or custom packers or even new versions of the existing ones.

Pinky is also a laboratory tool. It can be used as an analysis and information retrieval tool by reverse engineers just like OllyDBG or IDAPro with the benefit that this was built and designed with malware investigation in mind so it has more field specific functionality than the other two.

As a lab tool it can also be used in bulk scans to craft generic or specific reports. The generated data can include sample geometry, memory dumps, classification criteria, profiling data and so on and so forth.

Another unique feature is the ability to stop the emulation process in a coherent and platform agnostic fashion. This implies reproducibility no matter of the CPU frequency, memory size or type, disk IO throughput and other machine dependent factors.

# 3 Solution Details

In order to make integration easier, Pinky is provided as a library. In both static and dynamic forms.

So far Pinky has been integrated and used successfully in an antimalware engine environment (acting as a generic unpacker and memory inspection tool doubling the product's detection rate), as a bulk scanning tool for malware and clean sets, and also as a debugger-like reverse-engineering tool for sample analysis. Three applications that seamlessly integrated the library with success.

Pinky-based solutions are being used with success on multiple operating systems and hardware platforms. For exam-

ple the bulk scanning tool runs on Linux, OpenBSD and Windows with 32-bit and 64-bit Intel-derivate CPUs. Also quick nightly scans are conducted on a plethora of system configurations, both big endian and little endian, with hardware platforms such as Intel 32-bit and 64-bit, ARMv5 and ARMv7, MIPS-64, PowerPC, Sparc, Sparc64, HP-PA, and on operating systems such as Windows versions from Windows XP up to Windows 8, OS X, Linux, FreeBSD, OpenBSD, NetBSD, Solaris, IllumOS, Darwin and others.

Pinky is written in C++ with a focus on the C-subset of the C++ Language. Thus, besides being portable, the C++ interface makes it very easy to integrate, natively or through wrappers, in virtually any project no matter of the language chosen.

The interface is simple and intuitive. It consists of three parts: the emulator interface, the configuration interface and the instrumentation interface. It is implemented through abstract virtual classes that make it easy to decouple from the rest of the project.

The emulation speed is given by tiered compilation through threshold-based mixins of JIT compilation and VM emulation. Even more, every codeblock that gets through is cached and will be reused the next time it's encountered. Currently there are two caching strategies to choose from. Further optimizations may occur when a codeblock is seen often enough. If a platform is missing JIT support, it will always fallback on the VM implementation which is very fast due to the custom orthogonal ISA and its optimized opcodes.

Instrumentation is done by dynamically enabling and disabling information retrieval points throughout the emulation process. Data points can even respect a certain caller-callee protocol and exchange data structures that can affect the sample's control and/or data flow.

The emulator supports as many instrumentation points as needed due to the zero performance impact when the points are disabled.

One can create complex scenarios that play with enabling and disabling such points of interest in order to profile or analyze a certain path the sample might take. Or better, creating behavioural detection patterns that boost pro-active rates and heuristic detections.

The callback system is designed with the antimalware engines eco-system in mind. For example a common issue that comes up in the field is handling polymorphic routines in static unpackers and coping with the different versions and variations in the wild. This can get to a point where the static routine gets so complex and has to deal with so many cases that it slowly turns into a crippled dynamic analysis tool. To solve such cases, Pinky can be used to patch things up. Let the static unpacking process run until the offending polymorphic routine is reached. Stop and handle things over to the emulator which will dynamically unpack it and then give control back to the static routine.

When used as a reverse-engineering tool, it can act as a debugger by setting breakpoints, watches, single-stepping at different granularity (e.g codeblocks or instructions), setting different instrumentation points at runtime, tracing system calls and library APIs, enabling different logs at various logging levels for discrete periods of time, and

many other similar useful features.

# 4 Business Benefits

This product is designed to boost detection rate through integration with the rest of the engine eco-system. Not only that, but due to its dynamic nature it will also increase significantly the pro-active results visible in antivirus comparative tests and through the user-base.

Thorough tests can be ran in the laboratory and, through information retrieval, new heuristics can be added and the old ones can be adjusted bringing false positives down and keeping once-again pro-active detection up.

An engineer can quickly setup through instrumentation a look-up routine that once scripted into a bulk scan can lead him to a conclusion very fast be it in regards to a family of malware, a general rule-of-thumb for a specific set of samples, or other statistics that might be of interest. This has shown in our laboratory that is much more effective and a lot less time consuming than scripting a similar routine for IDAPro or OllyDBG.

The speed-up compared to using other virtual-machines is considerable and it's highly visible when running bulk scans on millions of samples. The high-speed and low-memory footprint is also very welcomed and appreciated by the end-users.

Another great feature is the ability to provide it to reverse-engineers as a very accurate customizable sample dissecting tool bringing faster detection and unpacking routines along with more accurate disinfection scripts.

# 5 Conclusions

This is a turnkey product.

It can be quickly integrated into any existing solution and with minimal effort at that.

Being of modern design it has an inner-module decoupled design and so it's easily adaptable to specific needs, be it specific hardware dedicated algorithms, memory or on disk size constraints, accelerated throughput for special performance oriented chips, and so on.

The instrumentation framework allows for us to fulfill on demand crafting of information retrieval policies or data acquisition strategies that the upper layers might need from the product.

All-in-all it's a very dynamic production-ready product with multiple modes of operation that is used with great success in the field by both companies and end-users.