



Bauhinia Newsletter

Vol. 02 • March 2025

Vol. 02

Contents

Generative AI "Artwork"
Windows AMSI / ETW
Coppersmith's Method
Discovering DOMPurify CVE
Tcache Poisoning
High Rank Elliptic Curves
Function Interposition for Dynamic Analysis
Neural Network Adversarial Attacks
...and more



Table of Contents

Table of Contents	1
Foreword	2
Ad-Hoc	
Puzzle – <i>Mystiz</i>	3
Isekai Tensei Hakka Vol 1 Issue 3 – <i>Ozetta</i>	4
Isekai Tensei Hakka Vol 1 Issue 4 – <i>Ozetta</i>	6
Knowledge	
How to hack a signage using a \$9 remote – <i>a1668k</i>	8
AI “Artist” (1): Text-to-image – <i>apple</i>	10
AI “Artist” (2): Prompts & Inpainting – <i>apple</i>	12
APT techniques studying – <i>botton</i>	14
Introduction to Coppersmith's Method (simply) – <i>Eason</i>	16
Analysis of Web Applications From a Noob's Perspective – <i>ensy</i>	18
手把手解 Heap pwn: Tcache Poisoning – <i>gldanoob</i>	20
A Primer on Searching for High Rank Elliptic Curves – <i>grhkm</i>	22
IDA Tips 1: Custom Structures – <i>harrier</i>	26
Technical Minecraft - Chunk Loading I – <i>hoifanrd</i>	28
Decrypting UNIX-based OpenSSL TLS Traffic – <i>vikychoi</i>	30
Introduction to Adversarial Attacks: Fast-Gradient Sign Method – <i>Vow</i>	32
Events	
SINCON 2024 Recollections – <i>Gon7K</i>	34
Credits and Afterwords	36

Foreword

Welcome to the second public edition of our newsletter, presented by Black Bauhinia (blackb6a) team members. Black Bauhinia is a Capture-the-Flag team from Hong Kong founded in 2019 and have been actively participating in CTF games since then. Whether you're an industry expert or a student, we hope this newsletter will inspire you about different aspects of CTF and the cybersecurity landscape.


What is CTF?

Capture The Flag (CTF) is a popular type of cybersecurity competition that challenges participants to solve various puzzles and problems to capture hidden "flags". Often, players are required to break a system and workaround the security measures to get the flags.

CTFs are designed to simulate real-world cybersecurity scenarios, providing a platform for learning and demonstrating skills in a fun, competitive and legally safe environment.

About Black Bauhinia

Black Bauhinia is a CTF team from Hong Kong dedicated to advancing cybersecurity knowledge and skills. Our mission is to foster a community of learners and professionals who are passionate about cybersecurity and eager to tackle new challenges.

 **Black Bauhinia**


Also known as

- BlackBauhinia
- blackb6a

Website: <https://b6a.black>

Twitter: [blackb6a](#)

[Sign in](#) to join the team.



A team based in Hong Kong.
(We are NOT affiliated with any associations.)

Participated in CTF events

2025	2024	2023	2022	2021	2020	2019
------	------	------	------	------	------	------

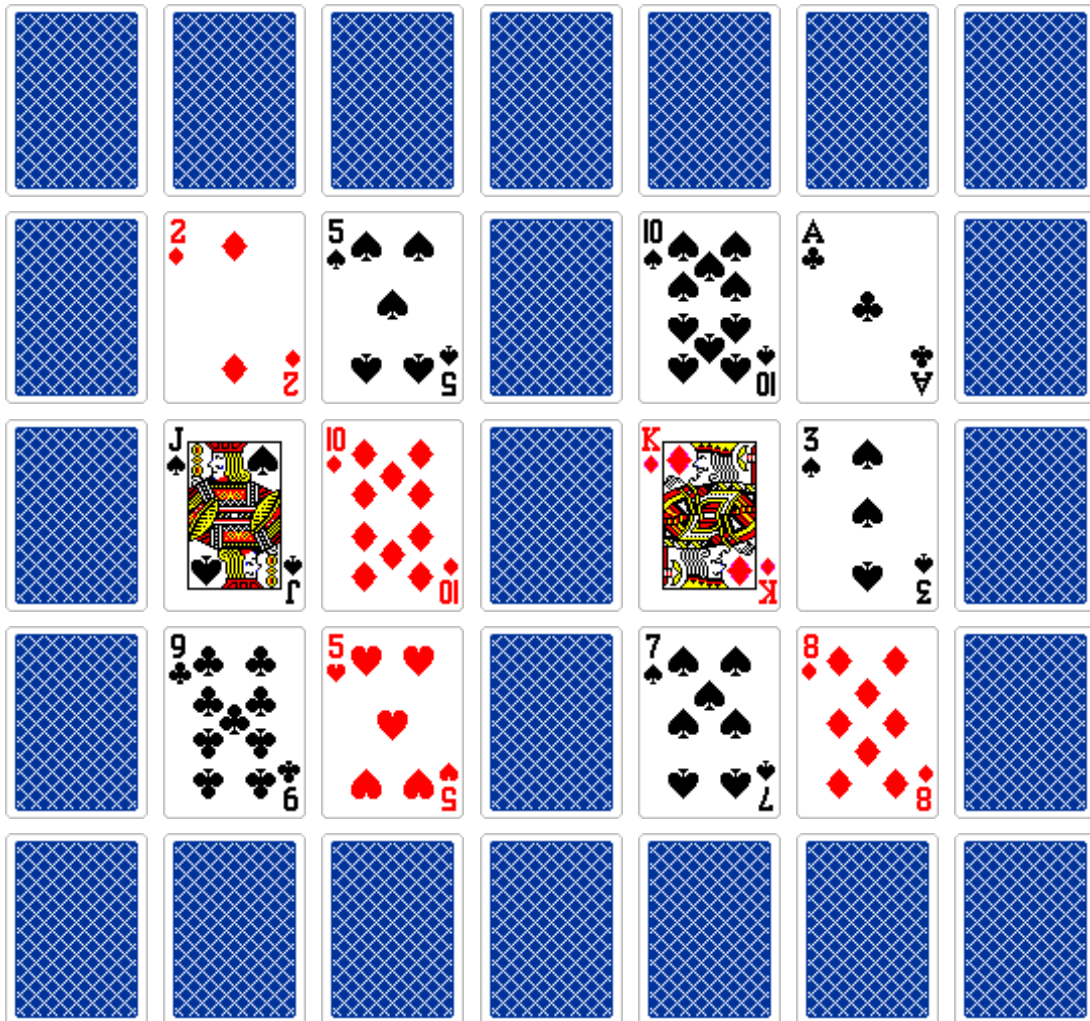
Overall rating place: **54** with **417,670** pts in 2024

Country place: **1**

Our team also co-organized the BSidesHK 2025 CTF. We hope it would be the perfect chance to encourage newcomers to dive into the world of CTFs and have fun. We extend our heartfelt gratitude to BSidesHK 2025's organizer, participants, and supporters who make this possible. Join us in our journey, and let's capture the flag together!



What are the cards facing *down*? 



*Pick five cards to form a hand and compete with the team!
Send a direct message to Mystiz on the cards you picked.*



異世界轉生黑客

Isekai Tensei Hakka

威噏變久噏？以 Black Bauhinia CTF Team 為題材的輕小說正式拖稿登場！如果你想 (or 唔想) 你出現在本小說中，請 Send 1BTC 畀 @ozetta，你的意見有可能被接納。

第一卷 — WOG FFA Battle

付費即可下載無刪減版

參考資料：

<https://github.com/blackb6a/blackb6a-ctf-2023-challenges/tree/main/20-isekai-tensei-hakka>

第三章 — 佔領中央大陸風雲榜

當我看到世界冠軍是如此遜色的時候，我便手賤按了一下挑戰冠軍。
「挑戰冠軍需付3000元費用」系統又出現奇怪的訊息。「挑戰冠軍最少需5000」
突然又彈多了一個對話窗，系統你未食藥？結果我先回到水晶之間掃蕩一番。
系統又猝不及防地出現了奇怪的訊息：「站長要考驗大家是否有認真在玩」
你看我認真到把自己的能力值改滿，還有誰比我還認真？
下面有個疑似 CAPTCHA 的老土圖片，然後牠叫我「請輸入安全驗證碼」。
我只好無奈地輸入。那個提交按鈕還有個奇怪的標籤「填好答案了，放我過關吧!!」
按了一下，又彈對話窗「恭喜答對了」，多X謝晒喎。
結果我在水晶之間奮戰幾分鐘之後（當中幾乎是等候的時間），終於儲夠錢了。
我交上 3000 大元去挑戰冠軍。結果很明顯，一回合也不到就順利當上冠軍了。
我滿意地看著世界冠軍的告示板。唯一美中不足的是那個英雄檔案是我亂址的主頁。
「滿意啦？」謎之音對我說。「唔滿意囉，打一場又要等，又要磅水，咁麻煩。」
「你去睇下風雲榜？」謎之音慫恿地說道。我又找到了一個叫風雲榜的告示板。
這裡竟然寫著各項屬性值的排行。我毫不意外奪下了各項的榜首。
「等級 202、hp 5277481、物理攻擊 65535、魔法攻擊 65535、速度65535...」
最底有一項叫「好野人 top」，裡面都是一些有錢人，窮如我看不見自己的蹤影。
「為什麼叫『好野人』？係咪同表姐你好野有關？」
「你去問呂■■囉，台灣話我識鬼咩」隨後系統畫面浮出一個 :bcgugu: 的表情符號。

於是我開始想辦法搵真銀。如果只靠冒險應該打到佛誕都似。不如試下去賭場？
在風雲榜旁邊有一個賽鳥場。我也不知道這賭檔是怎麼運作的，於是我向右摔了下。
我找到了 `class/wog_etc_race.php` 的原始碼。在第 40 行，投注金額如下：

```
$money=(int)trim(addslashes($_POST["money"]));
```


多麼複雜的處理呢.....看來這裡的賭場不太歡迎我這種滲透術士。
之後我再用 Sublime Text 找找哪裡有可以利用的 `update wog_player` 指令句。

其中 `class/wog_act_pk.php` 的 `pk_setup` 有點有趣。看來是跟X街有關？
「你有錢設定PK金額咩？」謎之語輕蔑地說。我回了它：「設定完咪有錢」
隨後系統畫面出現了 :blobcatthinkingglare: 的表情符號。我心諗：「慢慢諗啦諗樣」
我再三確認一下這個漏洞是否可被利用。首先 `pk_setup` 會在第 28 至 35 行檢查，
檢查 `$_POST["pk_money"]` 有沒有填寫和是否數字。反正我對這東西也沒興趣。
之後在第 37 行會先找出玩家現有的金錢，再於第 39 行檢查是否有足夠金錢，
以及如果參加PK的話，PK金額是否於 1000 至 100000 之間。反正我也不想 PK。
最後在第 44 行執行以下指令句：`$DB_site->query("update wog_player set p_pk_s=".$_POST["pk_setup"].",p_pk_money=".$money." where p_id=".$user_id."");` 而 `$_POST["pk_setup"]` 看起來可以改的。
把它從數字改成「數字,欄位=數字」就是典型的 SQL 注入攻擊來更新其他欄位。
我隨後去到 PK設定頁，把 `pk_setup` 的數值由 `0` 改成 `0,p_money=9e99`。
再把金額隨便填成 1000，然後按下確定按鈕。系統提示：「設定成功!!」
我再看看自己的角色狀態，發現金錢果然變成了 4294967295。發達啦 ATM
再去風雲榜看看排名，不出所料我也變成了好野人 TOP 的第一名了。
發咗達梗係去包間總統套房啦。於是我按了住宿.....
結果系統還是老土的顯示「休息了一晚後,HP回復精神跑滿!!」
「喂系統，說好的異世界度假呢？」我不滿地叫嚷著。謎之音：「嚟啦喂嚟啦喂」

我回過神後，床邊突然堆滿了錢。「好好笑㗎」我不滿地說著。
然後我看看系統畫面，結果竟然是這個房間的畫面。最令我詫異的是我竟然無著衫？「你依家先知你無衫著咩？」謎之音冷冷地嘲笑著。

為了這故事不會被淫審處 Bam，我立刻跑到PK設定弄幾套衣服。
你問為什麼是PK設定而不是裝備欄？既然錢可以變出來，衣服也可以吧.....
我匆忙地把 `pk_setup` 的數值由 `0` 改成 `0,d_body_id=118`，之後按確定。
我再看看角色狀態。好像沒甚麼分別？「你試下登入多次啦」謎之音回應著。
我隨後再登入一次，發現身體多了裝備了！多了個.....「究極緞帶」.....

.....原來我把身體的裝備設成了頭部的裝備.....我自暴自棄的穿著緞帶走在街上。
路人也沒有用甚麼奇異的目光看著我，看來這裡的 NPC 已經習慣了玩家的品味。

冷靜了一下後，我打算把風雲榜其餘的位置都佔據，包括勝場、PK WIN和鳥奪冠。
另外等級和 hp 也可以改大一點。之後我又去到PK設定操作一番：把那個 `0` 改成
`0,p_win=9e99,p_pk_win=9e99,p_cho_win=9e99,p_lv=9e99,p_exp=9e99`
`,p_nextexp=1,p_hp=9e99,p_hpmax=9e99,p_bank=9e99`
順便再弄點個銀行存款吧。按確定後，周圍的景色突然閃了一下。再去風雲榜看看：
勝場、等級、hp、鳥奪冠都是 4294967295、PK WIN 則是 2147483647。
「開心啦？滿意啦？」謎之音不屑地說。我回答：「未滿㗎，魅力果啲都未滿。」
「順便轉埋做大賢者好無？」謎之音提議到。我回答：「哦...ch_id=30，得啦」
改成 `0,p_vit=9e99,p_luck=9e99,p_au=9e99,p_be=9e99,ch_id=30`
確定後再查看角色狀態，我說：「做乜重係術士？」謎之音：「你都係登入多次啦」
重新登入後職業變成了大賢者。我心諗大賢者和術士有甚麼分別？反正能力都是滿。



異世界轉生黑客

Isekai Tensei Hakka

威囑變久囑？以 Black Bauhinia CTF Team 為題材的輕小說正式拖稿登場！如果你想 (or 唔想) 你出現在本小說中，請 Send 1BTC 畀 @ozetta，你的意見有可能被接納。

第一卷 — WOG FFA Battle

付費即可下載無刪減版

參考資料：

<https://github.com/blackb6a/blackb6a-ctf-2023-challenges/tree/main/20-isekai-tensei-hakka>

第四章 — 次元攻擊

WIN 4294967295 / LOST 0

英雄檔案



復仇率: 100%

力量

敏捷

魅力

體質

物攻

物防

男 0歲

毒

27295

究極鐵指

我決定不再理會那堆樸實無華的數據，好好地享受一下異世界生活。
首先我想食飯。可是原始碼所顯示的世界竟然連個像樣的食物也沒有。
那裡只有提及到銀鯨仕事所的酒吧老闆娘，又不是酒吧薯條，食得咩？(黃韋建: 娣)
還有道具屋賣的 hp 回復劑，是另類的辟谷丹嗎？我只好在街上閒逛看看有沒有食店。
我好不容易才找到一間中世紀歐陸風的食店，進門後拿起兩個金幣給店員。
以我對異世界的認知，一個金幣等於一百個銀幣，一個銀幣等於一百個銅幣。
這就像麻雀裡的一筒代表一個銅幣，一索代表一百個銅幣，一萬就代表一萬個銅幣。
而一個銅幣的價值應該跟一百日元差不多。兩個金幣應該差不多值十萬港幣啦。
店員看到兩個金幣之後兩眼發光，隨後問我：「客官想要啲咩？」
我問她：「有無 Omakase？」店員回答：「有！套餐包主菜和飲品，OK 嗎？」
我表示了 OK 的手勢後，店員就開始準備食物了。等了十五分鐘後，終於上菜了。
擺在我眼前的竟然是焗豬扒飯和凍檬茶。說好的中世紀歐陸風呢？
還有這兩樣東西值十萬港幣？會唔會過份咗啲？「得咁多咋？」我不愉快地問道。
「係啊客官，最近通漲得好犀利啊。銀行話唔知點解印多左四十三億銅幣㗎？」
聽到她這樣一說，我突然飆了冷汗……心裡想起那個 `p_bank=9e99` 的東西。
「客官你唔舒服牙？」我作狀從緞帶裡拿出一個金幣問：「加多個湯夠唔夠？」
「夠！梗係夠！找返你七十個銀幣……」我心虛地說：「唔使找啦，當畀貼士。」
店員禮貌地道謝後就去了準備食物。後來店員送來了一盅老火湯。我都不想吐嘈了。

飲飽食醉之後，我走出街上再逛逛。反正這異世界這麼離譜，說不定還有音驚機舖。
正當我在努力的尋找異世界的秋葉原時，我感覺身後好像有一班黑衣人跟著我。
我心想「想跟蹤敏捷 65535 的我，無咁易」隨後我以飛快的步法跑到去一個後巷裡。
「哈哈，重唔畀我捕到你。」眼前一名穿著黑色西裝的男人 XXXXXXXXXX 看著我。
然後他從口袋裡拿出一把短劍，「好耐無見㗎 ozetta」，他囂張地叫嚷著。
我思考了一會後，想起眼前的人正是多年沒見的 Dr. ROT10。「哦，原來係付伊笙」
XXXXXXXXXX 他傲驕地說。我想起上次塵夏幟說的同一番話，難道這真的是夢？

我狡猾地回應。突然謎之音傳出：「正經啲啦」 Dr. 突然周圍環境一黑。「唔掂啦要 Rollback」謎之音怒氣沖沖地警告：「呢度係劍與魔法的數碼世界，唔係 言情小說」

.....
「好耐無見喎 ozetta」，一名手持短劍身穿黑色西裝的男子囂張地叫嚷著。
「你也水啊？想點啊？」我驚訝地說。男子回應：「你唔認得我啦？我想打劫啊。」
「你睇我成身上下有咩可以打劫？」我不屑地回應。
.....眾人一片沉默。連謎之音也沉默了。「唉呢個腐醫生真係無好帶挈，Skip啦」
隨後天上的雲朵突然郁得好快。那男子也突然郁得好快。65535 敏捷的我也追不上。
然後眼前被一陣白光包圍，

突然系統畫面出現了一個提示，我向下摔了一下畫面，寫著「主線任務」
謎之音興奮地說：「終於嚟啦，快啲揸入去睇下」，「任務一：殺死 Dr. ROT10」

我隨後找到了於未知區域的 Rotten，然後用偵查對手功能看看。「竟然只有 10 級」我再向他發起 PK 挑戰，結果系統彈出了對話窗：「對手拒絕 PK 或 PK 條件不符合」我只好再利用 PK 設定 0,p_lv=10 來把自己的等級改到 10 級來滿足系統的條件。毫無懸念我在一回合內把他殺死了。「ozetta 獲得了勝利！！HP 剩下 4294967295」我的等級也從 10 級升到 1195 級。但是我不是殺了那渣為什麼沒系統提示？隔了一會，在競技場一旁忽然冒出了一陣白光，Dr. ROT10 從那完好無缺地站起來。
「哈哈 ozetta，你殺唔死我架」

.....眾人一片沉默。為了打破沉默，謎之音說：「唉我無眼睇啦」

為了不再被系統 Skip 掉重要場景，我以 65535 敏捷快速逃離競技場。
如果普通攻擊殺不死玩家，那試試次元攻擊？連裝備編號 70 的次元之鎧也擋不著。「你知道有鎧甲點解重要著條爛鬼緞帶？」謎之音極度不滿地吶喊著。我沒有理它。
先試試奪舍大法吧，我試試把 Dr. ROT10 的密碼偷來看看。我把 PK 設定的值改成
0,p_sat_name=(SELECT p_password FROM wog_player WHERE p_name='Rotten')，按下確定後系統彈出了對話窗：「有不正常符號(1)」.....
原來我忘了把單引號拿掉。那我應該怎樣避免使用「不正常符號」呢？方法很簡單，只要把 'Rotten' 改成 0x526F7474656E 就好了。我再點了一下，結果沒反應。我再看看開發人員工具裡 Network 的內容，發現系統出錯了：

You can't specify target table 'wog_player' for update in FROM clause

那我只好再加一層 Subquery 了：0,p_sat_name=(SELECT p FROM (SELECT p_password p FROM wog_player WHERE p_name=0x526F7474656E) x)
確定後再重新登入就看到必殺技名稱變成了 pvkq{Lido}。果然是個老土的密碼。
我用這密碼登入了 Rotten 的帳號，周圍的環境也變了。

我登入回 ozetta 的帳號，周圍的環境也變回我住的總統套房。隨後我走到帳號中心，在「角色自殺」裡把 Rotten 的帳號和密碼填下並提交。「刪除成功!!」任務完成！



How to hack a signage using a \$9 remote

a1668k

Recently, I have been working on a research project related to signage hacking. One of the most interesting findings I found is about the Infrared (IR) sensor. I discovered that I can nearly fully control all the Android signages using an IR transmitter, given that we found the correct sets of codes and the signage contains an IR sensor.

If you read Bryon's "Flipper Zero 推坑簡介" in public volume 1, he said he can control the projector using the infrared module inside Flipper Zero. I also used Flipper Zero to do the attack. Besides that, an even more interesting thing is that I bought a \$9 universal remote in Apliu Street, and it works too UwU...

So... How IR remote control works? How can we control these devices using IR remote control?



Almost all the signages had an infrared signage near the bottom right of the screen. If you saw there is a light there, most likely is there.

Code of Ethics: Again, it is very likely that you can control most of the Android signages using an IR transmitter, e.g. Flipper Zero, or a \$9 universal remote control. So please don't attack those signages you saw in the public after reading this article :D I am not responsible for that :D



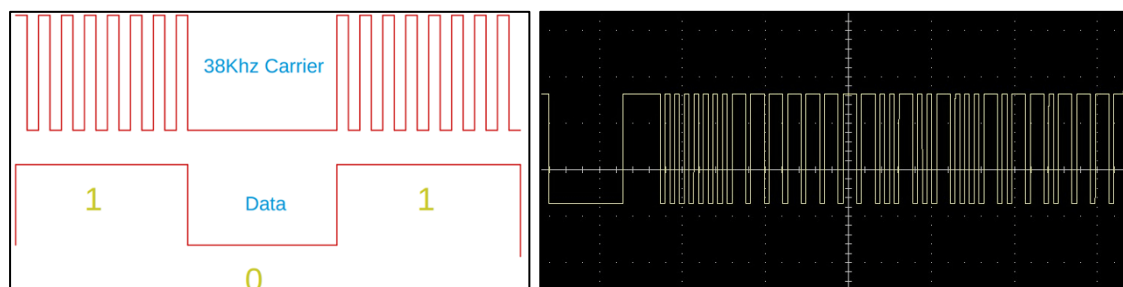
If the remote is functioning properly, you will see a light if you try to capture it using a digital

Before we dig into how to control devices containing IR receivers using IR remote control, let's just talk about how IR remote control works first.

From the name of the remote controller, an IR remote control works by using infrared light to send signals to a device (X bilge). However, as there are millions of infrared lights everywhere, IR receivers will continuously receive all those IR signals when it is on. Therefore, IR receivers will need a way to filter out the correct IR signal and perform actions based on the IR signal.

There are various types of IR protocols out there, but the most common ones are **NEC Remote Protocol** and **RC5 Remote Protocol**. And for all the Android signages I tested, they used NEC Remote protocol. Therefore, let's take NEC as an example.

Like most of wireless signals, the NEC code uses a carrier frequency of 38KHz to avoid interference. The actual data is modulated using 38KHz (26.3μs) modulating frequency. When a button on the remote is pressed, the IR Blaster sends a stream of data to the receiver. And the receiver will process it to retrieve the address and command.



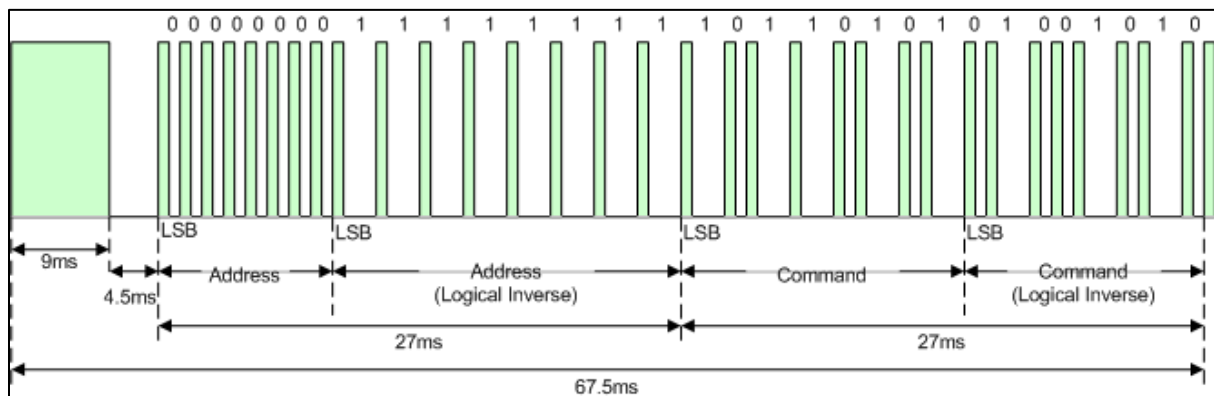
Left: illustrate the modulating frequency, Right: the real IR signal captured under oscilloscope

Reference: <https://circuitdigest.com/microcontroller-projects/build-your-own-ir-remote-decoder-using-tsop-and-pic-microcontroller>

The NEC format contains multiple sections, including:

- A 9 ms leading pulse burst (/ carrier waveform)
- A 4.5 ms space (/ OFF waveform)
- The 8-bit **address** for the receiving device
- The 8-bit logical inverse of the address
- The 8-bit **command**
- The 8-bit logical inverse of the command
- A final 562.5 μ s pulse burst to signify the end of the message transmission.

The following example illustrates the format of an NEC IR transmission frame, for an address of 00h (00000000b) and a command of ADh (10101101b).



Reference: <https://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol>

With the knowledge of how IR remote works, if we know the **address** and **command** of a specific brand of IR receiver, we can try to send an IR signal following the format of that protocol. Tools like Flipper Zero's Infrared module can do the job by writing IR signal files specifying the address and the command. You can also search for these IR signal files from databases like Flipper-IRDB, or the "Universal Remote" function in Flipper Zero.

```
1 Filetype: IR signals file
2 Version: 1
3 #
4 name: Cursor Up
5 type: parsed
6 protocol: NEC
7 address: 00 00 00 00
8 command: 4d 00 00 00
9 #
10 name: Cursor Down
11 type: parsed
12 protocol: NEC
13 address: 00 00 00 00
14 command: 5a 00 00 00
```

A snippet of the IR signal file I wrote for controlling the Android signages.

Also, most of the IR receiver uses the same address for all the actions they have. Therefore, we can try to brute-force the value of the command after we found a correct set of address and command. In the research project, I found that all of the Android signages use the address of 00h or 40h. And I can perform nearly all the actions, like open settings, go to the home screen, or even turn off the signage, by brute-forcing the value of the command.

If you try to buy a universal remote in Apliu Street, they typically come with a database of codes for various TV brands and models. And you may be able to find a correct set of codes that can control the devices using the search function on the universal remote. (I used a \$9 universal remote xDD)

With the knowledge of these, you can now control almost all the devices comes with an IR sensor :D
If you want to know more about what else I have found during this research project, maybe this is the topic of my next article :D



AI “Artist” (1): Text-to-image

AI Image Generation is slowly adopting to different industries, especially advertisement.

For HongKongers, the most notable one must be the 社企友建樹¹ ads from the government as it is ... literally everywhere.

The technology is quite promising that - it already causing illustrators losing jobs to AI, serious concerns among human artists and the anti-AI atmosphere among them. Well as a tech person, we always trying on new stuffs (not new anyway)... ride on it and see if it works. Let's get our hands dirty.

Warming up your GPU

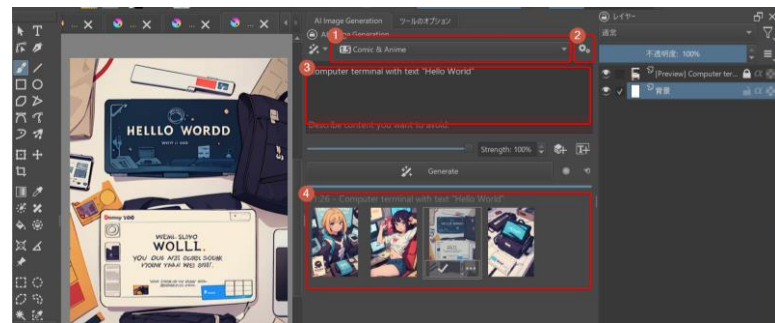
There are many services for image generation, and recently you can just use ChatGPT or Bing to generate image. I don't like online services, so instead a local version of Stable Diffusion (SD) would be used. The minimum requirement is a (reasonably) modern GPU with at least 4GB VRAM.

There are few UI for controlling SD, the popular choices would be [A1111](#) and [ComfyUI](#). Personally, I found ComfyUI is better... but I would recommend a third UI: [Acly/krita-ai-diffusion](#) which is a plugin for Krita, an opensource painting program. Follow the [documentation](#) and you can setup it to use your local GPU or remote GPU².

¹ Retrieved from [www.sehk.gov.hk](#)

² I am using a “Custom Server” setup connecting to my remote GPU server. Works flawlessly, and my laptop won't get hot. Using software from Internet at your own risk and I am not responsible for whatever problem you got into. I got it VM isolated.

Hello World: Greeting to Krita AI



After setting up, you can try “Hello World” for text-to-image (T2I), locally! Start by opening a 512*768px canvas and get started:

- ① **Model selection:** choose art styles (model checkpoint, LoRAs and style prompt). We chose the default “Comic & Anime”.
- ② **Settings:** Setup additional art styles, etc.
- ③ **Prompt:** Type image description as detail as possible for text-to-image generation³
- ④ **Preview:** Generated images would be shown here. Number of images depends on the image size and how powerful your GPU is. ~~Who can buy me some 4090?~~

Art style: Model Checkpoints

There are two main image generation base models, namely Stable Diffusion 1.5 (SD1.5) and Stable Diffusion XL (SDXL)⁴.

SD1.5	SDXL
Older model	Newer model
Trained on 512px	Trained on 1024px
Run faster	Run slower



Let's add a model: [AnythingV5](#) (SD1.5) from CivitAI, which was used for Cover of Vol.2

³ Prompt used: Computer terminal with text “Hello World”

⁴ Pony is also SDXL based, and it have its own selection of prompt keywords. Recently something called Flux.1 (F.1) appeared too, model file already ~15GB...

The model checkpoint⁵ plays a major role for the art style and understanding of prompt. Check footnote⁶ for installation instructions.



Apparently, compared to previous default “Comic & Anime” model, AnythingV5 have better idea of what a computer terminal is!



▲ During the training stage, all images were cropped to a square with 512 / 1024 pixels width and labeled with text description. Then, the T2I process starts with a random noise image from a random seed⁷, then mutate the image repeatedly for ~20 steps⁸ to follow the prompt. This process is called denoise. That’s why we set our canvas size to 512px as it is what the neural network trained on.

Positive and Negative Prompts

Enter the description for desired image to the (positive) prompt input box ③, as detail as possible, in English of course. For example:



You can see the images are nice as instructed by “good quality” prompt, but the model is not following the prompt well – most of the images got the eyes color wrong, some hair

color mixed with purple, and where the hell the cat ears come from? None of the image appears to be in morning too.

The cat ears might appear in our image due to tagging mistake – some “paw pose” images in the training data did not have “cat ears” as its description, or there is not enough paw pose images without cat ears. We can use negative prompt to get rid of cat ears and other undesired features, for example:

cat ears, nsfw, lowres, worst quality, low quality, blurry, high contrast, bad 3d, jpeg artifacts, text, signature, watermark

You can Google for more positive/negative prompts. Model understanding on your prompt depends on the training data quality.

But I want morning with moon and stars!

So, SD can’t follow our prompts correctly, especially when your taste is unique. The easiest way is to Gacha (draw) more images and try your luck. However, “morning” is conflicting with “moon”, so it won’t work. Fix it by increasing the strength⁹ of certain words in our prompt, simply add some brackets:



- ((((((morning, blue sky))))))¹⁰
- (morning, blue sky:1.5)

Can it replace human?

Is SD just copy-pasting from others drawing? Can SD generate new idea? Well, I believe human involvement is still the key for AI artworks. See you in the next time and ofcuz let me know if you have any ideas.

⁵ **Checkpoint (ckpt)**: the model file storing the tensors value (*.safetensors)

⁶ Download and place it to the ComfyUI “models/checkpoints” folder. Then, open the Settings dialog ② and add a Style Preset. Choose “AnythingXL_v50” (the filename) for Model Checkpoint and click OK.

⁷ **Seed**: the seed for random latent noise image to start with, same seed same image.

⁸ **Steps**: how many iterations to mutate the image incrementally, even 1 step works.

⁹ **Strength (weight)**: the importance of the keyword, typically +-1.5

¹⁰ 5 pair of brackets so strength is 1.1^5, same as (prompt keywords:1.61051)



AI “Artist” (2): Prompts & Inpainting

apple

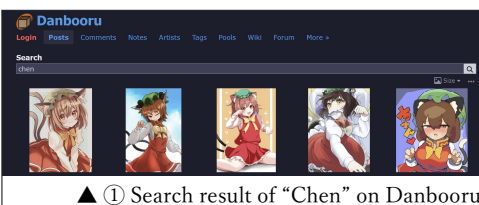


In the last article we had some basic idea on SD – which are just common sense right? Let's dive into more in depth topics:

What was it trained on

In the training stage, the SD model takes the images and text descriptions pairs to find relationship between them - word/phase are associated with some image features.

Where are those images sourced from? Well, it is mostly scraped (pirated) from Internet along with text descriptions. Part of the images are pirated from **Booru image boards**, which are image sharing & tagging site run



▲ ① Search result of “Chen” on Danbooru

by hobbyists. It was established in 2005 – way before AI images

is a thing and become a valuable source for AI engineers thanks to its quality tagging system (was tagged by a bunch of unpaid humans!)



Example ② human art¹ of Chen² on Danbooru:

Artist: konna reshiki, Copyright: touhou, Character: chen
General Tags: 1girl, solo, animal ear piercing, animal ears, cat ears, hat, earrings, single earring, jewelry, mob cap, brown eyes, brown hair, short hair, bow, bowtie, white bow, white bowtie, petticoat, frills, red skirt, puffy long sleeves, puffy sleeves, long sleeves, skirt, skirt set, vest, red vest, flat chest, hands up, sweat, speech bubble, simple background, white background
Meta Tags: highres, translation request
Rating: General (Safe for work), Score: 7 (quality)
(composition), (face), (body), (action), (background/layout)

That's detailed right? Now you know how the model define quality, SFW/NSFW, and what to write for your prompt as the anime style models were trained with these tags³.

How detail should my prompt be?

Can we generate our purrfect Chen-chan?



Left ③: The model got few common features of Chen from all the images that contains the tag “chen”, therefore you can see the distinctive cat ear, green hat, and the brown hair. However, the clothes, hair style, even the eye colors are completely different as it varies between images in the training data / have more association to other tags (this allows us to put Chen in other clothes!)

Right ④: Simply putting all the tags of ② as prompt and we got our Chen with similar outfit as the images ① and ②. But it couldn't get the white bowtie... probably due to most of the images with similar tags had yellow bowtie, as the original character design was yellow (see search result ①). Therefore, keep in mind that tags / phrases will interact with each other in some (unexpected) way, and SD will generate image that it sees fits.

Overall, ④ is better than ③ by making Chen-chan looks like Chen. So, try to describe your image in tags and natural language in detail, can ask ChatGPT to generate prompt too!

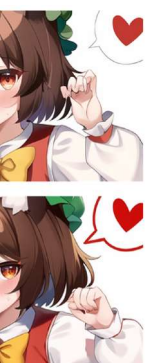
¹ Art by 紺奈れしき (@konna_resniki), retrieved from danbooru:7722676

² This good girl is called Chen (橙) from Touhou project. The original character

design was dated back to 2000 era so there are lots of fan arts!

³ In Krita AI, you can enable auto-complete for tags in Settings -> Interface.


Embeddings (Textual Inversion)

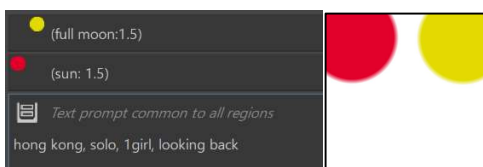


Sometimes you might find it hard to describe complex concepts that wasn't tagged in the model, and **embeddings**⁴ could solve this. You can find them from e.g. CivitAI and use them as prompt keywords. [EasyNegativeV2](#) and [badhandv4](#) are commonly used to get rid of unwanted image features and AI mistakes. As any prompt keywords, it would affect unexpected image features, e.g. image ⑤ the hat color is different, speech bubble got a red outline etc. See footnote⁵ for usage on Krita.

Gaining more control: Regional Prompt

When you want to have better control on the layout and detail - regional prompt can help.


Click the add region button  twice, then start drawing on the new region and type the prompt for the region. Prompt will only effect on the area that's not transparent.




As shown, we have successfully put two conflicting objects on the sky!

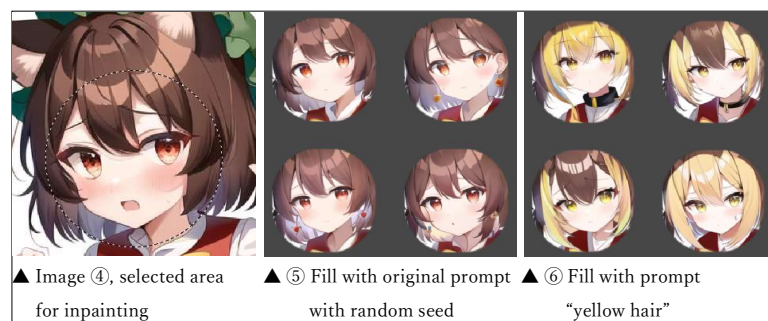
Partial Gacha: Inpainting (Generative Fill)

Say we are satisfied with image ④, but we just don't like the face. We can re-generate partial area with **inpainting**. Adobe is selling this as "Generative Fill" and the ads are all around...

Simply select the area with any selection tool (e.g. rect or circle select tool ). Then,

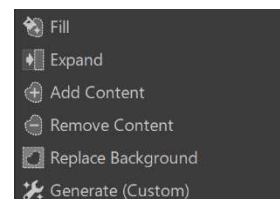
⁴ Embeddings (Textual Inversion) does not change the model itself; it just adds additional "keywords" and associate it to the image features in model (thus its name)

the Generate button will turn into  Fill which only the selected area will be generated.

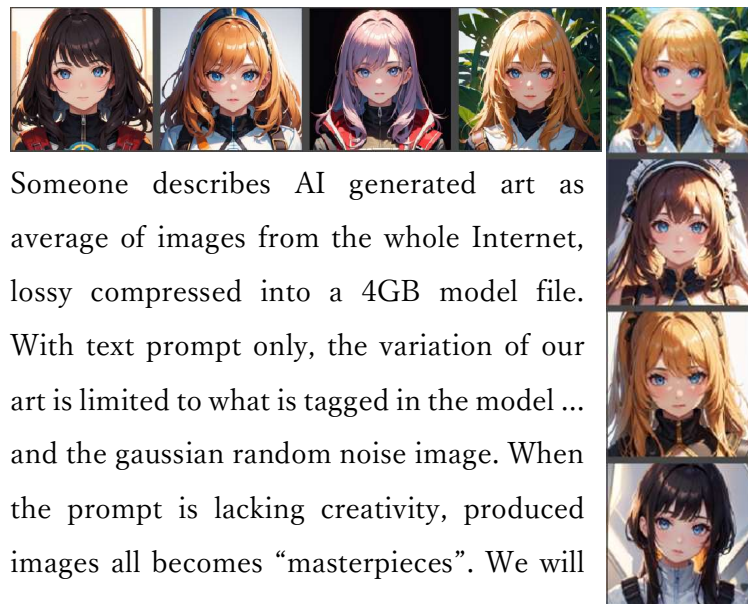


The **context area** (i.e. pixels around the selected area) is also sent to the model to generate image that fit seamlessly, as demonstrated by ⑥. Context can be changed with the "Generate (Custom)" mode in the dropdown menu of the Generate button.

The dropdown menu also contains other features, such as "Expand" for **Outpainting**. Others menu item are self-descriptive.



Masterpiece face (マスピ顔)



Someone describes AI generated art as average of images from the whole Internet, lossy compressed into a 4GB model file. With text prompt only, the variation of our art is limited to what is tagged in the model ... and the gaussian random noise image. When the prompt is lacking creativity, produced images all becomes "masterpieces". We will try more controlling methods next time, which hopefully make us better AI "artists"?

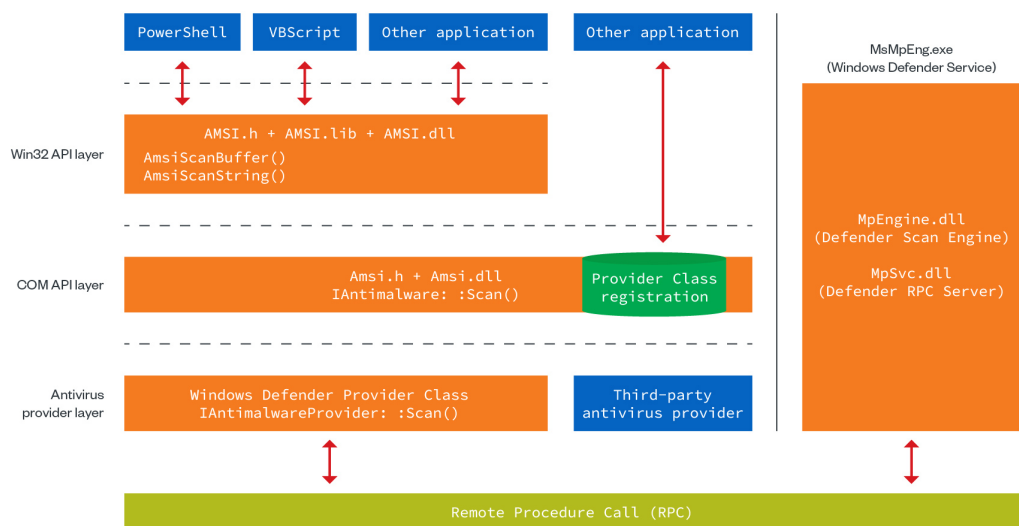
⁵ Copy the files to ComfyUI "/models/embeddings", then add the following negative prompt to the Krita setting, e.g.: "(embedding:EasyNegativeV2:0.5)"



In this chapter, I will talk about a very common techniques that Microsoft used for detecting malicious software, **Antimalware Scan Interface (AMSI)** and **Event Tracing for Windows (ETW)**. and how the attackers techniques to bypass those detecting.

Antimalware Scan Interface (AMSI)

The Windows Antimalware Scan Interface (AMSI) is a versatile interface standard that allows security applications and services to integrate with any antimalware product that's present on a machine. AMSI provides enhanced malware protection for your end-users and their data, applications, and workloads. For example, it will detect the malware or malicious files by searching for any common strings or signature hex used such as "AMSI", "ANTIVIRUS", or "X50!P%#@AP[4\PZX54(P^)7CC)7}\$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!\$H+H*".



©2022 TREND MICRO

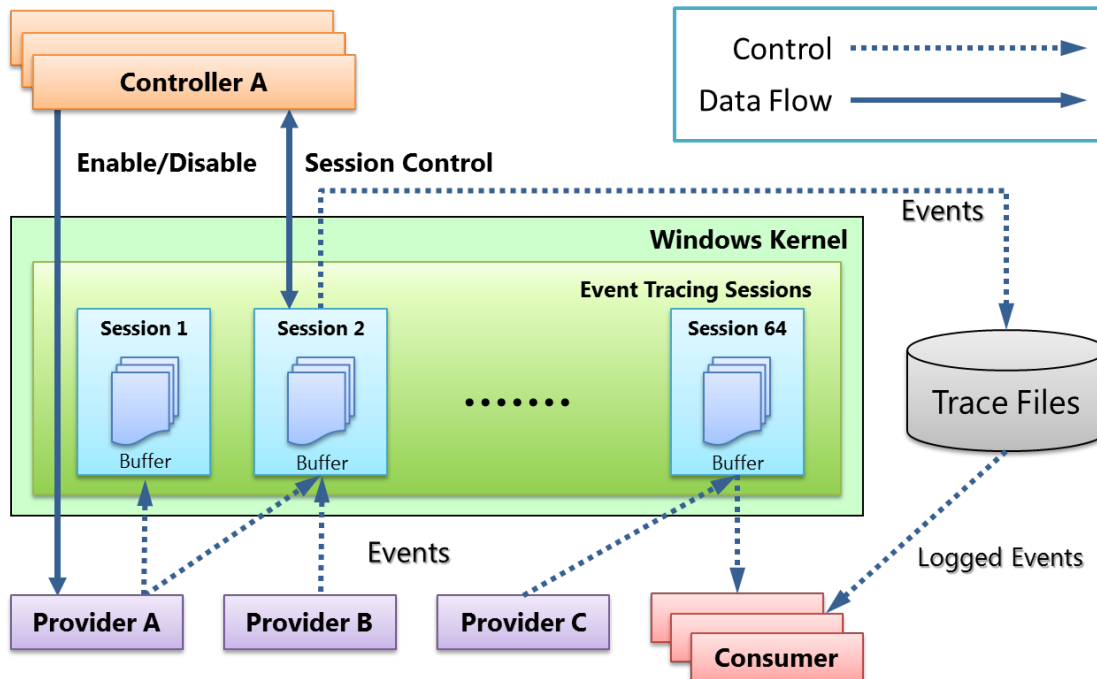
Windows AMSI architecture |

https://www.trendmicro.com/en_us/research/22/1/detecting-windows-amsi-bypass-techniques.html

Event Tracing for Windows (ETW)

Event Tracing for Windows (ETW) is a mechanism commonly used for debugging and security logging. Logging is leveraged by user-mode and kernel-mode applications. ETW is implemented in the Windows operating system and provides developers a fast, reliable, and versatile set of event tracing features. The security software can register the ETW provider to trace the function behavior of a running process.

ETW Architecture



ETW Architecture | <https://ithelp.ithome.com.tw/articles/10279093>

Bypass AMSI and ETW

Both AMSI and ETW functions are the code block inside the dll file

- AMSI: `AmsiScanBuffer()` in `amsi.dll`
- ETW: `EtwEventWrite` in `System.Management.Automation.dll`

Therefore, we can patch the code block of those dll to bypass them.

Let's say a very rough and simple patching solution:

1. We can use `GetProcAddress()` to retrieve the dll memory address.
2. Then use `VirtualProtect()` to modify the memory region to be read, write, and executable.
3. Finally, patch the targeted function and make it unable to be functional as intended.

However, the above method is very easy to be detected, so we can reference the techniques on <https://github.com/S3cur3Th1sSh1t/Amsi-Bypass-Powershell>

Those are the various techniques that the attackers commonly use for bypass AMSI and ETW such as setting Hardware break point or using CLR Hooking.



Introduction to Coppersmith's Method (simply)

Eason

In my previous article "How LLL works (simply)", we know about how to find solution from lattice matrix by applying LLL. This time, we would learn about how to solve modular polynomial equations with the Coppersmith's method.

There are different types of polynomials. In this article, we look at the easy case of univariate monic polynomials. Let N be an integer and $f(x) = x^d + \sum_{i=0}^{d-1} a_i x^i$ be a monic polynomial of degree d .

Build a matrix

Normally, if we wanna solve a polynomial mod N , we need to factor N . Most of the time, N factors into large primes and is thus hard to factor. Coppersmith's method helps us to find all integer solution x_0 to the equation $f(x_0) = 0 \pmod{N}$.

The main idea of Coppersmith's method is to **construct a polynomial $h(x)$ over the integers with larger coefficients such that $h(x_0) = 0$ holds over the integers, not just mod N** . To understand it, consider the lattice generated by the rows of B :

$$B = \begin{bmatrix} N & & & & & \\ & NX & & & & \\ & & NX^2 & & & \\ & & & \ddots & & \\ & & & & NX^{d-1} & \\ a_0 & a_1 X & a_2 X^2 & \dots & a_{d-1} X^{d-1} & X^d \end{bmatrix}$$

Each row in the matrix corresponds to the coefficients of a polynomial. For example, the first row $(N, 0, \dots, 0)$ corresponds to the polynomial $h_0(X) = N$, the second row corresponds to $h_1(X) = NX$, etc. The **last row** corresponds to the polynomial $f(X) = a_0 + a_1 X + \dots + a_{d-1} X^{d-1} + X^d$. The key is to notice that each row is 0 modulo N when evaluated at x_0 - for example, $h_0(x_0) = N \equiv 0 \pmod{N}$ and $f(x_0) \equiv 0 \pmod{N}$ by definition.

Since this matrix is lower-triangular, the determinant of B is the product of diagonals:

$$\det B = N^d X^{\frac{d(d+1)}{2}}$$

and we want to find the upper bound of X and vector b_1 :

$$\|b_1\| \leq 2^{\frac{n-1}{4}} (\det L)^{\frac{1}{n}} = 2^{\frac{d}{4}} N^{\frac{d}{d+1}} X^{\frac{d}{2}}$$

In order to satisfy the Howgrave-Graham bounds, $\|b_1\| < \frac{N}{\sqrt{d+1}}$ is needed. Simplifying the algebra, we obtain an upper bound for X :

$$X < C(d) N^{\frac{2}{d(d+1)}} =: \tilde{X}$$

which we can treat it as $\tilde{X} \approx N^{1/d^2}$. In other words, if the original root x_0 of f satisfies $|x_0| \leq \tilde{X}$, then we can solve the equation $f(X) \equiv 0 \pmod{N}$ by running LLL on B and solving the equation over the integers.

Implementation

Let's say we are given $N = 667 (= 23 \cdot 29)$ with unknown factorization, and $f(x) = x^2 + 6x + 352 \pmod{N}$. Suppose that we know the solution is less than $\tilde{X} = 20$. We construct

the lattice B like this:

$$B = \begin{bmatrix} N & & & & \\ & N\tilde{X} & & & \\ & & \ddots & & \\ & & & N\tilde{X}^{d-1} & \\ a_0 & a_1\tilde{X} & \cdots & a_{d-1}\tilde{X}^{d-1} & \tilde{X}^d \end{bmatrix} = \begin{bmatrix} 667 & 0 & 0 \\ 0 & 20 \cdot 667 & 0 \\ 352 & 6 \cdot 20 & 20^2 \end{bmatrix}$$

After applying LLL on this matrix, we will get the reduced basis B' :

$$B' = \begin{bmatrix} -315 & 120 & 400 \\ 352 & 120 & 400 \\ 167 & 12260 & -3600 \end{bmatrix}$$

We can get the first row and interpret it as the coefficients of the polynomial $h(\tilde{X}x)$ (since the matrix is scaled by \tilde{X}), then rescale it back to $h(x)$ like this:

$$h(20x) = 400x^2 + 120x - 315 \Rightarrow h(x) = \frac{400}{20^2}x^2 + \frac{120}{20}x - 315 = x^2 + 6x - 315$$

Finally, we can solve for the roots of $h(x)$ over the integers with the quadratic formula or Newton's method and find that the solution $x_0 = 15$. Note that we solve the equation over the integers, not over modulo N , and **without knowing factorisation of N** .

If we try different values of X , you can find that if X is too small, there is no solution. If X is larger than one of the factors of N , the first row will be moved to the second row:

$$X = 7, B' = \begin{bmatrix} 37 & 84 & 98 \\ -315 & 42 & 49 \\ 131 & 2695 & -2303 \end{bmatrix} \Rightarrow x_0 = -3 \pm \frac{1}{2}i\sqrt{38} \text{ (not integers)}$$

$$X = 24, B' = \begin{bmatrix} 667 & 0 & 0 \\ -315 & 144 & 576 \\ 204 & 15000 & -4032 \end{bmatrix} \Rightarrow \text{First row gives } h(20x) = 667 \neq 0.$$

Improvements

The upper bound $X \approx N^{1/d^2}$ are quite tight, so Coppersmith came up with 2 improvements on the matrix B to increase the bound X :

1. Using x -shifts (of function f) $xf(x), x^2f(x), \dots, x^kf(x)$.
2. Increase the power of N , as for any $0 \leq k \leq h$:

$$F(x) \equiv 0 \pmod{N} \iff F^k(x) = CN^k \iff N^{h-k}F^k(x) \equiv 0 \pmod{N^h}$$

After the improvements, the upper bound of X is improved to about $N^{1/d} \gg N^{1/d^2}$.

...Can we NOT build a matrix every time?

In [Sage](#), there is a function called `small_roots()`, it uses Coppersmith's method which you can just build a polynomial then call the function, but it can only solve univariate polynomial. Therefore, you can use some tools and codes for multivariable polynomials, such as [defund/coppersmith](#) and [kionactf/coppersmith](#) on GitHub. They are useful and effective for general modular polynomial solving problems. And it is also a good example to learn coding in Sage. Still, knowing how to build a matrix and how linear algebra works first is always the best way to flexibly deal with different kind of lattice-based problems.



Analysis of Web Applications From a Noob's Perspective

During the first few days of this year, while I was working on the front-end of one of my projects, when suddenly, I noticed that the rendered page was malformed for no reason at all. As a CTF player ~~with just 1 year of experience~~, I had flashbacks from doing XSS challenges and immediately identified it as a potential bug. After thoroughly checking my code, I determined that it had to be the front-end framework's problem, which is how I reported my first security vulnerability.

A few more bugs and CVEs later, I found out that code used in production was actually way more insecure than I thought it was, such that even a CTF newbie can discover 3 XSS bugs in a well-known JavaScript front end framework. This motivated me to put my CTF skills into practice, and what better way to challenge myself, than to go for one of the hardest targets, "the big end boss of XSS" according to popular youtuber Live Overflow, **DOMPurify**.

Of course, a secondary school student with one year of CTF experience can't simply find a bypass in DOMPurify, it's basically the most well-maintained defence mechanism against XSS! That's also what I thought, but since I had the Chinese New Year holidays ahead of me, I decided to give it a try, treating it as a learning opportunity.

During the "challenge" I gave myself, I found a few resources particularly useful, these include "Exploring the DOMPurify library: Bypasses and Fixes By Kevin Mizu", the Dom Explorer from YesWeHack, and a mXSS cheat sheet from Sonar Research.

If you've ever heard of the resources listed above, you will know that they all are in some way or another related to mutation XSS. Why mutation XSS you may ask? Because I looked at the recent DOMPurify bypasses and spotted a common theme: all of them are mutation XSS bypasses paired with comments.

To illustrate, here is the DOMPurify 3.0.8 bypasses found by Kevin Mizu:

```
<svg><annotation-xml><foreignobject><style><!--</style><p  
id="--><img src='x' onerror='alert(1)'">
```

The simple explanation is that due to namespace confusion with the `<svg>` and `<foreignobject>` tags, the first parsing treats `<!--` as text, while the second parsing will treat the same expression as a comment. This effectively means that DOMPurify will fail to recognise the expression as a comment, but the browser will, meaning there is a mismatch in interpretation, and that allows for exploits like XSS to occur. (The browser treats `</style><p id="--` as a comment, while DOMPurify does not recognise it as a comment, due to the parsing order.

What I have found is that since these types of mXSS are quite hard to patch, cure53(DOMPurify's maintainers) decided to straight up prohibit ending comment tags `-->` from appearing in attributes. However, the team did not invalidate attributes with the `>` character. Why would it need to be removed? Because HTML comments can actually appear in multiple forms, and while the most common form you see looks like `<!-- ...-->`,

the expression `<! ... >` is also surprisingly a valid comment, thanks to the incorrectly opened comment exception documented in the WHATWG HTML specification.

Does this mean that if you replace `<!--` with `<` and `-->` with `>` you can bypass DOMPurify? Well, not exactly. This is because for the mXSS payloads to work, after the comment start tag, you also have to first end the style namespace with `</style>`. Notice anything? It contains the `>` character! This means that the comment will end here instead of in the attribute of `<p id=">...>`, which is not ideal, since it makes the whole payload useless.

What we need to do is somehow remove `</style>` when rendering... After taking a deep, deep look at the code, I realised that there is a function right at the end which removes any expressions in the form of `${...}` which makes the code safe for template engines. However, this action is quite dangerous as it may conflict with the sanitized material. Taking advantage of this, I used this feature to remove the annoying `</style>`, and successfully bypassed DOMPurify version 3.2.3.

After 3 days of non-stop code auditing, "faking" bypasses, and questioning my sanity, I finally did it, I bypassed DOMPurify. Throughout the journey I learned way more than I initially expected, including mXSS techniques and how HTML tokenization worked, so the time spent was really worth it. Words cannot describe how thankful I am to the entire Black Bauhinia team, as they were the ones who got me interested in CTFs and helped me massively along the way.

1. DOMPurify

`<style><!--</style><p id="-->...`

2. Browser

`<style><!--</style><p id="(->...)`
XSS!
Patched
:(

Templates removed!

`<style><!-- ${</style> } <p id=">...)`
XSS✓

(Please note that the actual PoC uses a different starting payload from the one listed here, as I tried to simplify it for explanation purposes.)

The full PoC and rest of the details are at <https://ensy.zip/posts/dompurify-323-bypass/>.



手把手解 Heap pwn: Tcache Poisoning

gldanoob

Pwning 嘅 CTF 入邊算係門檻比較高嘅類別，而身邊都好少人特登去鑽研 pwn。所以今次我會用 vsCTF 叫 [Domain Expansion](#) 嘅呢題做示範，教吓近排 heap pwn 題比較興嘅 tcache poisoning，希望引起多啲人對 pwning 嘅興趣。以下會假設大家識少少 stack pwn。

領域... 展開

當我哋用 IDA 爆咗題目個 binary，會發現呢個又係 note app，裏邊嘅功能係用緊 malloc 同 free 等去實現。

但係今次寫呢個 binary 嘅人有咁慧居啦，識得記低每個 note 嘅長度，亦都有嘅 free 咗個 note 之后劃埋個 pointer，看似就有 heap overflow 同 use after free 嘅漏洞。唯一嘅“bug”係如果你揀 260 嘅話，某個 note 就會擴大至任何長度，而每次 run 都淨係可以展開一次。

```
Enter your choice: 260
DOMAIN EXPANSION
Enter an index: 0
Expanded size: 1024
```

咁今次我哋可以點利用五条悟幫我哋開個 shell 呢？喺度我會補充下 heap allocator 嘅運作先。假設我用 malloc 整咗兩個 heap chunks A 同 B，再依次序 free 咗 B 然後 A，嗰啲 chunks 唔會即刻消失，而係畀 allocator 以一個 linked list（呢個情況係 tcache bin）嘅形式記住：

```
pwndbg> bin
tcachebins
A fd B
0x30 [ 2]: 0x55555555b2a0 -> 0x55555555b2d0 -> 0
```

（向左嗰個箭咀代表最尾嘅 pointer 為 null。）當我哋再 malloc 嘅時候，如果呢個 bin 有啱大小嘅 free chunk，個 allocator 就會直接擺呢個 list 頭嘅 chunk。呢個係 glibc 其中一個 optimization。用 Pwndbg 嘅 vis 指令（全名係 vis_heap_chunks），可以睇到每個 tcache chunk 係點嘅樣：

```
0x0000000000000000 fd 0x0000000000000031 } A
0x0000000000000078b 0x04d1627f5ab411b2 }
0x0000000000000000 0x0000000000000000 }
0x0000000000000000 0x0000000000000031 } B
0x0000000000000055b 0x04d1627f5ab411b2 }
0x0000000000000000 0x0000000000000000 }
0x0000000000000000 0x00000000000000d11 }
```

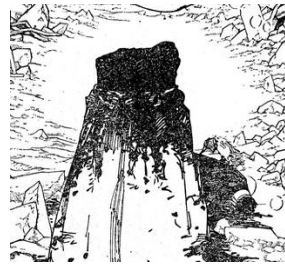
留意 chunk A 指去 chunk B 嘅 pointer（名為 fd），係直接儲存嘅 chunk A 入邊。意即如果我哋有能力去修改 memory 嘅呢個地方，咁我哋就可以 fake 個 allocator，話其實下個 chunk 係我哋任意揀嘅地方：

```
tcachebins
A fd Target
0x30 [ 2]: 0x55555555b2a0 -> 0x7ffff7e1a080 (*ABS*@got.plt)
fastbins
empty
B
```

呢個 challenge 嘅 idea 就係利用五条悟賜畀我哋嘅 buffer overflow 去修改 free chunks 嘅 FD pointer，令到 list 嘅第二個 entry 變咗我哋揀嘅地方，然後再叫 allocator malloc 多兩次，第二次 malloc 嗰個 chunk 就會變咗我哋可以控制嘅 target 啦。

而家既然我哋有 arbitrary write，咁係唔係直接 GOT overwrite，令個 program 當咗 system 係要查找嘅 libc function 就搞掂呢？有咁易。

RELRO: Full RELRO



LIBC 都有 GOT ？

喺 glibc 2.34 之前，有個叫 __free_hook（同 __malloc_hook）嘅 libc variable，用處係畀 user set 一個 function 作為 free() 嘅 handler。當我哋得到 arbitrary write 嘅能力之後，可以將 __free_hook 嘅位置改做 system()，令到 free(buffer) 嗰陣會 call system(buffer)，幫我哋開個 shell 出嚟。但係今次我哋嘅 libc version 係 2.35 嗰，除咗 __free_hook 仲有咩嘢可以畀我哋 overwrite 呢？

其實 GOT (Global Offset Table) 呢樣嘢係 executable 同 shared libraries 入邊都有嘅。當我哋 run 個 binary，個 libc load 入嚟 memory 嗰時，會根據裏面嘅另一個 GOT 嚟 look up 自己嘅 functions。Glibc version < 2.39 嘅話，個 libc ELF（唔係題目嗰個 ELF）

只係會有 Partial RELRO，所以我哋可以用類似 GOT Hijacking 嘅呢招。

一個最常畀人 exploit 嘅 libc GOT entry 就係 `__strlen_avx2`（即係 libc 內部 `strlen` 嘅名稱），而喺 Pwndbg 度打 `got -p libc` 就可以見到佢嘅位置：

```
[0x78f9d781a088] _dl_find_dso_for_object@GLIBC_PRIVATE → 0x78f9d781a090
[0x78f9d781a090] *ABS**+0xa88d0 → 0x78f9d779f040 (__strncpy_avx2)
[0x78f9d781a098] *ABS**+0xa86a0 → 0x78f9d779d7e0 (__strlen_avx2)
```

由於 `puts(*s)` 嘅 implementation 係會 call 到 `strlen(*s)`，我哋將 `__strlen_avx2` 改做 `system` 之後可以將 `"/bin/sh"` 寫入一個 chunk 入面，再 read note，即係 print 個 chunk 出嚟，咁 `puts("/bin/sh")` 嗰時就會 call `system("/bin/sh")`，即係開個 shell 出嚟啦。

但係若果我用 tcache poisoning 整個 fake chunk 直接擺喺 `__strlen_avx2` 度，嗰個 chunk 嘅 BK pointer 會變做 null，意即當 libc 用到 `__strlen_avx2` 嘅下一個 GOT Entry (+0x8) 嗰陣會 segfault。一個解決方法係將 chunk 擺喺三個 GOT entry 之前嘅位置 (-0x18)，而呢個位置嘅下一個 Entry (-0x10) 係較少機會畀 libc 去 dereference 嘅，咁就唔需要擔心嘅 exploit run 完之前 segfault。

少少嘅 details

細心嘅你可能留意到第一版幅圖嘅 fd pointer 根本唔係指向 chunk B。原因係 glibc 一個名為 Safe-Linking 嘅保護機制。設 P 為原本嘅 pointer， L 為 pointer 所在嘅 address (chunk A 嘅位置，即某個 heap address)，咁 $P' = P \oplus (L \gg 12)$ 就會係 encrypt 咗嘅新 pointer，而 P' 先至係擺喺 chunk A 入邊嘅 value。本來呢個機制會令好多有 ASLR bypass 嘅 heap exploit 唔再運作到，因為 attacker 有一個 heap address 嚟 decrypt 個 pointer。不過喺一個 special case 入邊，如果我哋 leak 到呢個 pointer 出嚟嘅話，可以正正利用

佢去 bypass heap 嘅 ASLR。當 tcache bin 得一個 entry，該 fd 會指向 null：

```
tcachebins
0x30 [ 1]: 0x55555555b2a0 ← 0
```

設 P_0 係被 encrypt 咗嘅 null pointer，即 $P_0 = 0 \oplus (L \gg 12) = L \gg 12$ 。咁 P_0 就係 encrypt 或者 decrypt 任何 fd pointer 嘅 key 啦。（呢度假設嘅 heap address 只係最尾嘅 12 個 bits 唔同）要得到呢個 key，我哋只需要 read 個 free chunk 出嚟。

咁既然我哋要好準確去 overwrite libc GOT，libc 嘅 ASLR 又點樣破解呢？我哋有另一個叫 unsorted bin 嘅 free list，佢嘅結構為 circular doubly linked list，而頭尾都會係 `main_arena` (libc 裡面其中一個 struct)。

相信大家知道要做嘅乜 — 整個 chunk 出嚟放入 unsorted bin，再 leak 佢個 fd pointer，即 `main_arena` 呢個 libc address。但係個 chunk 要入到去 unsorted bin 有兩個條件：一係個 chunk 要夠大 (擺唔入 tcache bin)，二係佢唔可以係 heap 最頂嗰個 chunk。

點贏？

[Solution](#) 可以有無數個，以下係一個 guideline:

1. 整一個 32 bytes 同一個大 (> 1032 bytes) 嘅 notes，再整多個 note 以符 consolidation，delete 咗頭兩個 notes
2. Domain expansion 最先嗰個 note，分別用兩個 notes 嚟 leak heap 同 libc address
3. 將 tcachebin 清空，allocate 兩個 32 bytes 嘅 notes，然後倒序放入 tcachebin
4. 修改 tcachebin 最尾嗰個 chunk，令佢嘅 fd 指向 LIBC GOT 入面 `_dl_find_dso_for_object` 嘅位置 (切記要先自己 encrypt 個 pointer)
5. 清空 unsorted bin，然後 allocate 兩次，以控制 libc GOT，將 `__strlen_avx2` 取代為 `system`
6. 將 `"/bin/sh"` 寫入任何一個 note，再 read 佢，享受爆出嚟嘅 shell



A Primer on Searching for High Rank Elliptic Curves

grhkm

On August 29th, Noam Elkies and Zev Klagsbrun announced [4] the discovery of an elliptic curve of rank (at least) 29* over the rational numbers. It has Weierstrass equation:

$$E: y^2 + xy = x^3 - 0x10ce554d1283aeafe731dd2285b1cbb543c3ed4853127f6894e9x \\ + 0x1b206f3de15feb61e310e519ad8a7a827cca68b96d2601d74570c8b6c488211c097e59cfac359$$

This increments the previous record of a rank 28 curve, also found by the two. In this article I hope to explain this result means, why it is significant, and the general idea behind the search.

1 Rank of E/\mathbb{Q}

To begin, we must first understand what the rank of an elliptic curve is. Consider the set of rational points $(x, y) \in \mathbb{Q}^2$ satisfying $E: y^2 = x^3 + x$. It is easy to find one such point: $(0, 0)$, but other than that it is unclear whether there are any other points. Indeed, it can be proven that this curve has finitely many points: $E(\mathbb{Q})$ is finite. By contrast, consider the curve $y^2 = x^3 - 16x + 16$. It is not difficult to find many points on this curve: $(\pm 4, \pm 4)$, $(\pm 4, \mp 4)$, $(0, \pm 4)$, $(0, -1)$, etc. One may again wonder if there are infinitely many such rational points. In this case, it can be proven that there *are* infinitely many points. They can be generated via the following SageMath code:

```
sage: E = EllipticCurve([-16, 16])
....: G = E.gen(0)
....: print(*[(G * k).xy() for k in (1..8)])
(0, -4) (4, -4) (-4, 4) (8, 20) (1, 1) (24, -116) (-20/9, -172/27) (84/25, 52/125)
```

To differentiate between these curves, we can use the fundamental theorem of finitely generated abelian group and Mordell-Weil theorem (1922), which are stated below:

Theorem 1.1: Fundamental Theorem of Finitely Generated Abelian Group

Let G be a f.g. abelian group. Then, there exists $r \in \mathbb{N}$ such that $G \cong \mathbb{Z}^r \oplus G_{tors}$, where G_{tors} are the elements of G with finite order and is finite. The value r is called the *rank* of G , denoted $rk(G)$.

Theorem 1.2: Mordell-Weil Theorem

Let E/\mathbb{Q} be an elliptic curve. Then $E(\mathbb{Q})$, also called the **Mordell-Weil group**, is finitely generated.

These two theorems combined tell us that $E(\mathbb{Q})$ as an abelian group has structure $\mathbb{Z}^r \oplus T$ for some finite group T . Just to be clear, the r here indicates the “dimension” of (the torsion-free part of) $E(\mathbb{Q})$. For example, when $r = 0$, then $E(\mathbb{Q}) \cong T$ is a finite group, while when $r > 0$, the set of rational points on E is infinite. This is one of our goals from the start, but we can extract more information from r . More specifically, when $r > 0$, there exists a basis $\{G_1, G_2, \dots, G_r\}$, each of infinite order, such that every point in $P \in E(\mathbb{Q})$ can be uniquely written as $P = P' + \sum_{i=1}^r [c_i]G_i$ for some integers $c_i \in \mathbb{Z}$ and a point of finite order P' .

Moreover, Mazur’s theorem (1978) shows that T only has a finite number of choices (in particular, $|T| \leq 16$), and is efficiently computable. Hence to understand the structure of an elliptic curve over the rationals, we want to determine the rank r . More generally, we would like to understand the distribution of r and how it relates to other invariants.

Now we understand Elkies and Klagsbrun’s latest result: they discovered a curve with rank $r \geq 29^*$, improving on the previous record of $r \geq 28!^*$. They explicitly found 29 linearly independent points on the curve given at the start.

*All rank inequalities $r \geq r_0$ in this article can be replaced by $r = r_0$, under the assumption of GRH when $r_0 \geq 2$.

2 Specialising fibrations

Next, I want to attempt to explain the high level ideas behind the method of discovering such a curve, as clearly it is not obtained by testing coefficients one by one. Elkies and Klagsbrun’s strategy consists of two parts: (1) finding an *elliptic fibration* $\mathcal{E}/\mathbb{Q}(t)$ on a $K3$ surface with large Mordell–Weil rank r , and (2) sieving for good values of t for which the specialisation E_t has large rank.

First off, don’t be scared of the scary terms like *elliptic fibration* and *elliptic surface*! A simple mental model for them is just a normal elliptic curve $y^2 = x^3 + ax + b$, but with coefficients a, b in the function field $\mathbb{Q}(t)$, or for our purpose even $a, b \in \mathbb{Q}[t]$ holds[†]. Recall from my isogeny article that the notation E/K just means an elliptic curve E defined over K , as a subset of K^2 .

For the remainder of this article, we will use the elliptic surface $\mathcal{E}_0 : y^2 = x^3 + (t^{12} - 26t^6 - 343)$. Notice that it is an equation of 3 variables, which suggests that this is a surface. However, the correct way to view this is that t is a parametrising variable, and \mathcal{E}_0 is a parametrised family of elliptic curves[‡].

A key property of \mathcal{E}_0 is that it has Mordell–Weil rank at least 4. Indeed, it is proven in [5, Theorem 5.8] that the following four points on \mathcal{E}_0 are linearly independent as points over $\mathbb{Q}(t)$:

$$(8, t^6 - 13), \left(\frac{-t^6 + 49}{t^2}, \frac{-11t^6 + 343}{t^3} \right), (t^6 + 7, t^9 + 11t^3), \frac{1}{64} \left(36t^{10} + 344t^4 + \frac{196}{t^2}, 27t^{15} + 387t^9 + 1145t^3 - \frac{343}{t^3} \right)$$

How does this help us find high rank elliptic curves defined over \mathbb{Q} ? The hope now is that by **specialising** t to a rational number t_0 , i.e. by evaluating the equation and points at $t = t_0$, we can obtain an elliptic curve E_{t_0}/\mathbb{Q} which automatically has four rational points on it, meaning that if the other points behaves “randomly” and we are able to find more linearly independent rational points, then E_t will have a higher rank than usual. Of course, we are relying on the assumption that the four points above remain linearly independent over \mathbb{Q} once evaluated at $t = t_0$. Fortunately, Silverman proved that this is usually true:

Theorem 2.1: Silverman’s Specialisation Theorem [7, Theorem 11.4]

Let \mathcal{E} be an elliptic surface defined over the function field $\mathbb{Q}(t)$. Then the specialisation map

$$\begin{aligned} \sigma_{t_0} : E(\mathbb{Q}(t)) &\rightarrow E_{t_0} \\ (x, y) &\mapsto (x(t_0), y(t_0)) \end{aligned}$$

is well-defined and injective for all but finitely many points $t_0 \in \mathbb{Q}$.

Let’s do an example with our elliptic surface \mathcal{E}_0 . By direct computation, we see that for $t_0 = 3, 9, 20$, we get $E_3 : y^2 = x^3 + 512144$, $E_9 : y^2 = x^3 + 282415718672$ and $E_{20} : y^2 = x^3 + 4095998335999657$ respectively, and for the first case, the 4 points provided specialise to $(8, 716)$, $(-680/9, -7676/27)$, $(736, 19980)$ and $(302857/9, 166669613/27)$. In Section 4, we shall verify that in all cases, the specialised points are indeed linearly independent, so right off the start, we already have three curves with rank at least 4. However, it is even better than that, as we are able to find more linearly independent points. In fact, the naive method of enumerating **integers** x and search points on $E(\mathbb{Z})$ yields linearly independent points. In the end, we obtain that $\text{rk}(E_3) \geq 4$, $\text{rk}(E_9) \geq 7$ and $\text{rk}(E_{20}) \geq 8$, and we will prove equality in Section 5.

```
sage: points = [...]
....: for x in range(2^22):
....:     if E.is_x_coord(x):
....:         P = E.lift_x(x)
....:         if check(E, points + [P]):
....:             points.append(P)
```

[†]But $\mathbb{Q}[t]$ is a ring, so we can’t define a variety over it without scheme theory.

[‡]With some singular cubic curves here and there, but I digress.

Going back to Elkies and Klagsbrun’s result, they did precisely our method, but on a much larger scale, and by being much smarter. As described in [2, p. 24] and in detail in [3, p. 10], to find a suitable elliptic fibration, they enumerated the 167889 elliptic fibrations (that’s 167889 parametrised elliptic curve family!) on the K_3 surface with Néron–Severi group of rank 20 and discriminant -163 , words we don’t need to understand. Among those, they found a rank-17 fibration. However, they did not compute an explicit model for the fibration, so we are unable to play around with it ourselves. The only step that remains now is to find good specialisation values t . For my elliptic fibration, I was able to find the rank-8 specialisation by luck and (initially) a 2-hour brute force, but that doesn’t scale well at all. Instead, a smarter idea based on the BSD conjecture is used.

3 BSD conjecture and high rank elliptic curves

Being one of the millenium problems, the Birch and Swinnerton-Dyer (BSD) conjecture is one of the most famous open problems in number theory, providing deep insights into the structure of elliptic curves. Specifically, the conjecture links the rank r of an elliptic curve E/K over a number field K to the behavior of arithmetic invariants associated with primes p . One formulation of the conjecture is given below:

Conjecture 3.1: Birch & Swinnerton-Dyer Conjecture [1]

Let E/K be an elliptic curve of rank r . Then, $\prod_{p \leq B} \frac{N_p}{p} \approx C(\log B)^r$, where $N_p := p + 1 - |E(\mathbb{F}_p)|$ ^a and C is a constant dependent on E .

^aThis definition is true at primes with good reduction, and I do not want to talk about the other case.

To break this conjecture down even further, we can take logarithms to see that $\sum_{p \leq B} \log(N_p/p) \approx \log C + r \log \log B$. This gives us a somewhat heuristic method to check whether a curve has high rank.

First, fix a bound B , say 2^{16} . Given an elliptic curve E , we can reduce it at each (good) prime p , compute $|E(\mathbb{F}_p)|$ and hence N_p , and evaluate the sum $\sum_{p \leq B} \log(N_p/p)$. If it is large compared to other similar curves[§], then we know that it has high rank! For example, the two curves below have similar size, yet one has rank 11 and one has rank 3. The `E.pari_curve().ellrank()[2]` returns (unconditionally) upper and lower bounds for the curve, while `E.Np(p)` computes the quantity N_p defined above.

```
sage: E = EllipticCurve(QQ, [0, -54141938135, 0, 771820685537294757888, 0])
sage: E.pari_curve().ellrank()[2]
[11, 11]
sage: sum(n(log(E.Np(p) / p)) for p in prime_range(2, 2**18))
15.7842585917576
sage: E = EllipticCurve(QQ, [0, -54141938088, 0, 771820685537294757888, 0])
sage: E.pari_curve().ellrank()[2]
[3, 3]
sage: sum(n(log(E.Np(p) / p)) for p in prime_range(2, 2**18))
1.67922917143010
```

To further speedup the computation, instead of testing the specialisations E_t one by one and computing each N_p slowly, Nagao realised that if the coefficients of the defining equation of $\mathcal{E}/\mathbb{Q}(t)$ are all polynomials in t , then $|E_t(\mathbb{F}_p)|$ only depends on $t \pmod{p}$. As a result, it suffices to precompute $\log(N_p/p)$ and store $\sum_{p \leq B} p \sim B \log B$ values, and computing the sum $\sum_{p \leq B} N_p/p$ takes $\mathcal{O}(B/\log B)$ time. There are further tricks such as **sieving** documented in [4], low-level optimisations and (ab)using integer arithmetic, but go ask @happypotato about that.

As a remark, there have been other types of sums (“score functions”) used in place of $S(B) = \sum_{p \leq B} N_p/p$ (which is due to Mestre). In [6], Nagao used $S(B) = \sum_{p \leq B} (-N_p + 2)/(p + 1 - N_p)$ and $S'(B) = \sum_{p \leq B} -N_p \log p/p$ to find a curve of rank at least 20.

[§]For “similar enough” curves, the constant C will be approximately the same

4 Verifying linear independence

I realised that this article is getting too long, so I will put this on my blog soon. Stay tuned :)

5 Upper bound on $\text{rk}(E)$

I realised that this article is getting too long, so I will put this on my blog soon. Stay tuned :)

6 Appendix

In Section 2, we went from a $K3$ surface (not shown) to an elliptic surface of rank 4, and finally to an elliptic curve of rank 8. However, there is a much easier way to find elliptic curves of high rank. As a motivating example, consider three arbitrary points $P_1 = (x_1, y_1), P_2 = (x_2, y_2), P_3 = (x_3, y_3) \in \mathbb{Q}^2$. By solving simultaneous equations $y_i^2 = x_i^3 + a_2x_i^2 + a_4x_i + a_6$ for coefficients a_2, a_4, a_6 , we automatically obtain an elliptic curve of at least rank 3 – precisely P_1, P_2, P_3 . To generalise this method, we can use the fact that nine points determine a cubic to compute a cubic through nine points, though one has to be careful about fixing the origin.

References

- [1] B.J. Birch and H. P. F. Swinnerton-Dyer. “Notes on elliptic curves. II.” In: *Journal für die reine und angewandte Mathematik* 1965.218 (1965), pp. 79–108. DOI: [doi:10.1515/crll.1965.218.79](https://doi.org/10.1515/crll.1965.218.79).
- [2] Noam Elkies. “ $K3$ surfaces and elliptic fibrations in number theory”. Banff Workshop 18w5190: Geometry and Physics of F-theory. 2018.
- [3] Noam Elkies. “Three lectures on elliptic surfaces and curves of high rank”. 2007.
- [4] Noam Elkies and Zev Klagsbrun. “ \mathbb{Z}^{29} in $E(\mathbb{Q})$ ”. 2024.
- [5] Matthijs Meijer. “High rank elliptic surfaces”. In: *Rijksuniversiteit Groningen* (1999).
- [6] Koh-ichi Nagao. “An example of elliptic curve over \mathbb{Q} with rank ≥ 20 ”. In: *Proc. Japan Acad. Ser. A Math. Sci* 69.8 (1993), pp. 291–293.
- [7] J.H. Silverman. *Advanced Topics in the Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics. Springer New York, 1994. DOI: <https://doi.org/10.1007/978-1-4612-0851-8>.



IDA Tips 1: Custom Structures

harrier

This is an article series dedicated to tips in reverse engineering in IDA. While IDA has many issues (mostly our issue of being poor for an IDA Pro license) compared to other decompilers, e.g. Ghidra and Binary Ninja, up-to-date IDA Free and its IDC support in recent years offsets its disadvantages and makes it still one of the best tools for reverse engineering.

Let's start with something basic: custom structures

When doing reverse, you will often encounter structured pointers in all the places, especially when dealing with binary static compiled with large libraries / binary compiled from non-C/C++ language:

```
1 void __fastcall sub_74E60(__int64 a1)
2 {
3     *(_DWORD *)(a1 + 16) = 0;
4     *(_DWORD *)(a1 + 8) = 0;
5 }
```

Here, **a1** is a pointer pointing to some structure. You would want to define a structure for **a1** to make it prettier and consistent in the decompiled view, as you won't want to see **a1+16** all the time. Instead, you would want to link **a1+16** which appears in various functions to some common context, e.g. here, **a1** is the bignum structure from OpenSSL.

To do this, you will write the struct in C style. Usually, you can just get this from the repo if the struct is from an open-source library, say OpenSSL or libcrypto.

```
typedef struct unsigned long long BN_ULONG;
struct bignum_st {
    BN_ULONG *d;
    int top;
    int dmax;
    int neg;
    int flags;
};
typedef struct bignum_st BIGNUM;
```

In IDA <=8.3, you have to write the struct in a header file, then **File > Load file > Parse C header file** (or just **Ctrl + F9**). But in IDA >=8.4, you can directly do this in the **Local Types** subview, which replaces the previous Structures + Enums + Local Types subview. Just (**Insert / Right Click Add Type**), and directly type the struct definition in the “C syntax”. It is the same as the header file import method, but you don't have to save a file.

```

Original
00000010 BN_POOL_ITEM *tail;
44 00000018 unsigned int used;
45 0000001C unsigned int size;
46 00000020 };
47
48 00000020 typedef struct bignum_pool BN_POOL; // XREF: bignum_ctx/r
49
50 00000018 typedef struct bignum_st BIGNUM; // XREF: bignum_pool_item/r
51
52 00000000 struct bignum_ctx_stack // sizeof=0x10
53 {
54 00000000 {
55 00000000 unsigned int *indexes;
56 00000008 unsigned int depth;
57 0000000C unsigned int size;
58 00000010 };
59
60 00000000 struct bignum_ctx // sizeof=0x48
61 {
62 00000000 {
63 00000000 BN_POOL pool;
64 00000020 BN_STACK stack;
65 00000030 unsigned int used;
66 00000034 int err_stack;
67 00000038 int too_many;
68 0000003C int flags;
69 00000040 OSSL_LIB_CTX *libctx;
70 00000048 };
71
72 00000048 typedef struct bignum_ctx BN_CTX;
73
74 00000010 typedef struct bignum_ctx_stack BN_STACK; // XREF: bignum_ctx/r
75
76 FFFFFFFF struct OSSL_LIB_CTX;
77
78
79
80

```

After importing the struct, you can then change the type of **a1** to **BIGNUM***. The function looks like this after importing the custom structs:

```


1 void __fastcall BN_zero(BIGNUM *a1)
2 {
3     a1->neg = 0;
4     a1->top = 0;
5 }

```

More importantly, the struct can be reused and is much more effective on larger functions. Just by changing the type definition, the functions become much more readable.

<pre> 1 __int64 __fastcall BN_cmp(__int64 *a1, __int64 *a2) 2 { 3 int v2; // eax 4 __int64 result; // rax 5 int v4; // edx 6 unsigned int v5; // r9d 7 __int64 v6; // rdx 8 __int64 v7; // r8 9 __int64 v8; // rdi 10 unsigned __int64 v9; // rsi 11 unsigned __int64 v10; // rcx 12 13 if (a1 && a2) 14 { 15 v2 = *((_DWORD *)a1 + 4); 16 if (v2 == *((_DWORD *)a2 + 4)) 17 { 18 v4 = *((_DWORD *)a1 + 2); 19 v5 = v2 == 0 ? -1 : 1; 20 result = v2 != 0 ? -1 : 1; 21 if (v4 <= *((_DWORD *)a2 + 2)) 22 { 23 if (v4 < *((_DWORD *)a2 + 2)) 24 { 25 return v5; 26 } 27 } 28 else 29 { 30 return v5; 31 } 32 } 33 } 34 return v5; 35 } </pre>	<pre> 1 __int64 __fastcall BN_cmp(BIGNUM *a1, BIGNUM *a2) 2 { 3 int neg; // eax 4 __int64 result; // rax 5 int top; // edx 6 unsigned int v5; // r9d 7 __int64 v6; // rdx 8 unsigned __int64 *d; // r8 9 unsigned __int64 *v8; // rdi 10 unsigned __int64 v9; // rsi 11 unsigned __int64 v10; // rcx 12 13 if (a1 && a2) 14 { 15 neg = a1->neg; 16 if (neg == a2->neg) 17 { 18 top = a1->top; 19 v5 = neg == 0 ? -1 : 1; 20 result = neg != 0 ? -1 : 1; 21 if (top <= a2->top) 22 { 23 if (top < a2->top) 24 { 25 return v5; 26 } 27 } 28 else 29 { 30 return v5; 31 } 32 } 33 } 34 return v5; 35 } </pre>
--	---

Of course, we can prepare header files for commonly used structures to avoid defining the same struct over and over again, reusing library signatures in *different* decompilations.

Maybe we will do it in ctfools! 

Next time we will be talking about FLIRT signatures. Stay tuned!



Technical Minecraft - Chunk Loading I

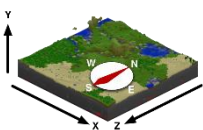
hoifanrd



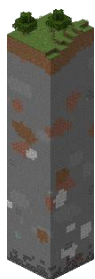
This series is going to be one of the longest series ever in Black Bauhinia Newsletter I think, where this series will be investigating and discussing the technical side of a popular sandbox game – Minecraft. Throughout the whole journey, we'll analyze the Minecraft source code thoroughly, such that we can perform different “exploits”. Such survival-constructible contraptions are very powerful in the game, which can lead to various game-changing behaviors. Currently, the goal of this series is to guide the readers to perform race conditions in Minecraft, thus allowing to get almost any illegal item and blocks in survival Minecraft. During the whole journey, we'll stick to Minecraft version 1.12 (still a very popular version!). However, there are lots of prerequisites and knowledge we need to understand before we can reach the goal. Therefore, let's get started!

Chunks

In order to manipulate with data stored in Minecraft, let's mess with the chunks first! In Minecraft, there are 3 **worlds**, namely the Overworld, the Nether, and the End. Each world consists of **blocks** and **entities**, where a 3D Cartesian coordinate system is used to map the blocks and entities to the world (I call it the **world coordinates**, to distinguish with the **chunk coordinates**, which will appear later). Notice that the axes are swapped in Minecraft, as shown in the figure at the right.



Entities are mapped using the world coordinates (i.e. float), where the Y-axis represents the floor level that the entity is standing on. Blocks are more interesting because they occupy a whole cube. For example, a block occupies a cube from vertex (10,63,10) to (11,64,11), thus its center coordinate is actually (10.5,63.5,10.5). It is rounded down to (10,63,10) and the game will use this integer **block coordinate** to represent instead.



A world has almost infinite length X, Z axes (from coordinates -30M to +30M), and Y axis from 0 to 255. Since it's too big to load and process everything in the world, only some of the blocks and entities will be loaded. Following the principle of spatial locality, similar to page frames in physical memory of a computer, Minecraft will also load a certain region around the block/entity when it is being accessed, and that region is called a **chunk**. A chunk is a cuboid that contains all the blocks and entities from the vertex (16m,0,16n) to (16m+16,255,16n+16) of the world coordinates, where m,n are integers. Therefore, a chunk has a length/width of 16 blocks, and a height of 256 blocks. Here, the coordinate (m,n) is then the **chunk coordinate** indexing a specific chunk. In Minecraft, many events will lead an access to a block/entity, causing the chunk that contains the

block/entity to be loaded. When a chunk is loaded, the game will then process (i.e. tick) all the blocks and entities in that chunk. (Not really the case with entities, and there are some hidden conditions..., to be explained in the next article probably).

Loading and Unloading with Chunks

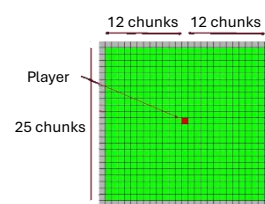
Here are the events causing chunks to load or unload:

(1) The Overworld has a world spawn, where chunks that are near the spawn are **spawn chunks** (determined by the code as shown). Spawn chunks are loaded at the start of the game, and will NEVER be unloaded.

```
public boolean isSpawnChunk(int x, int z) {
    BlockPos blockpos = this.getSpawnPoint();
    int i = x * 16 + 8 - blockpos.getX();
    int j = z * 16 + 8 - blockpos.getZ();
    return i >= -128 && i <= 128 && j >= -128 && j <= 128;
}
```

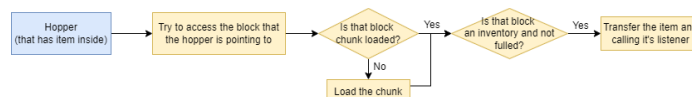
A code part determining whether the chunk is spawn chunk, given the chunk coordinate (m,n)

(2) Any movement event of a player that involves moving across chunk borders, or teleportation (e.g. nether portal/ender pearl teleportation), also causes chunks to load or unload. In game, the **render distance** d_r of a player is configured (e.g. 12 chunks). Then the square region of diameter $2d_r + 1$, centered at the player will be loaded. We call those chunks are in the render distance of the player. When a player moves across chunk borders or is being teleported, chunks that should be in the render distance with respect to the new position of the player, will be loaded; while chunks that aren't in any players' render distance, will be scheduled to unload.

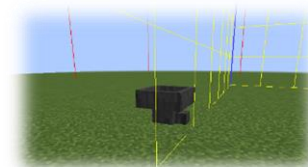


Example of 12 chunks render distance. Green is loaded and grey is unloaded.

(3) For some **tile entities** (i.e. blocks that stores state, such as hopper, chest and redstone), they will access their neighbor blocks and call their listener. The following is an example:



The above logic will be executed at every **game tick** (i.e. constantly), and not limited to hopper, powered redstone component also works. Therefore, if we put the hopper at the chunk borders (pointing to another chunk), the chunk next to the hopper will be loaded, as long as the chunk containing the hopper is being loaded.



Chunk borders shown, the right chunk will always be loaded, given the left chunk is loaded. Note: Hopper must have an item for this to work.

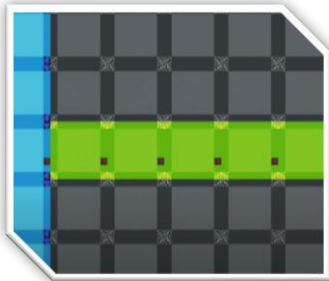
(4) Every 45 seconds, the game will perform an auto-save, thus schedule to unload all the chunks except the spawn chunks (1), and chunks around players (2).

Note that chunk loading is instantly loaded when required, while all chunk unloading is scheduled (i.e. being unloaded later).

Naïve Chunk Loading



We know, players love to make automatic farms in Minecraft. Thus the problem arises – the farm no longer works when the chunks are unloaded (e.g. the player leaves). As a result, keeping chunks loaded is crucial for a systemic game in Minecraft.



Blue chunks: Source chunks that are loaded.
The hopper in the blue chunk loads the 1st green chunk, then the hopper in the 1st green chunk loads the 2nd green chunk... and it continues.

Combined with what we have learnt, it's easy to think of a naïve way to keep loading chunks: Create a “hopper chain” from chunks that are loaded. However, this requires a stable “source” of loaded chunks. For example, spawn chunks can be used as a source for such hopper chains, but they are only available in the Overworld and it's annoying to build a chain if the target chunk to be loaded is far away from spawn.

Another option is to use chunks near the player as the source. For example setting up the chain source to your base, since you usually use more time to stay in your base when playing Minecraft.

Ticking in Minecraft

Assume a long chunk loading chain is set up from your base. Since player movement only unloads the chunks in the player's original render distance, even if you leave your base and the source chunks of the chain are unloaded, further chunks of the chain will not be unloaded!

However, don't forget about the auto unload every 45 seconds. So we are doomed!... Or is it? Let's see if there is a way to bypass the auto unload. To understand how unloading works, we need to know about what a game tick is, which has been mentioned before.

Given Minecraft is a single threaded game, events in the game are executed sequentially in a **(game) tick**, where a tick lasts for 0.05s, thus there are 20 ticks in 1 second. Some events in a tick are executed in order as shown, with a lot of details omitted.

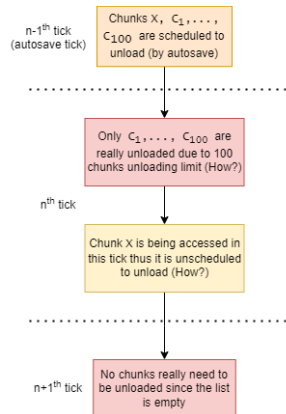
Also, function code marked in the red box (i.e. really unloading the scheduled unload chunks) is shown below:

```
// For each world
private final Set<Long> droppedChunksSet = Sets.<Long>newHashSet();
// ...
public boolean unloadQueuedChunks() {
    if (!this.droppedChunksSet.isEmpty()) {
        Iterator<Long> iterator = this.droppedChunksSet.iterator();
        for (int i = 0; i < 100 && iterator.hasNext(); iterator.remove()) {
            Long oLong = iterator.next();
            Chunk chunk = (Chunk)this.id2ChunkMap.get(oLong);

            if (chunk != null && chunk.unloaded) {
                // Perform unload
                ++i;
            }
        }
    }
}
```

Basic Idea

From the code, we may see that only at most 100 chunks will be really unloaded in each tick (to ensure performance probably, thus the 101st and the chunks following will be really unloaded in the next tick). Now, assume we have a chunk X , which we want to prevent its unloading from autosave, we may first prepare 100 auxiliary chunks C_1, \dots, C_{100} in the same world as X in, and perform the following as shown in the flowgraph.



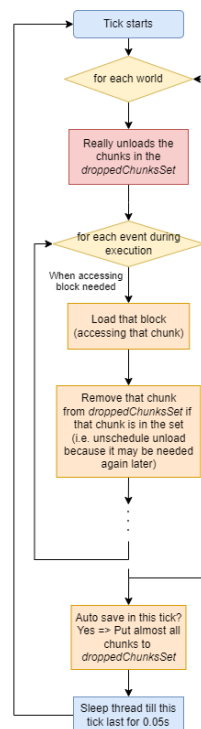
With this, we may get 3 questions:

1. How can we manipulate with the iterator of the *droppedChunksSet*, so that the chunks C_1, \dots, C_{100} must be unloaded first?

2. How to make chunk X always being accessed to unschedule its unload?

3. Also, C_1, \dots, C_{100} will be unloaded thus this no longer works in the next autosave tick. (i.e. we need to load C_1, \dots, C_{100} again before the next autosave tick).

Since there is not enough space left to discuss the above questions, let's answer them in the next article of *Technical Minecraft*. As a challenge, you may also try to solve those questions yourself first!





Sharing a Dynamic Analysis Technique: Function Interposition

Motivation

Once upon a time, there was a Linux application that needed to be reverse engineered. The app actively communicated with a server, pulling instructions to perform actions like updating software versions or changing configurations. I wanted to see the clear text HTTP traffic to better understand how the client and server communicated, but I wasn't very skilled at reverse engineering. Luckily, I found a simple method that even a dummy like me could use to inspect the encrypted HTTPS traffic.

TLDR

So to obtain the SSL master key from any program that uses openssl, all you have to do is to compile the code at <https://github.com/Lekensteyn/wireshark-notes/blob/master/src/sslkeylog.c> and load the compiled library via LD_PRELOAD. When the target application is executed, the loaded custom library will log the SSL master key to whatever path defined in SSLKEYLOGFILE. With the master key, you can decrypt the relevant captured TLS encrypted packets on wireshark.

However, what I found more interesting is the technique used, called Function Interposition. I was surprised by how versatile this method can be for dynamically analyzing any Linux application. If you are interested in writing your own code to fit your specific use case, keep reading.

Modus Operandi (How does it work)

The basic principle is simple. When the target binary is executed, a customized library is loaded that hooks into the OpenSSL library. This is achieved by defining the environment variable LD_PRELOAD, which the system uses to preload any specified shared libraries during program execution. This allows you to override or extend any functions loaded by the running program.

To understand how the custom library searches for the target function, we can look at the code in `sslkeylog.c`. The function `try_lookup_symbol` locates and returns the address of the specified function.

```
static inline void *try_lookup_symbol(const char *sym, int optional)
{
    void *func = dlsym(RTLD_NEXT, sym);
    // ...SNIP...
    return func;
}
```

In the function `try_lookup_symbol`, a call to `dlsym` is used to obtain the address of any function based on the provided function name. `dlsym` is a function that retrieves the address of a symbol from any loaded dynamic shared objects. The `RTLD_NEXT` flag allows the use of the real original definition of the function while embedding only the necessary logic to implement the statistics gathering function.

Once the address of the target function, such as `SSL_SESSION_get_master_key`, is found, the function can be directly invoked.

```
static void copy_master_secret(const SSL_SESSION *session,
    unsigned char *master_key_out, int *keylen_out)
{
    #if OPENSSL_VERSION_NUMBER >= 0x10100000L
        static size_t (*func)();
        if (!func) {
            func = lookup_symbol("SSL_SESSION_get_master_key");
        }
        *keylen_out = func(session, master_key_out, SSL_MAX_MASTER_KEY_LENGTH);
    // ...SNIP...
}
```

Now we want to get the code to run at the right time. Since we can override symbols that are dynamically linked in other shared libraries, we might want to override some commonly used symbols in those shared libraries by defining functions with the same signatures in our custom library. In our example, our goal is to get the master key, so it might be a good idea to fetch the master key whenever there is a new SSL connection. This is exactly what the code does, as shown below. Note that the function `copy_master_secret` is invoked in `tap_ssl_key`.

```
int SSL_connect(SSL *ssl)
{
    static int (*func)();
    if (!func) {
        func = lookup_symbol(__func__);
    }
    SSL_TAP_STATE(state, ssl);
    int ret = func(ssl);
    tap_ssl_key(ssl, &state);
    return ret;
}
```

Whenever `SSL_connect` is called by the target application, the definition in our `LD_PRELOAD` library will be resolved instead, and it will call the function we wrote, achieving our goal.

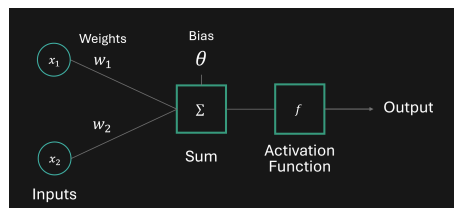
Conclusion

By leveraging the technique of Function Interposition, it is able to dynamically analyze the Linux application and inspect the encrypted HTTPS traffic without needing extensive reverse engineering skills. This also provides valuable insights into the application's behavior. If you find yourself needing to analyze similar applications, this straightforward approach could be a powerful tool in your toolkit.



Neural Networks: Foundation

All neural networks are built and connected together using something called **neurons**. A single neuron can take **n** inputs, and each input is **multiplied** by a weight value. Then, all the values are summed together along with a bias value and we pass the result through an activation function. We can chain many of these neurons together to form a neural network, but how do these neural networks learn and output the results we want?



Neural Networks: Learning

Neural networks “learn” by trying to reduce the amount of errors of a given loss function by updating the weight and bias values. In other words, the neural network tries to output the results we want, and we use the loss function as a metric to determine how accurate the model is. This is known as “**Supervised Learning**”.

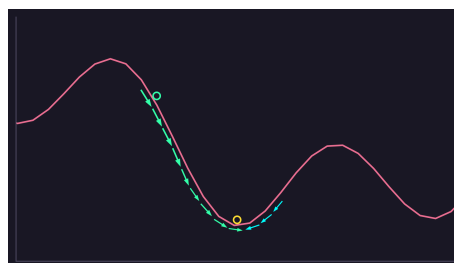
Calculus (M1/M2) students should know that we can sometimes find a function’s local minima or maxima using the first-derivative test, however this becomes infeasible when dealing with more complicated functions.

A better method to find a function’s critical point while being computationally cheaper would be using **gradient descent**. The idea is that we can just find the slope of a given point on the function. If the slope is positive, we can shift the point a little bit to the left by **adjusting the weight and bias values**, and vice versa for negative. After iterating this process for a while, we should reach a local minima point, reducing the errors made by our neural network. Most people describe this process as “pushing a ball down a hill”.

This process can be written as the following equation:

$$\theta = \theta - \epsilon(\nabla J(\theta))$$

Where θ represents the parameters of the neural network model (weights and biases), and we take the first derivative of the loss function J to obtain the gradient, then we multiply it with a small value ϵ known as the learning rate, which controls how much we move, and at last we subtract it with our original parameters in order to move downwards in the loss function. If we use the function in the image as an illustration and iterate this process for a long time, we would reach and oscillate around the yellow point.



Adversarial Attacks: What is it?

Adversarial attacks is an attack caused by manipulating data in a way such that the adversarial data is indistinguishable from other untampered data by human observation, yet it causes an AI model to misclassify the data. These attacks may cause the AI model to malfunction and make wrong output, which could have disastrous consequences. One such example would be [tricking a self-driving car to suddenly accelerate while driving](#)¹.

¹MIT Technology Review: Hackers can trick a Tesla into accelerating by 50 miles per hour

Adversarial Attacks: Fast Gradient Sign Method (FGSM)

One of the simplest and most famous adversarial attack method is called the Fast Gradient Sign Method[1]. To understand it, let's recall how we update the parameters of a neural network - we calculate the gradient of the loss function, then subtract the gradient from the parameters to reduce the total error of the model. Now, what happens if we decide to **add** the gradient to the parameters? In this case, we are causing the model to converge to a local maxima, which means increasing the error of our model:

$$\theta = \theta + \epsilon(\nabla J(\theta))$$

But our goal here is not to create a poor model, but rather to **create an input** that causes the model to perform poorly. To achieve this, we can add the gradient to the input data \mathbf{x} instead of the model parameters (**remember, multiplication is commutative**):

$$\mathbf{x} = \mathbf{x} + \epsilon(\nabla J(\mathbf{x}))$$

At last, since the precision for most inputs is limited. For example, image pixels are often represented only using 8 bits and will hence discard information below $1/255$, therefore we would need gradients that are large enough to cause the model to misclassify the input, but also small enough so that it is not susceptible by human observation. To achieve this, one method is to only take the sign of the gradients:

$$\mathbf{x} = \mathbf{x} + \epsilon \cdot \text{sign}(\nabla J(\mathbf{x}))$$

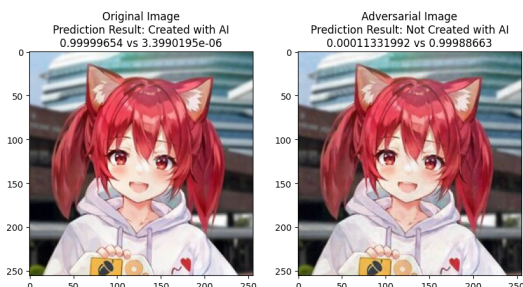
Now, if we iterate this process for a few rounds, we are now able to modify a normal input to become an adversarial input.

Adversarial attacks: FGSM Demo

To demonstrate FGSM, we are going to use it on an [image classification model](#)², which detects whether an image is AI generated or not.

The image classification model takes in 256×256 images, passes the data through 3 convolution layers with MaxPooling Layers, then the image gets flattened and goes through a dense layer, and at last it goes through another dense layer with Softmax activation function, which gives 2 probabilities, representing how likely the image is generated by AI.

When we pass an AI-generated image through the model, the model predicted the image is AI with 99.99% probability. However after 3 iterations of FGSM, the model now predicts the image as not generated by AI with 99.98% probability. Yet, the difference between the images before and after FGSM is hardly noticable just by looking, with the maximum pixel value difference being 12 only. The demo code and data is located [here](#)³.



References

- [1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples." arXiv, Mar. 20, 2015. doi: [10.48550/arXiv.1412.6572](https://arxiv.org/abs/1412.6572). Available: <http://arxiv.org/abs/1412.6572>. [Accessed: Jun. 24, 2024]

²ArtifAI Detector: <https://github.com/AustinBoyuJiang/ArtifAI>

³FGSM Demo: <https://github.com/Vyanide/FGSM-B6A-Demo>



I was in SINCON 2024 to promote the products of my company. So I was like spending my first 3 days on booth setup and staying around the booth.

What have I done?

I was given a section to give a talk on any topic. In the end, I tried to explain to the audience how to use NIST Cybersecurity Framework (CSF) and Cyber Defense Matrix (CDM) to check if there are any missing plates for the defense mechanism. Although the content is about ransomware and data leak detection in darkweb, some audience found my talk interesting when CDM is infused in my slides.

When I had some spare time not at my booth, I tried to attend the following workshops:

- Toyota Motor Corporation x Car Security Quarter (CSQ) — Automotive Security 101
- Introduction to Software Defined Radio (SDR) Workshop

Car Hacking

At the beginning of the session, we were asked to install **can-utils** on our VM. RAMN and CAN Bus.

As the workshop only has 15 **RAMN** (Resistant Automotive Miniature Network) prepared and the workshop was too popular, we had to team up to try playing with it. I was lucky and my teammate let me do all the hands-on parts. Either their environment is not functioning, or they find it more fun to look at others playing with the commands.

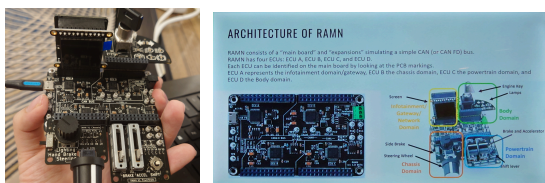
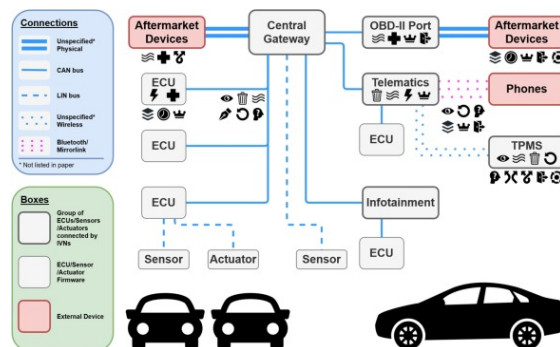


Image of RAMN and its architecture



How communication works in a modern vehicle

Source: A classification of attacks to In-Vehicle Components

Car systems nowadays make use of many communication systems that work like neural networks. The main aim of our workshop is to hijack the CAN bus so that the components (e.g. accelerator, brake, turn signals) work in the way the attacker wishes to.

Of course, there are other communication protocols. We are targeting the CAN bus this time.

Packet Sniffing

cansniffer is used to track down the data and see the data received from RAMN. Items highlighted in red represent an update of value. Note that **cansniffer** keeps on updating the console. It is impossible for users to track the historical record.

```
12|ms | ID | data ... < can0 # l=20 h=100 t=500 slots=10 >
00010 | 024 | 00 00 32 B7 42 AD 31 99 ..2.0.1.
00011 | 039 | 00 00 32 B7 42 AD 31 99 ..2.0.1.
00010 | 062 | 0B 6A 32 B7 B5 64 70 0B ..j2..dp.
00099 | 077 | 00 01 05 11 9C 61 41 37 .....aA7
00099 | 098 | 00 00 05 11 AB 0B 83 36 .....6
00100 | 150 | 01 00 05 11 CE 6C 3F 8E .....1?
00099 | 1A7 | 00 00 05 11 AB 0B 83 36 .....6
00099 | 188 | 01 00 05 11 CE 6C 3F 8E .....1?
00098 | 188 | 00 00 05 11 AB 0B 83 36 .....6
00099 | 1D3 | 00 00 05 11 AB 0B 83 36 .....6
```

candump was used to track the data of CAN Bus. By using the value and mask pair, we can trim the data and have every instance printed line by line on the console.

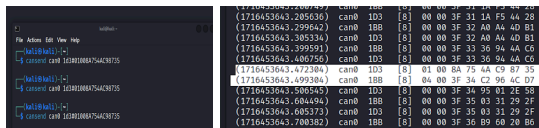
```
↳ candump can0,1BB:7BB
can0 188 [8] 00 00 19 4D 29 4B 29 B2
can0 188 [8] 00 00 19 4E 93 1A 20 2B
can0 188 [8] 00 00 19 4F 05 2A 27 5C
can0 188 [8] 00 00 19 50 F0 27 2F D1
can0 188 [8] 00 00 19 51 66 17 28 A6
can0 188 [8] 00 00 19 52 DC 46 21 3F
```

Injection

We capture a handbrake signal as follows:

```
(1716453526.483563) can0 1D3 [8] 01 00 8A 75 4A C9 87
35We attempt to send it back using canbus, trying to
pretend as a normal input or even an interception
```

cansend can0 1d3#01008A754AC98735Result:
RAMN recognised the handbrake signal and
actually braked the car.

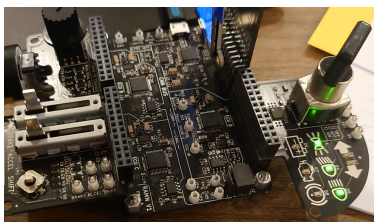


It is also possible to use **camplayer** to replay a
signal capture without knowing which ID
represents the exact component action (just
packet-by-packet.)

Capture: **candump can0 -f ./can0.log**

Replay: **can player -I ./can0.log -l i can0=can0**

I tried to play with all buttons while capturing the
sequences but I forgot to take a video. In short, you
may see that the headlight turns on itself by
replaying the whole sequence captured from
candump.



Demo and CAN Log

If you want to see the video demo and my CAN bus
capture, please PM me, and I will share it with you.

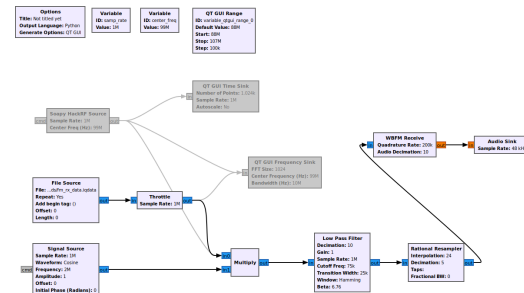
SDR

The SDR workshop mainly covered the following:

- Wave theory
- Modulation / Demodulation
- GNU Radio hands-on

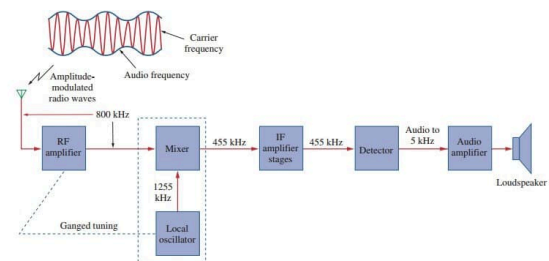
While we were learning the theories, we had to
install **gnuradio** at the same time. Speakers also
suggested using OS **PENTOO** with boot USB. This
allows the hardware to communicate directly and
facilitates the use of HackRF One.

An IQ file (I/Q data) was provided in the workshop



and we were asked to demodulate it in order to
hear the FM broadcast recorded. The IQ files work
as a "raw" file of the signal. Within **gnuradio**, we
are able to create "flowgraphs", which tells
gnuradio how to process data into a desired
format. Once it is completed, it should look like
this:

You may find it similar to an FM receiver.



Extra

I won the lucky draw from Offensive Security,
which is a box of Lego...



Credits and Afterwords

- Editor-in-chief: *GonJK*

- Article contributors:

<i>a1668k</i>	<i>apple</i>	<i>botton</i>	<i>Eason</i>
<i>ensy</i>	<i>gldanoob</i>	<i>GonJK</i>	<i>grhkm</i>
<i>harrier</i>	<i>hoifanrd</i>	<i>Mystiz</i>	<i>Ozetta</i>
<i>vikychoi</i>	<i>Vow</i>		

- Design: *apple*

- Cover art: *GonJK*

- Article review (Knowledge-wise):

<i>apple</i>	<i>cire meat pop</i>	<i>grhkm</i>	<i>Ozetta</i>
--------------	----------------------	--------------	---------------

If you have any comments on the newsletter, please don't hesitate to drop a direct message through Facebook, X (Formerly known as Twitter), E-mail, or even Discord – let us know what's on your mind!

As you dive into the articles, I hope you feel the passion and dedication that went into each piece. Thank you once again to our incredible writers and reviewers. Your contributions are deeply valued, and hope you enjoyed these articles.

Connect Us



blackb6a



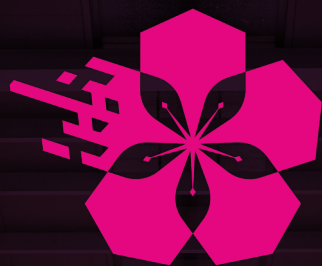
blackb6a



team@b6a.black



blackb6a



Black Bauhinia

<https://b6a.black>