

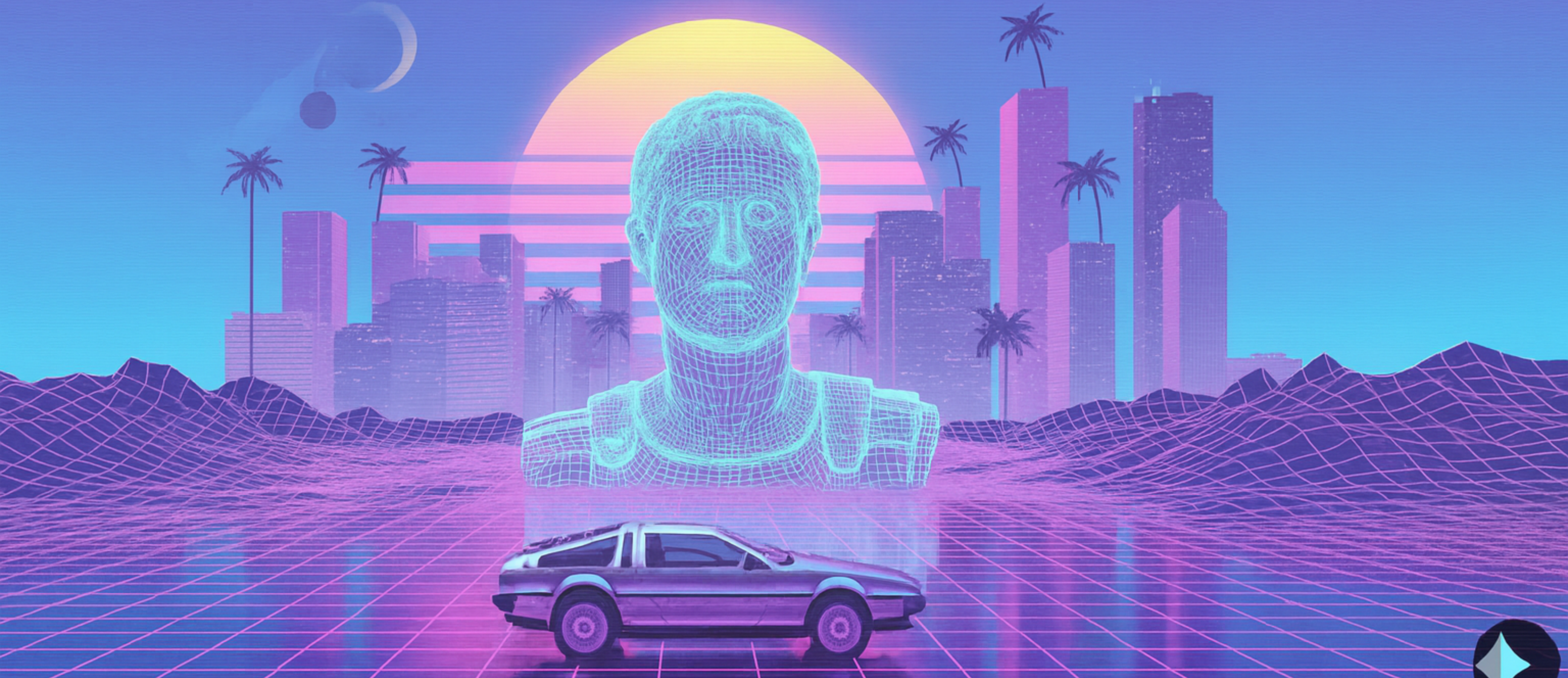


Reverse Engineering in the *2020s*

Where are we, and where should we go?

Brendan Dolan-Gavitt

REVERSE ENGINEERING IN THE 2020s



Why Reverse Engineering?

I. Motivation

Reverse Engineering (RE) is the process of discovering undocumented internal principles of a piece of code. **Why would anyone who is not a criminal want to do that?** We

Repeatable Reverse Engineering with PANDA, 2014

- Long ago (the 2000s), the field of reverse engineering had a slightly disreputable reputation
 - Are you trying to steal someone else's intellectual property??
- But there are lots of good reasons to reverse engineer!

Malware Analysis

Interoperability

Software Archaeology

**Uncovering
Hidden
Behavior**

Vulnerability Discovery

Debugging

How I Got Here

- Started working on reverse engineering to support Windows memory forensics
- Expanded into memory analysis for virtualization security (translation: same thing, but from a hypervisor)
- Built systems for supporting reverse engineering through whole-system dynamic analyses (PANDA, Malrec)
- Worked on automating reverse engineering for *rehosting* – emulating embedded devices
- Side projects on neural decompilation, fuzzing



Reverse Engineering for Memory Forensics

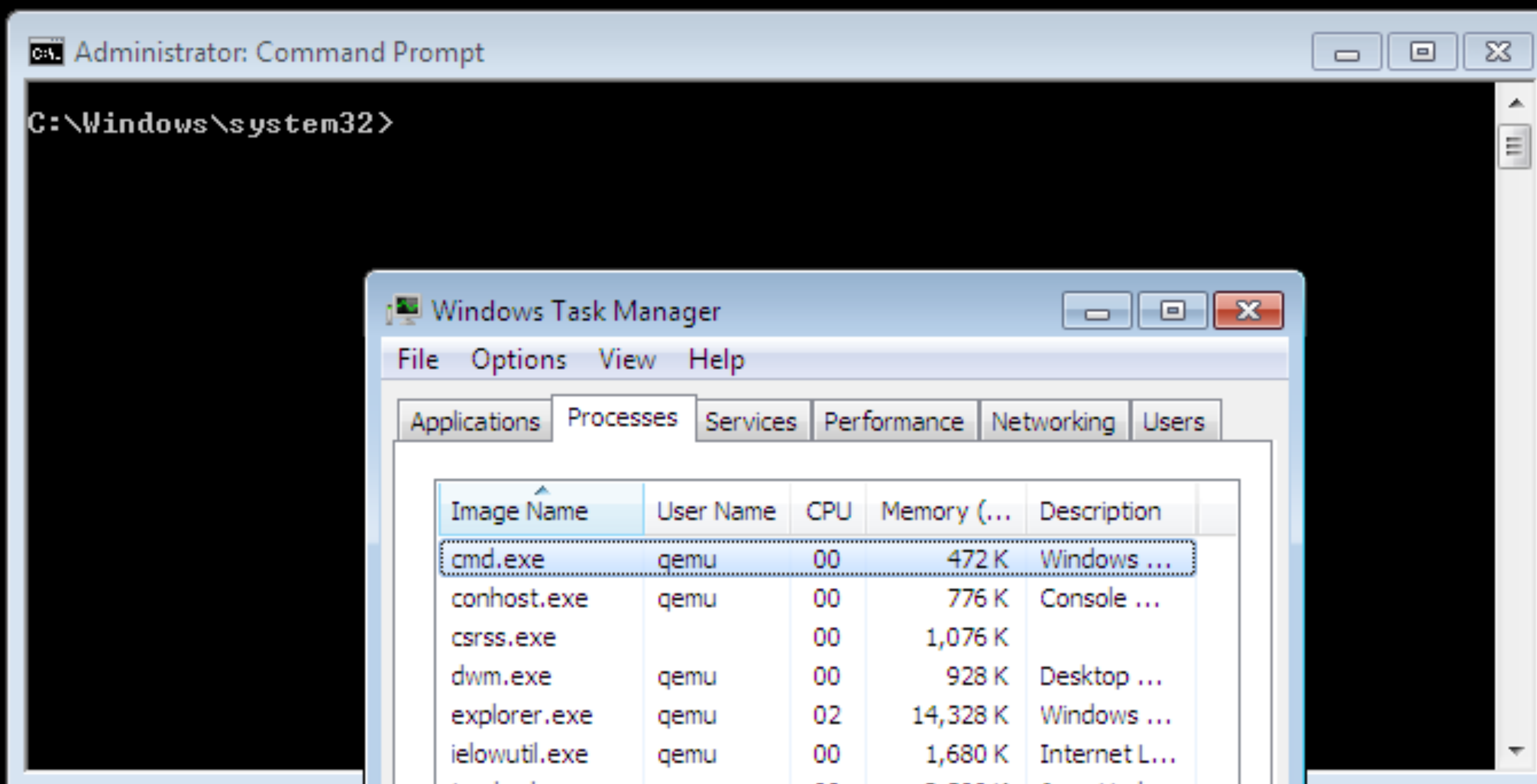
- Basic problem: how do you extract the *live state* of a system from a memory dump?
- Answer: by reverse engineering the Windows NT kernel, of course!
- You can get lots of cool stuff from this:
 - Lists of running processes, loaded DLLs
 - Information about active network connections
 - Registry data as it appeared in memory
 - ...



Recycle Bin



Adobe Reader
XI



Windows Task Manager

File Options View Help

Applications Processes Services Performance Networking Users

Image Name	User Name	CPU	Memory (...)	Description
cmd.exe	qemu	00	472 K	Windows ...
conhost.exe	qemu	00	776 K	Console ...
csrss.exe		00	1,076 K	
dwm.exe	qemu	00	928 K	Desktop ...
explorer.exe	qemu	02	14,328 K	Windows ...
ielowutil.exe	qemu	00	1,680 K	Internet L...
jusched.exe	qemu	00	2,300 K	Java Upd...
taskhost.exe	qemu	00	1,428 K	Host Proc...
taskmgr.exe	qemu	09	1,404 K	Windows ...
winlogon.exe		00	1,256 K	

Show processes from all users End Process

Processes: 33 CPU Usage: 17% Physical Memory: 30%

Windows 7
Build 7601

This copy of Windows is not genuine



2:52 AM
5/29/2015

Administrator: Command Prompt

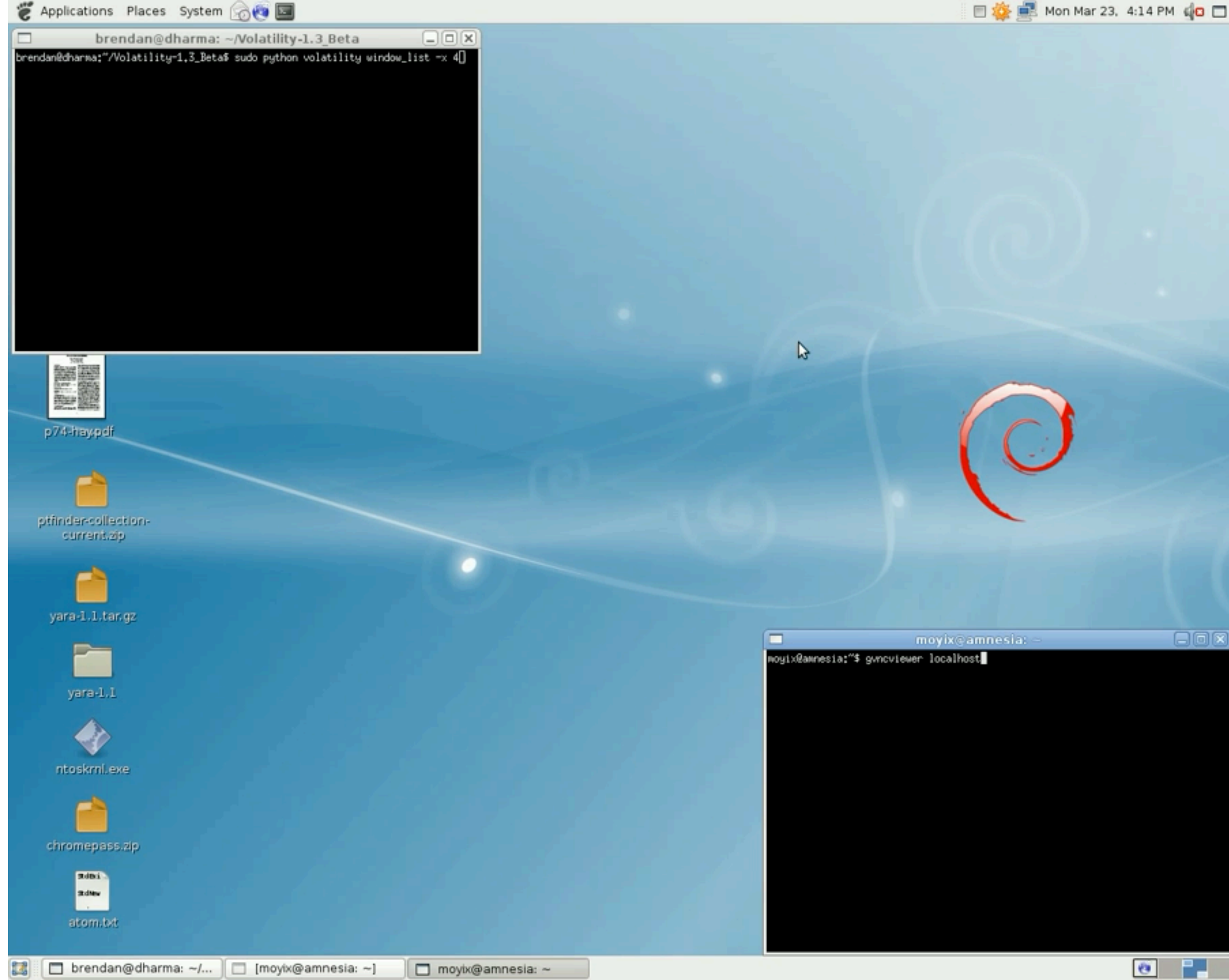
Windows Task Manager

Tab1

Processes

&Show processes from all users

&End Process



[https://
www.youtube.co
m/watch?
v=c6OMISoDXrw](https://www.youtube.com/watch?v=c6OMISoDXrw)

Whole-System Dynamic Analysis

Breaking Spotify DRM

- DRM has a strong signature
 - **High** entropy, **high** randomness (χ^2) input
 - **High** entropy, **low** randomness (χ^2) output
- We can look for functions that match this description
- Then just hook their output, and save it!

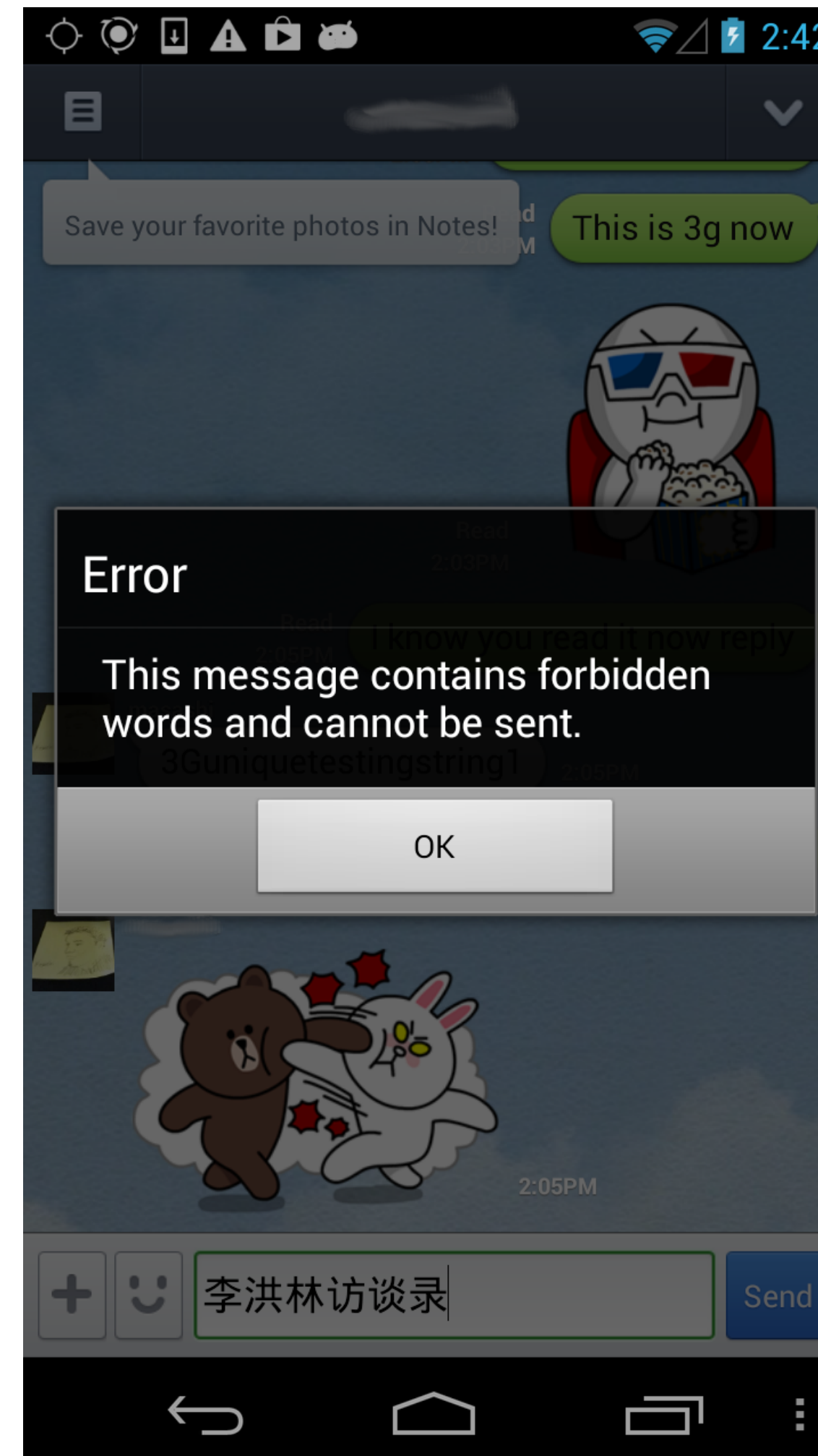


Whole-System Dynamic Analysis

Censorship Blacklist Extraction



- LINE is a Japanese-made IM app for Android with ~560M users worldwide
- Found by CitizenLab to censor some words for Chinese users
- We wanted to find out which ones
- Very simple strategy: search every memory read & write to find usage of strings likely to be in “bad words” list (法轮 (Falun), 天安门 (Tiananmen))
- Dump everything else passing through that point



Censorship Blacklist (sample)

198964	政变	共党	彭博
FLG	周斌	共匪	天朝
GCD	祖莹	共贼	天朝
GFW	共C档	胡温	屠城
18大	08宪章	江派	屠杀
38军	89事件	江系	团派
八九	艾未未	江贼	退党
半羽	薄瓜瓜	近平	汪洋
鲍彤	薄熙来	九评	瘟神
暴政	曹建明	军警	晓波
柴玲	曾庆红	六四	学潮
赤匪	陈光诚	马凯	学运
	大纪元	民运	余杰

For translations & context see <https://china-chats.net/>

Themes in RE Research Over Time

- Over the past 20 years, academic research in reverse engineering has undergone some substantial shifts
 - x86 software running on desktops -> many architectures, desktop/embedded/mobile
 - Closed tools, hard to reproduce -> open artifacts, good open source tools
 - Reverse engineering to support X -> papers studying RE itself regularly in top conferences

Then: Closed Tools, Poor Reproducibility

- It used to be very common to release papers with no accompanying code
- This made it extremely difficult to build on prior research
 - Each group needed to reimplement the history of the field before they could make any new progress!
 - Particularly for areas that require significant amounts of system-building, this cost quickly exceeds what anyone is willing to do for a single paper.
- I speculate this is one reason why there was so little progress in decompilation and binary type inference for almost 20 years

Example: Retypd

(Details from Bosamiya et al.'s TRex paper at USENIX Sec 2025)

- Originally presented at PLDI in 2016 by Noonan et al. (GrammaTech)
- In 2021, released open source 😊
 - ...well, kind of. Actually, reimplemented by a different GrammaTech employee after the original author had left.
 - They found it very difficult!

¹Indeed for one paper [25], other employees at the same company attempting to reproduce the work stated that “[i]t is a powerful system but difficult to understand” and the “presentation is very dense and subtle” [12].

<https://github.com/GrammaTech/retypd/blob/f8dd231478c3e1722d0d160c3cf99c628a25/reference/type-recovery.rst>

Now: Open Tools and Artifact Evaluation!



TRex: Practical Type Reconstruction for Binary Code

Jay Bosamiya^{1*}, Maverick Woo², and Bryan Parno²

¹*Microsoft Research*

²*Carnegie Mellon University*

Then: Reverse Engineering for *X*

- Reverse engineering was most often a *supporting* discipline
- It was fairly rare to see top conference papers dedicated to RE itself!
- Instead, you would find papers that came up with some cool new technique for binary analysis, and then applied it the topic du jour:
 - Botnets (e.g., automated RE of network protocols)
 - Vulnerability discovery (e.g., CGC & binary fuzzing work)
 - Debloating

This paper is essentially an automated reverse engineering paper disguised as a debloating paper.... and I like it.

— Anonymous Reviewer of our IEEE S&P 2022 paper, IRQDebloat

Now: Reverse Engineering on Its Own

- Increased interest in:
 - Decompilation: <https://mahaloz.re/dec-progress-2024>
 - Type reconstruction
 - Reliable disassembly (and reassembly)
 - Fuzzing of binary software
 - Automated reverse engineering of embedded devices (rehosting)

Fulfilling the Potential of RE?

- Reverse engineering has the potential to *help return control of software back to its users*
- This is important since software increasingly mediates so much of what we do every day: how we talk to other people, the devices in our homes, even the code that runs on cars and trains
- We have seen many examples of how reverse engineering can be used as a force for good, uncovering bad behavior in software systems

Notable Examples of “RE for Good”

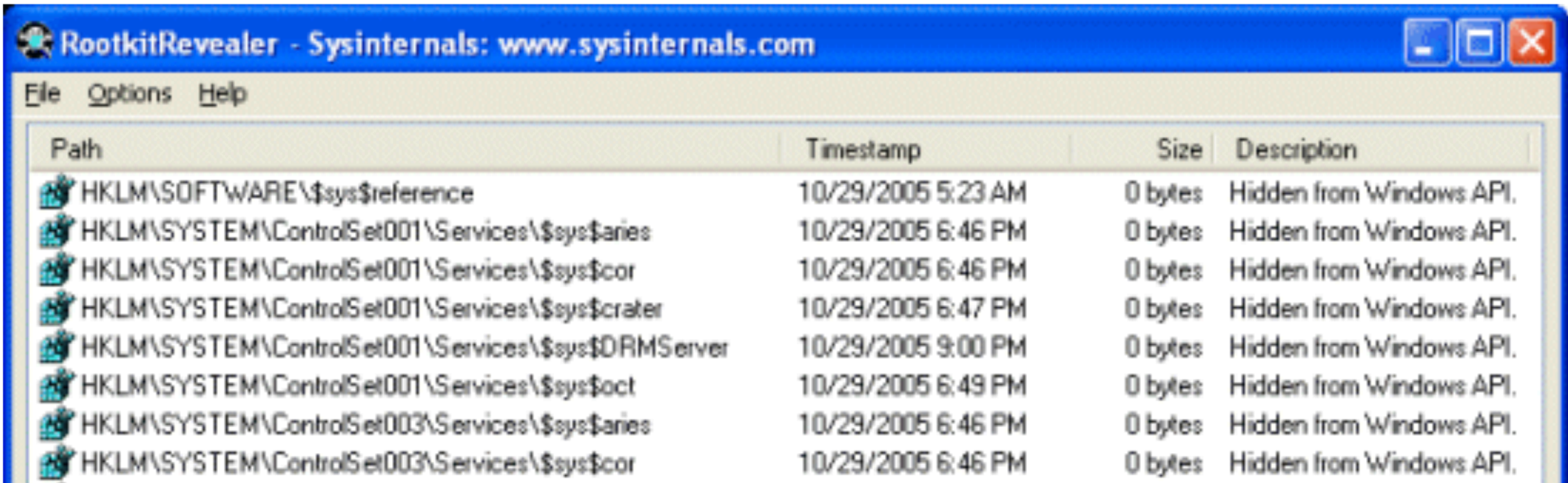
Sony Rootkit (2005)

Mark's Sysinternals Blog

Monday, October 31, 2005

Sony, Rootkits And Digital Rights Management Gone Too Far

Last week when I was testing the latest version of [RootkitRevealer](#) (RKR) I ran a scan on one of my systems and was shocked to see evidence of a rootkit. Rootkits are cloaking technologies that hide files, Registry keys, and other system objects from diagnostic and security software, and they are usually employed by malware attempting to keep their implementation hidden (see my [“Unearthing Rootkits”](#) article from the June issue of Windows IT Pro Magazine for more information on rootkits). The RKR results window reported a hidden directory, several hidden device drivers, and a hidden application:



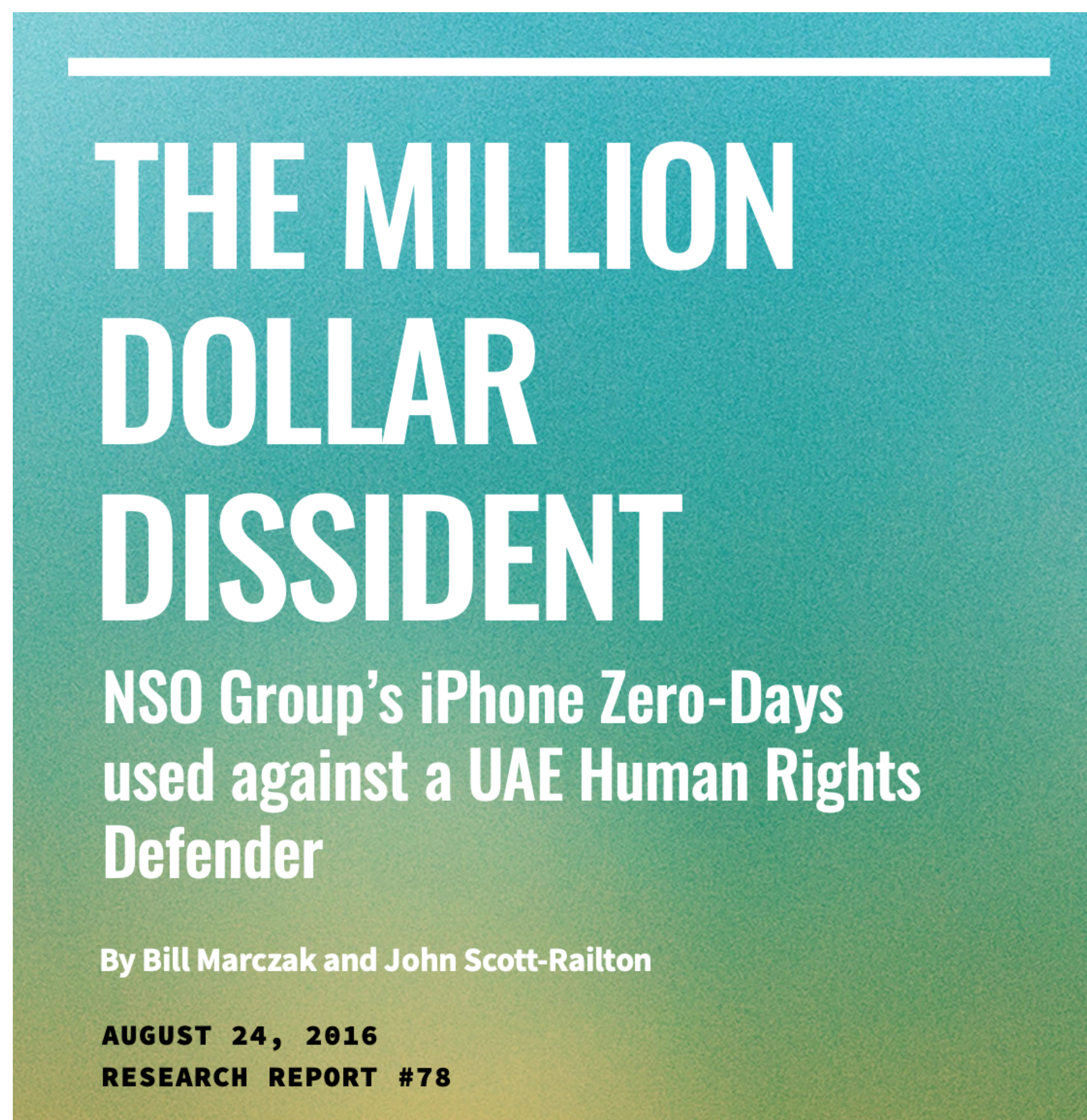
The screenshot shows the RootkitRevealer application window with the title bar 'RootkitRevealer - Sysinternals: www.sysinternals.com'. The menu bar includes 'File', 'Options', and 'Help'. The main content area displays a table of hidden registry keys.

Path	Timestamp	Size	Description
HKLM\SOFTWARE\sys\$reference	10/29/2005 5:23 AM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Services\sys\$aries	10/29/2005 6:46 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Services\sys\$cor	10/29/2005 6:46 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Services\sys\$crater	10/29/2005 6:47 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Services\sys\$DRMServer	10/29/2005 9:00 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Services\sys\$oct	10/29/2005 6:49 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet003\Services\sys\$aries	10/29/2005 6:46 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet003\Services\sys\$cor	10/29/2005 6:46 PM	0 bytes	Hidden from Windows API.

<https://web.archive.org/web/20051102053346/http://www.sysinternals.com/blog/2005/10/sony-rootkits-and-digital-rights.html>

Notable Examples of “RE for Good”

CitizenLab Reverse Engineering of Malware Targeting Activists



<https://citizenlab.ca/2016/08/million-dollar-dissident-iphone-zero-day-nso-group-uae/>



Figure 3: SMS text messages received by Mansoor (English: "New secrets about torture of Emiratis in state prisons"). The sender's phone numbers are spoofed.

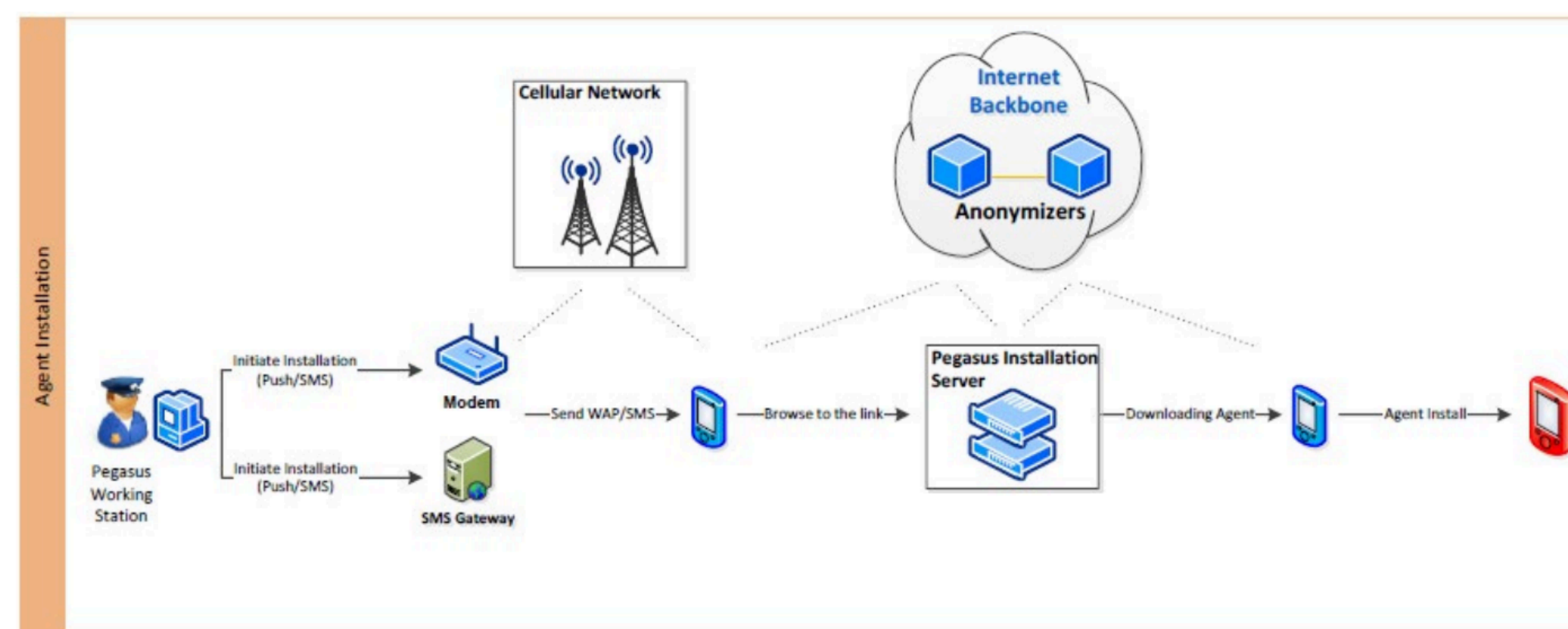
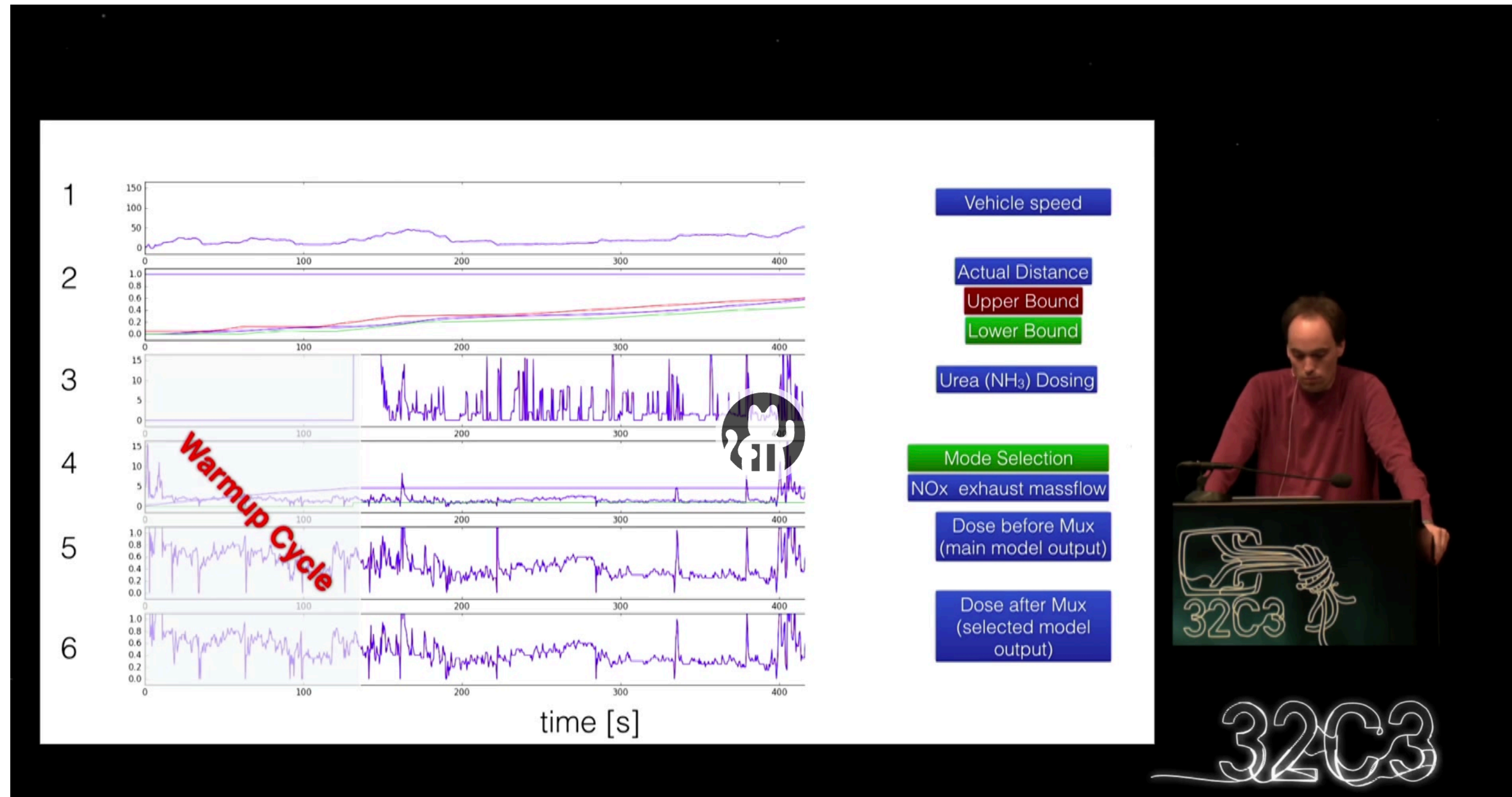


Figure 6: Diagram from purported NSO Group Pegasus documentation showing the sequence through which the spyware (“Agent”) is installed on a target’s mobile device. Source: Hacking Team Emails.

Notable Examples of “RE for Good”

“DieselGate” and other “Defeat Devices”



https://media.ccc.de/v/32c3-7331-the_exhaust_emissions_scandal_dieseldgate

Notable Examples of “RE for Good”

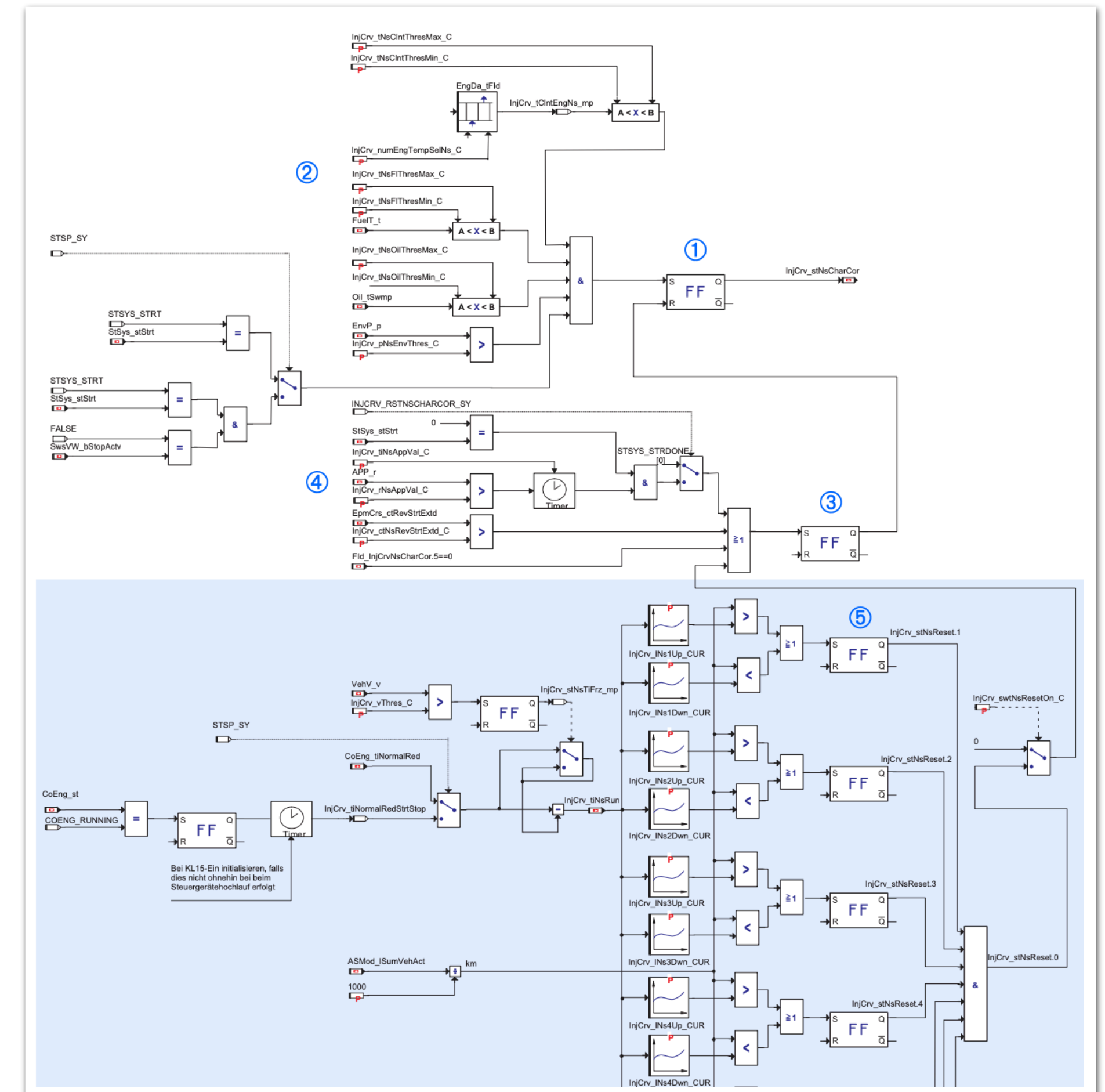
“DieselGate” and other “Defeat Devices”

How They Did It: An Analysis of Emission Defeat Devices in Modern Automobiles

Moritz Contag*, Guo Li†, Andre Pawlowski*, Felix Domke‡, Kirill Levchenko†, Thorsten Holz*, and Stefan Savage†

* Ruhr-Universität Bochum, Germany, {moritz.contag, andre.pawlowski, thorsten.holz}@rub.de
† University of California, San Diego, {gul027, klevchen, savage}@cs.ucsd.edu
‡ tmbinc@elitedvb.net

In this paper, we present our analysis of two families of software defeat devices for diesel engines: one used by the Volkswagen Group to pass emissions tests in the US and Europe, and a second that we have found in Fiat Chrysler Automobiles. To carry out this analysis, we developed new static analysis firmware forensics techniques necessary to automatically identify known defeat devices and confirm their function. We tested about 900 firmware images and were able to detect a potential defeat device in more than 400 firmware images spanning eight years. We describe the precise conditions used by the firmware to detect a test cycle and how it affects engine behavior. This work frames the technical challenges faced by regulators going forward and highlights the important research agenda in providing focused software assurance in the presence of adversarial manufacturers.



<https://cseweb.ucsd.edu/~klevchen/diesel-sp17.pdf>

Notable Examples of “RE for Good”

DragonSec RE of Anti-Repair Code in Newag Train Firmware

```
check1 = 53.13845 < lat && lat < 53.13882 && 17.99011 < long && long  
< 17.99837;  
check2 = 53.14453 < lat && lat < 53.14828 && 18.00428 < long && long  
< 18.00555;  
check3 = 52.17048 < lat && lat < 52.17736 && 21.53480 < long && long  
< 21.54437;  
check4 = 49.60336 < lat && lat < 49.60686 && 20.70073 < long && long  
< 20.70840  
    && (this->lock_function_test & 1);  
check5 = 53.10244 < lat && lat < 53.10406 && 18.07817 < long && long  
< 18.08243;  
check6 = 50.12608 < lat && lat < 50.12830 && 19.38411 < long && long  
< 19.38872;  
check7 = 52.77292 < lat && lat < 52.77551 && 18.22117 < long && long  
< 18.22724;
```



<https://zaufanatrzeciastrona.pl/post/o-trzech-takich-co-zhakowali-prawdziwy-pociag-a-nawet-30-pociagow/>

But: RE is Not Accessible

- All of the (extremely cool) analyses just discussed required enormous amounts of work by dedicated specialists
- There is too much software in the world, and too few reverse engineering experts, for this to scale
- Even most “automated” analyses require significant expertise to use and may not scale to real software

The Vision



Brendan Dolan-Gavitt ✓

@moyix



Speculating wildly, it feels like we may be getting close to something I've dreamed about since grad school – making the software running on your computer inspectable and understandable to the end user.



Brendan Dolan-Gavitt ✓ @moyix · Mar 31, 2023

Entirely anecdotal, but ChatGPT+GPT-4 was *extremely* helpful recently for figuring out why a closed-source program on Mac was crashing. I just attached Xcode to it and fed in chunks of assembly, asking what the original ObjC/Swift code might have been, until I hit root cause.

7:51 PM · Mar 31, 2023 · **8,977** Views

The Vision



Brendan Dolan-Gavitt  @moyix · Mar 31, 2023



Imagine an interface that let you point to something and say "what does this button do?" and it would pull out the relevant ASM, use GPT-4 (+trad reverse engineering tools?) to summarize its functionality, and produce an explanation linked to the original assembly.



1



19



1.6K



Brendan Dolan-Gavitt  @moyix · Mar 31, 2023



You could maybe also combine this with some kind of whole-system record/replay (with a smallish event buffer) so that you could ask questions events that happened on your system, like "Why did that window just pop up?"



2



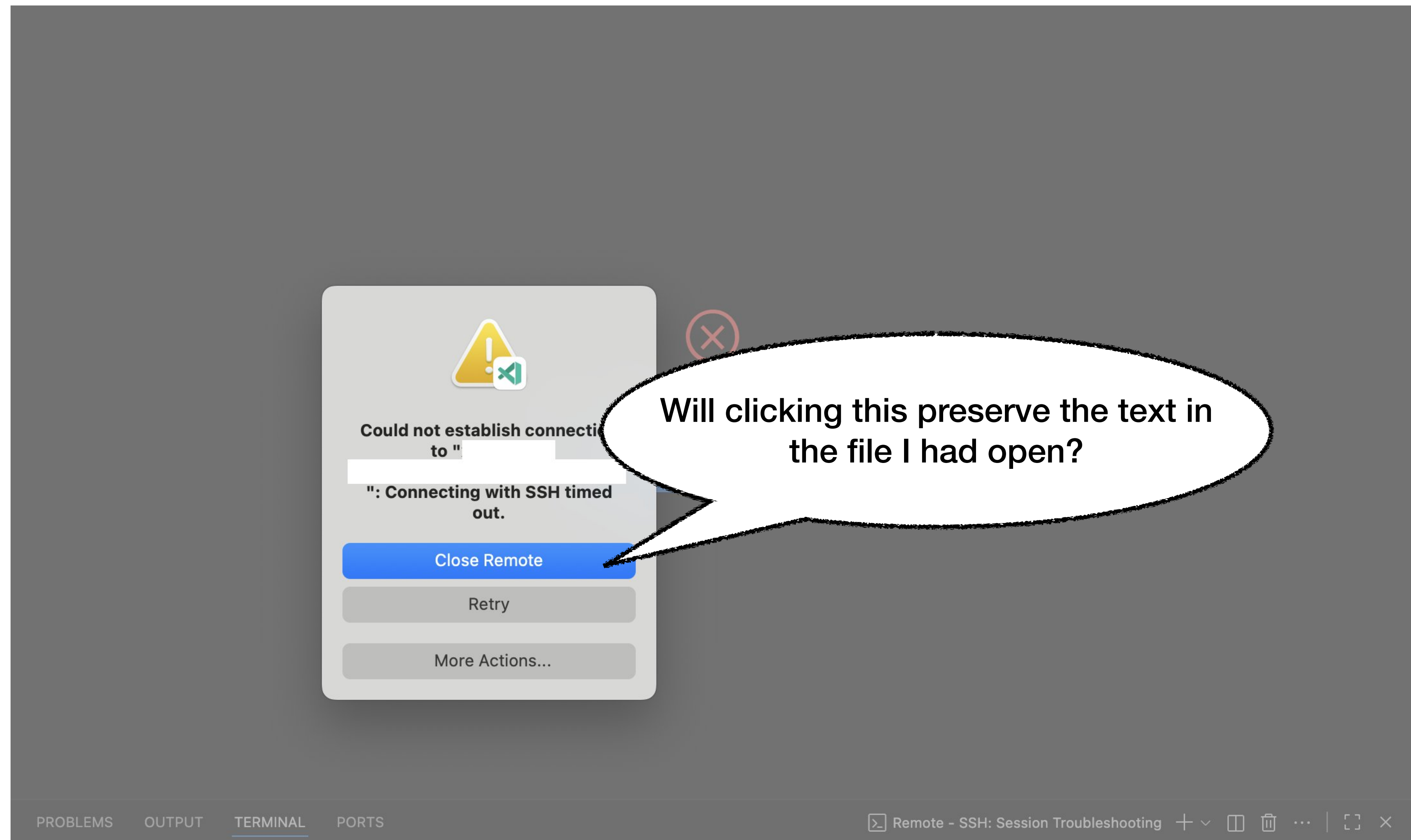
15



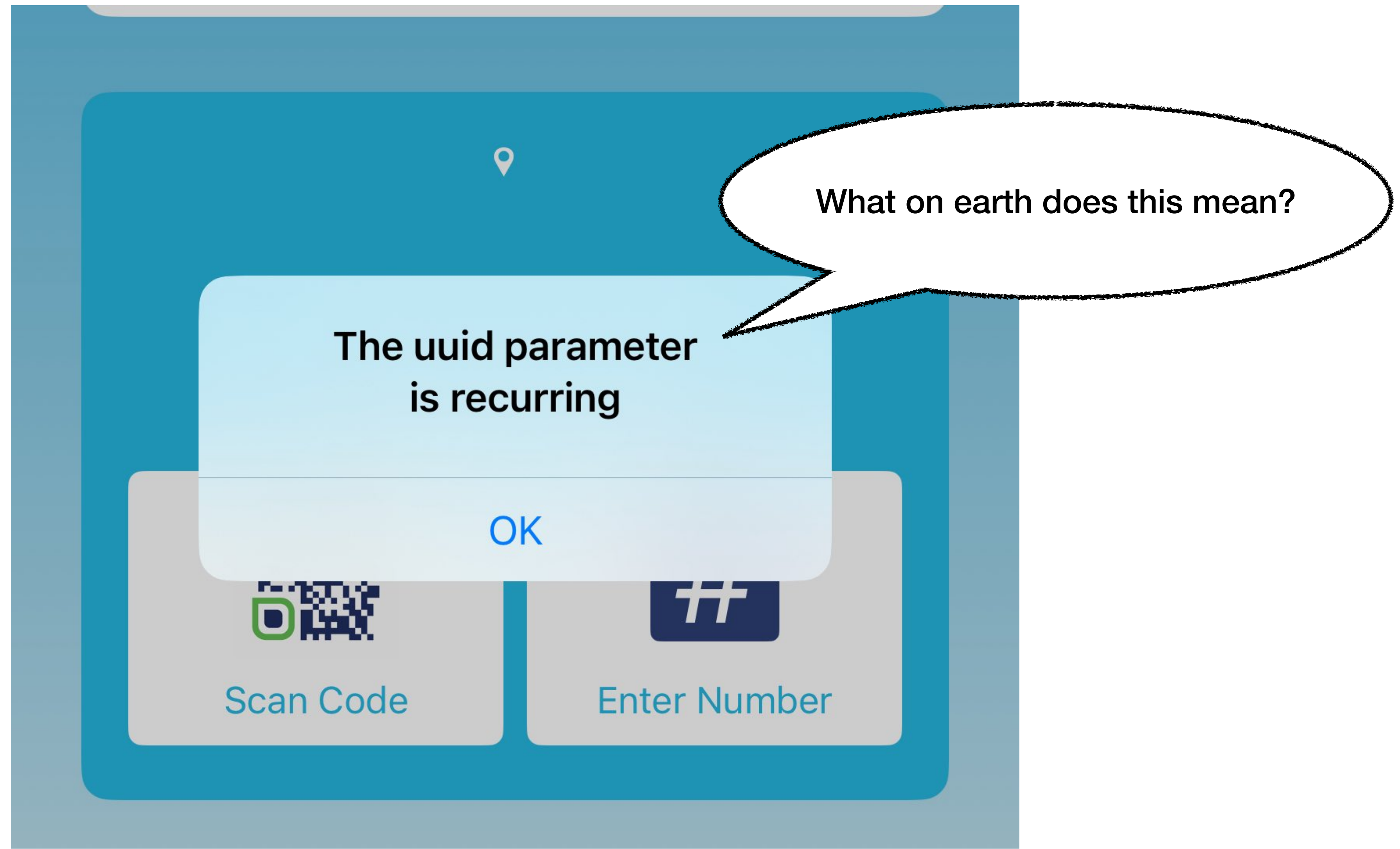
1.4K



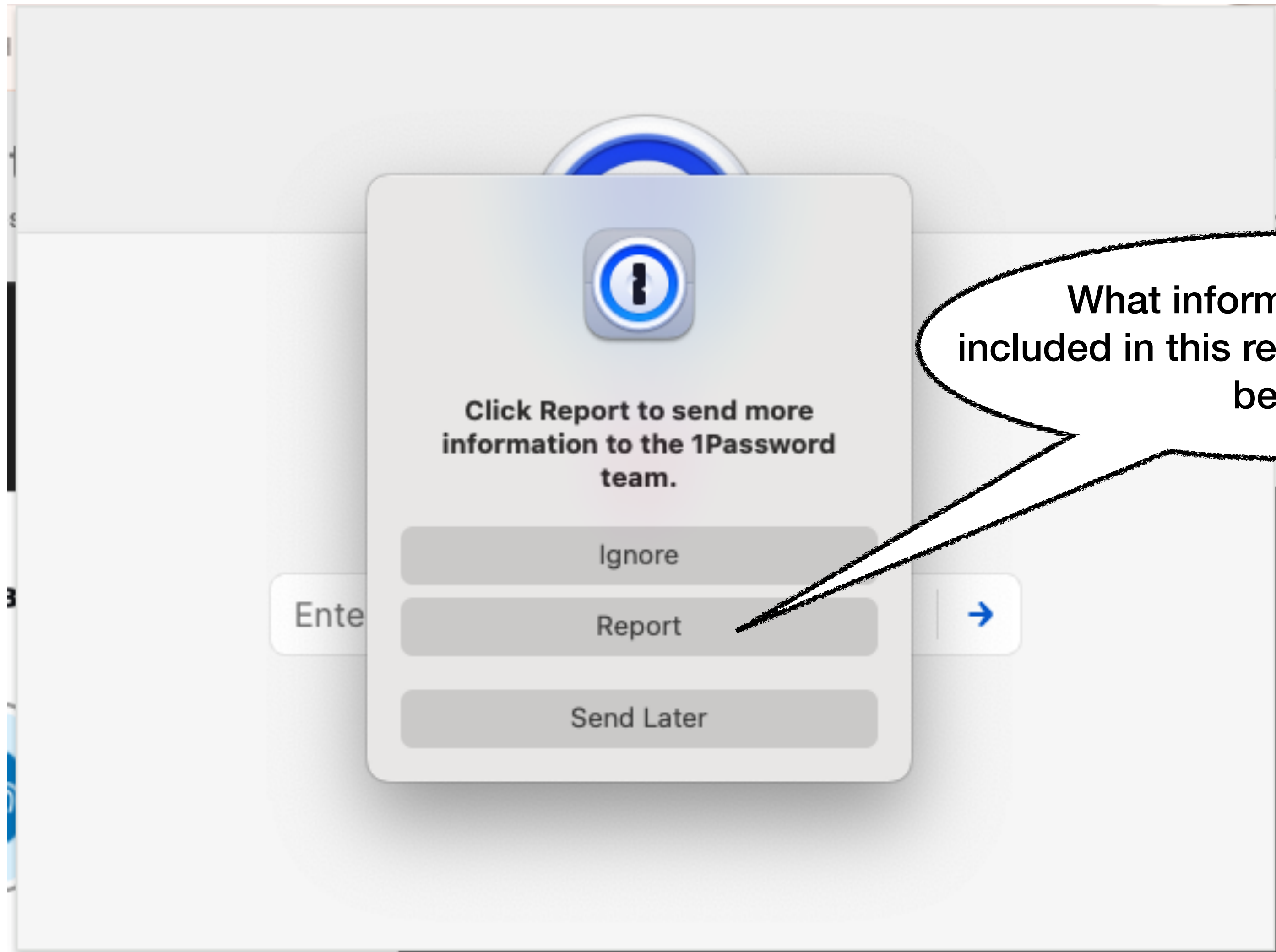
The Vision



The Vision



The Vision



What information would be included in this report, and where will it be sent?

Early Signs of Life?

Full transcript: <https://chat.openai.com/share/b41bcd92-80f8-4aa4-9086-e8aede516ace>

- Anecdote: I was doing my taxes with TurboTax (closed source) on OS X, but it kept crashing at a particular point
- Using ChatGPT4, I was able to:
 - Get instructions for attaching the XCode debugger and getting a backtrace
 - Feed the backtrace to GPT4 to identify the problematic code
 - Decompile functions along the way to Objective C / Swift
 - Identify and fix the problem (a missing JSON file in the installation)
- (I don't have much experience with OS X or Swift/Objective C reverse engineering)



Crash Source Identification



The crash occurs on the main thread (thread #1) with the stop reason: **`Fatal error: Unexpectedly found nil while implicitly unwrapping an Optional value`**. The issue seems to be related to the unwrapping of an optional value without proper checking for **`nil`**.



The relevant stack trace line is:

```
`frame #6: 0x0000000107814d1d TurboTax 2020`static  
TurboTax_2020.FDPContingencyHandler.readAndReturnExperimentInConfig() ->  
Swift.Dictionary<Swift.String, Swift.AnyObject> + 637`
```

This line is within the

```
`TurboTax_2020.FDPContingencyHandler.readAndReturnExperimentInConfig()`
```

function, which returns a dictionary. It appears that the crash is due to an implicitly unwrapped optional value being **`nil`** when it was expected to have a value. To resolve this issue, it's necessary to find out why the optional value is **`nil`** and ensure proper handling of optional values within the code. This can be done by using optional binding (if let or guard let) or providing a default value with the nil-coalescing operator (??).

Swift Decompilation



It's quite challenging to reconstruct the exact original Swift code from assembly, but I can provide you with a rough outline of what the code might have looked like. The details and syntax might be different from the actual code. Here's an attempt to reconstruct the code:



swift

Copy code

```
class TurboTax_2020 {  
    // ...  
  
    static func FDPContingencyHandler() -> [String: AnyObject] {  
        var dictionary: [String: AnyObject] = [:]  
  
        let experimentServiceController = ExperimentServiceController()  
        let experimentSettings = experimentServiceController.loadExperimentsS  
  
        if let settings = experimentSettings {  
            // Code to process experimentSettings and populate the dictionary  
            // ...  
        }  
  
        return dictionary  
    }  
}
```

Making Binary Systems (More) Transparent

- I don't actually think LLMs are ready to do all of this on their own
- More reliable, *deductive* systems are still needed to provide the meat of the analyses
 - Lightweight record-replay to keep track of things that are happening
 - Control and data flow analyses to understand which components are involved and identify the code that needs to be examined
 - Semantics-preserving decompilation
- But LLMs may provide helpful glue at the input and output portions:
 - Turning user queries into a sequence of calls that carry out automated analyses on binary code
 - Digesting the results of reliable analyses into an understandable (but accurate!) answer

Challenges (Research-Oriented)

- Real systems are a complicated mess of different technologies
 - Native libraries, JS code, bytecode, IPC, calls to remote servers
 - Even for binary code the “style” of the object code can vary significantly depending on source language, application domain, etc.
- *Performant* whole-system record/replay basically doesn't exist
 - Though with some hardware extensions it seems feasible
- How can we make sure the LLM bits faithfully translate what the deterministic/symbolic components provide?
- Tons of interesting UX and agent design challenges in putting all the pieces together and making them work on real software

Challenges (non-technical)

- Who will pay for this? Not much of a business model.
- Requires significant device privilege to collect and inspect relevant data
 - Trend has been toward taking this away from end users!
- If it becomes popular, we can expect anti-RE measures to become more common in “regular” binary software
 - But maybe not ubiquitous? The web is currently very “inspectable” by default, but obfuscation is still not the norm

Conclusions

- Reverse engineering has come a long way since I started my PhD in 2008
- We've gone from a culture of unreproducible, specialized one-offs to powerful tools that work on real software and can be built upon
- I am very excited and optimistic about the potential to finally take automated reverse engineering out of the lab and put it in the hands of people who can benefit from it most
 - A bit sad that I probably won't be the one doing most of this work!
 - But reassured to see a workshop like this carrying on the torch of excellent research in binary analysis