KULIAH TAMU · IF4020 KRIPTOGRAFI · ITB

Kriptografi di Dunia nyata dan Disrupsi AI

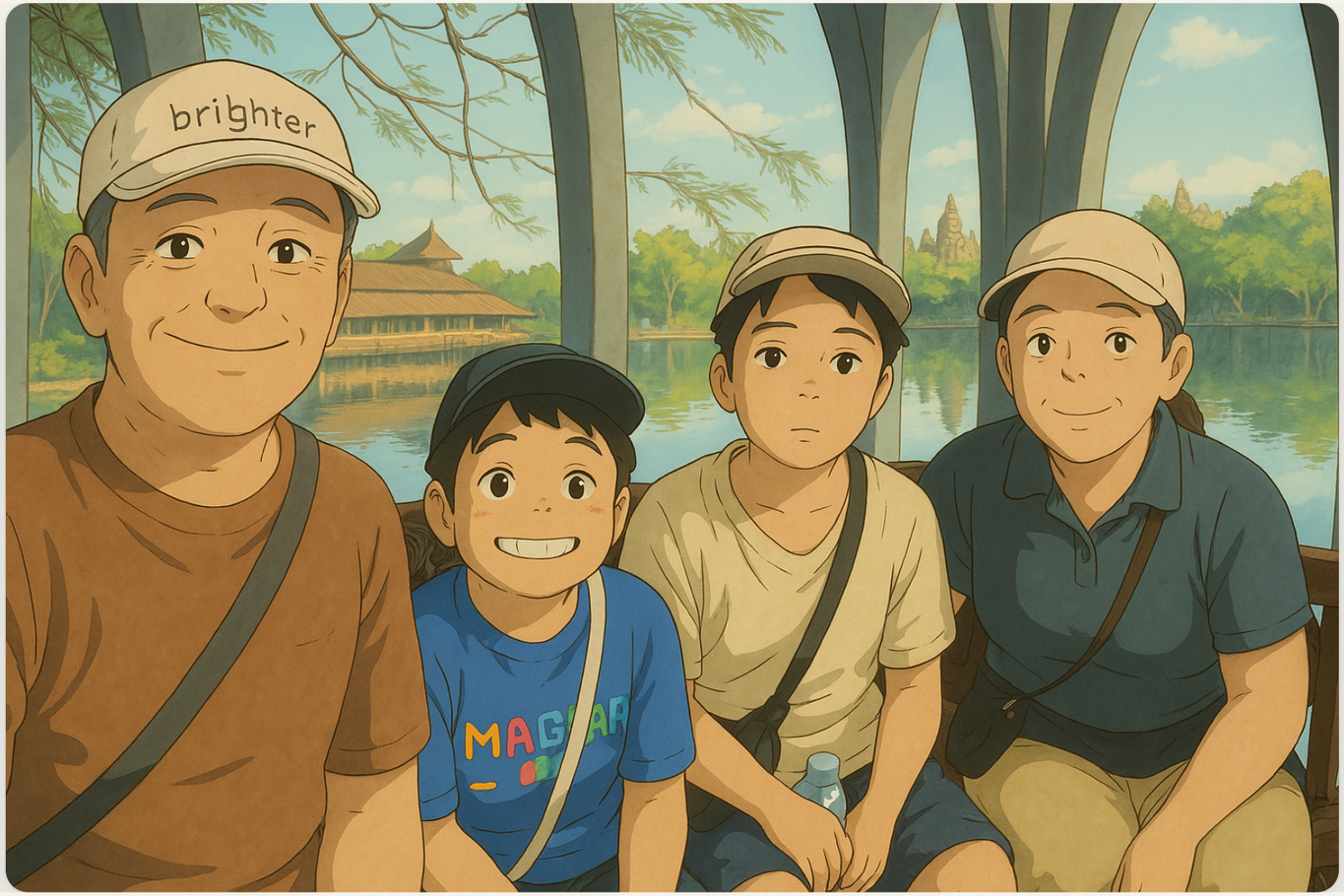
Yohanes Nugroho, S.T., M.T., M.Sc.

27 Oktober 2026

Tentang Saya

- Yohanes Nugroho
- Sertifikasi: OSCP, OSEP, OSWE, OSED, OSCE3
- Pendidikan:
- S1 Teknik Informatika ITB (ST)
- S2 Teknik Informatika ITB (MT)
- Master Cyber Security University of London (MsC)

Di Chiang Mai Thailand sejak 2007



Pengalaman Kerja

- Litic, Co., Ltd Programmer C++ (Tailan)
- Senior Security Engineer GeneusDNA. (Tailan)
- Advisory Board member GeneusDNA. (Tailan)
- Pentester di PT Xynexis International
- Membantu bebras Indonesia (Computational Thinking)
- Membantu TOKI

Kisah masuk dunia security

- Dulu pernah bandel waktu S1 : https://blog.compactbyte.com/2019/08/13/cerita-hacking-dari-masa-lalu/ (1998)
- Menjadi administrator jaringan selama kuliah S1
- S2 di ITB, hampir menjadi dosen (2003-2006)


Most cyber security experts today
Their younger years of illegally breaking into computers and websites

Outline Materi

- Kriptografi di dunia nyata: lebih dari sekedar memanggil API
- Lima cerita nyata (2015 - 2025), plus satu kasus bulan ini (2026)
- Disrupsi AI: apa yang sudah bisa, apa yang belum
- Peran manusia di zaman AI
- Pekerjaan saya sekarang
- Tips belajar

KRIPTOGRAFI DI DUNIA NYATA

BAGIAN 1

Kriptografi di Dunia Nyata

Bukan sekedar memakai library

KRIPTOGRAFI DI DUNIA NYATA

"Nanti juga cuma pakai library"

- Banyak yang berpikir: kriptografi itu bagus untuk diketahui
- Di dunia kerja: panggil `AES.encrypt()`, `sha256()`, selesai
- Kenyataannya: library-nya benar, cara pakainya yang salah
- Dan kadang kita harus membongkar kriptografi buatan orang lain

KRIPTOGRAFI DI DUNIA NYATA

Kriptografi dan Reverse Engineering

- Target saya jarang berupa source code
- Biasanya: binary, APK, firmware, malware
- Untuk tahu kriptografinya, harus reverse engineering (RE) dulu
- Pertanyaan RE: algoritma apa? key dari mana? nonce dari mana?


What people think pentesting is like
What pentesting is actually like
boredpanda.com

KRIPTOGRAFI DI DUNIA NYATA

2015

techinasia.com/indonesia-go-jek-security-loopholes

Hacker finds security bugs in Go-Jek app



Go-Jek is a popular app for hailing motorbike taxis.

Earlier this week, *CNN Indonesia* reported that an Indonesian programmer in Thailand named Yohanes Nugroho identified security flaws in on-demand motorcycle app Go-Jek.

CERITA 1

Bank dan hashCode() (2015)

Pentingnya tahu secure hash


vixentael
@vixentael
The story of my life.
Work in cyber security
Do something super cool during an engagement
Can't tell anyone
Can't tell anyone
imgflip.com
8:31 PM · Mar 29, 2021 · Twitter Web App
253 Retweets
47 Quote Tweets
1,671 Likes

Konteks

- Salah satu bank di Indonesia, tahun 2015
- Aplikasi mengirim transaksi dalam bentuk XML
- Integritas request "diamankan" dengan hash + secret

Kodenya Kira-kira Seperti Ini

JAVA

```java
String xml = "<trx><from>1234</from><to>5678</to>"
           + "<amount>10000</amount></trx>";

int mac = ("secret" + xml).hashCode();

// dikirim ke server: xml + mac
```

- Idenya: tanpa tahu `secret`, attacker tidak bisa membuat `mac` yang valid
- Konsepnya mirip MAC. Masalahnya ada di `hashCode()`

Apa itu `String.hashCode()`?

- Dibuat untuk `HashMap`, bukan untuk keamanan
- Output hanya 32 bit
- Rumusnya linear:

OUTPUT

```
h(s) = s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]   (mod 2^32)
```

- Akibatnya: `h(A + B) = h(A) * 31^len(B) + h(B)`

Hash yang linear bisa dihitung ulang

Dari karakter ke hash, lalu dari satu MAC sah ke MAC palsu.

Asli: `<amount>10000</amount>` Baru: `<amount>99000000</amount>`

KONTRIBUSI BARU
-689732762

+ →

HASH PESAN BARU
1534933257

= →

MAC PALSU
845200495

$$MAC' = -689732762 + 1534933257 = 845200495$$

✓ Server menerima MAC baru tanpa penyerang mengetahui secret.

08 **Tambahkan hash pesan baru**

MAC palsu sama dengan hasil server. Forge tidak menerima secret sebagai argumen.

Tiga Cara Menyerangnya

- **Brute force**: hanya 2^{32} kemungkinan, detik-an di CPU
- **Collision gratis**: `"Aa".hashCode() == "BB".hashCode()`
  - `"Aa"`: `65*31 + 97 = 2112`, `"BB"`: `66*31 + 66 = 2112`
  - Karakter pertama turun 1 (-31), karakter kedua naik 31 (+31): hash tetap
  - Bisa digabung: `"AaAa"`, `"AaBB"`, `"BBAa"`, `"BBBB"` hash-nya sama semua
- **Forgery**: dari SATU pasangan (xml, mac) yang valid, buat mac untuk xml apa saja

Forgery: Kode Attacker

JAVA

```java
// h(SECRET + xml) = h(SECRET) * 31^len(xml) + h(xml)
static int pow31(int n) {
    int r = 1;
    for (int i = 0; i < n; i++) r *= 31;   // overflow = mod 2^32
    return r;
}

// dari (xml, mac) yang valid → mac untuk xml2
static int forge(String xml, int mac, String xml2) {
    int k = mac - xml.hashCode();     // = h(SECRET) * 31^len(xml)
    return k * pow31(xml2.length() - xml.length())
             + xml2.hashCode();
}
```

Demo

OUTPUT

```
$ javac BankHash.java && java BankHash
Request asli : <trx><from>1234</from><to>5678</to><amount>10000</amount></trx>  mac=-789932763
Request palsu: <trx><from>1234</from><to>6666</to><amount>99000000</amount></trx>  mac=1168376508
Server verify: true
"Aa".hashCode()=2112  "BB".hashCode()=2112
```

- Rekening tujuan dan jumlah diubah, server menerima
- Secret tidak pernah diketahui

Solusinya

- Gunakan MAC yang benar: **HMAC-SHA256**
- Atau signature (RSA / ECDSA / Ed25519) jika perlu non-repudiation
- Bandingkan MAC dengan constant-time compare
- Tambahkan nonce / timestamp untuk mencegah replay

Yang Menarik dari Cerita Ini

- Bug ini sangat sederhana
- Tapi tidak ada yang menyadari, bertahun-tahun
- Dari temuan ini saya dipercaya mengetes lebih banyak bank
- Hari ini: AI akan langsung melihat ini dalam hitungan detik

Tapi Itu 10 Tahun yang Lalu...

- "Siapa yang pakai `hashCode()` untuk aplikasi bank?"
- "Mungkin 10 tahun lalu developer memang belum paham security?"
- Mari lihat kasus yang baru terjadi **bulan ini** (Oktober 2026)

eMudhra: Certificate Authority (Oktober 2026)

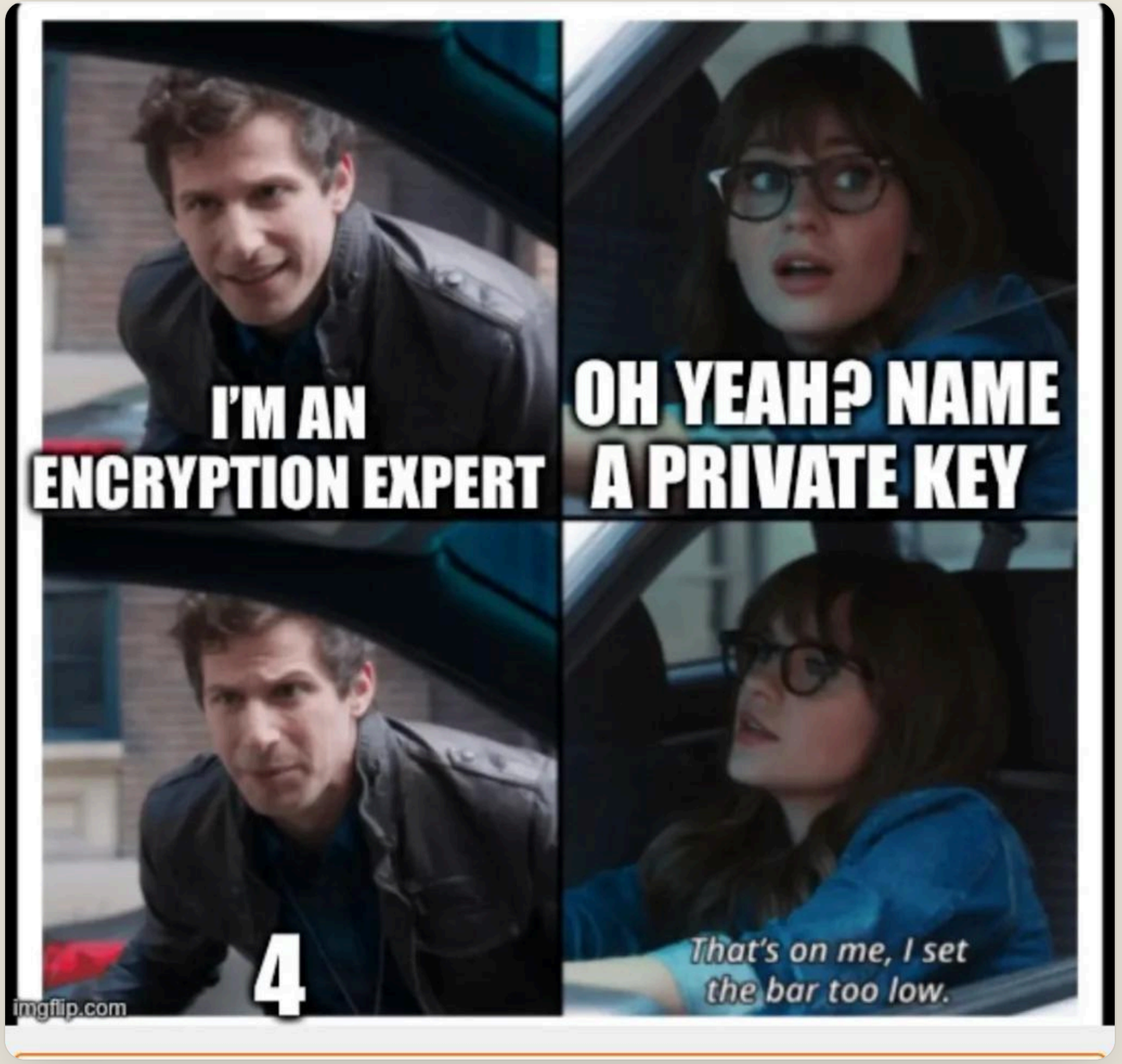
- CA dari India, root-nya dipercaya browser dan OS
- 8 intermediate CA: 6 untuk TLS, 2 untuk code signing
- Key ECDSA P-384: private key seharusnya angka acak **384 bit**
- Ternyata: semua private key-nya **di bawah 256**
- Salah satunya (`EM OV TLS CA - G3C`): private key = **4**

Kenapa Bisa Terjadi?

- Key dibuat di **HSM** saat key ceremony, 30 April 2025
- Bug di script: membuat set key kedua yang lemah
- Label key-nya mirip, yang lemah terpilih untuk membuat CSR
- Sudah di-cross-sign oleh Sectigo
- Hampir 18 bulan tidak ada yang sadar, sampai ada yang men-scan sertifikat publik (17 Oktober 2026)

Pelajaran: Kesalahan Bodoh Akan Selalu Ada

- 2015: bank memakai `hashCode()` sebagai MAC
- 2026: CA dengan private key = 4
- Bukan soal zaman: manusia selalu bisa salah, dan sekarang AI juga
- Alat dan prosedur yang mahal (HSM, audit) tidak menjamin hasilnya benar
- Cek hasil akhirnya: satu test sederhana (coba d = 1 sampai 255) sudah cukup

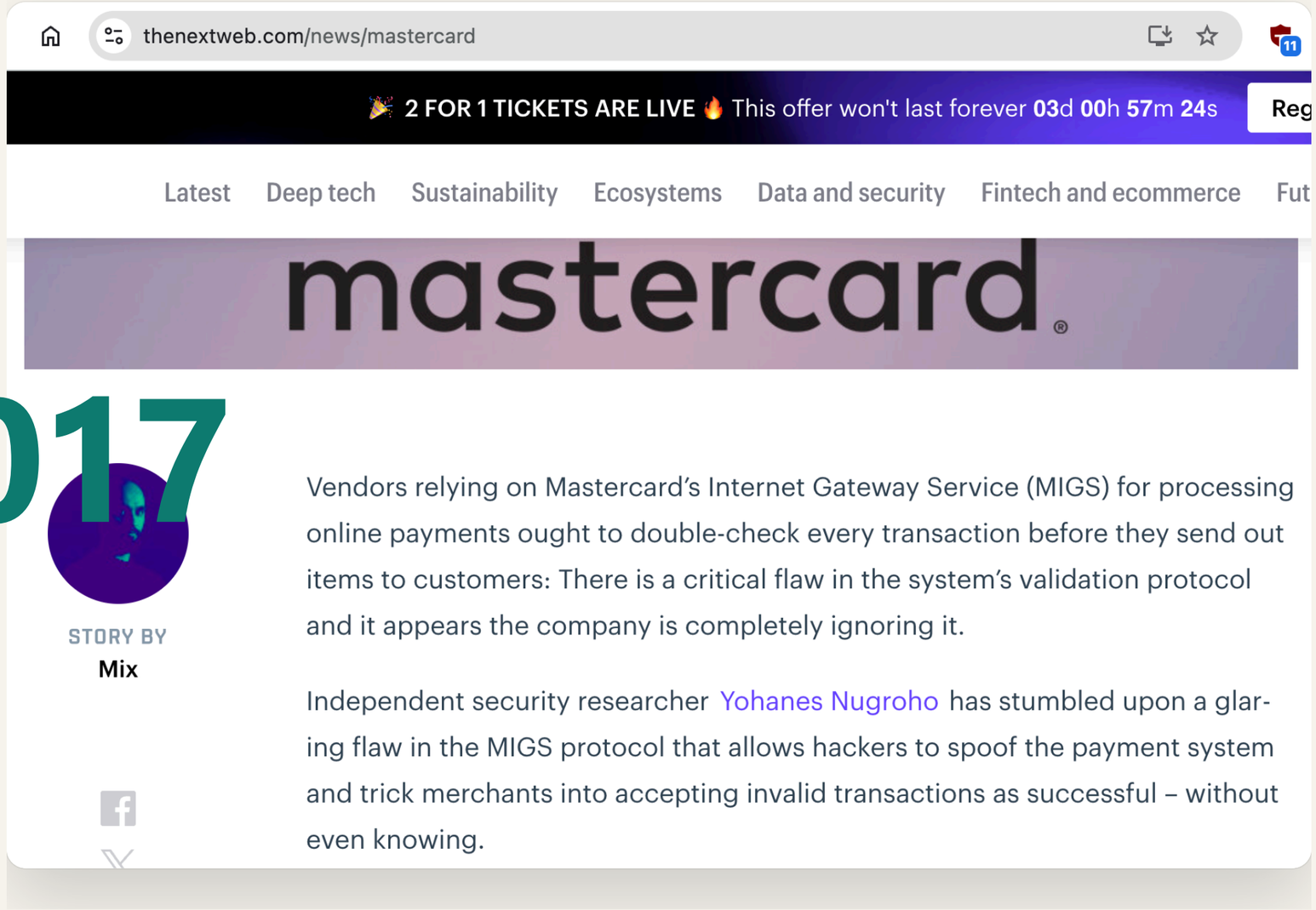

I'M AN ENCRYPTION EXPERT
OH YEAH? NAME A PRIVATE KEY
4
That's on me, I set the bar too low.
imgflip.com

CERITA 2

Mastercard Internet Gateway Service (2016 - 2017)

Perusahaan besar tidak lolos dari masalah kriptografi

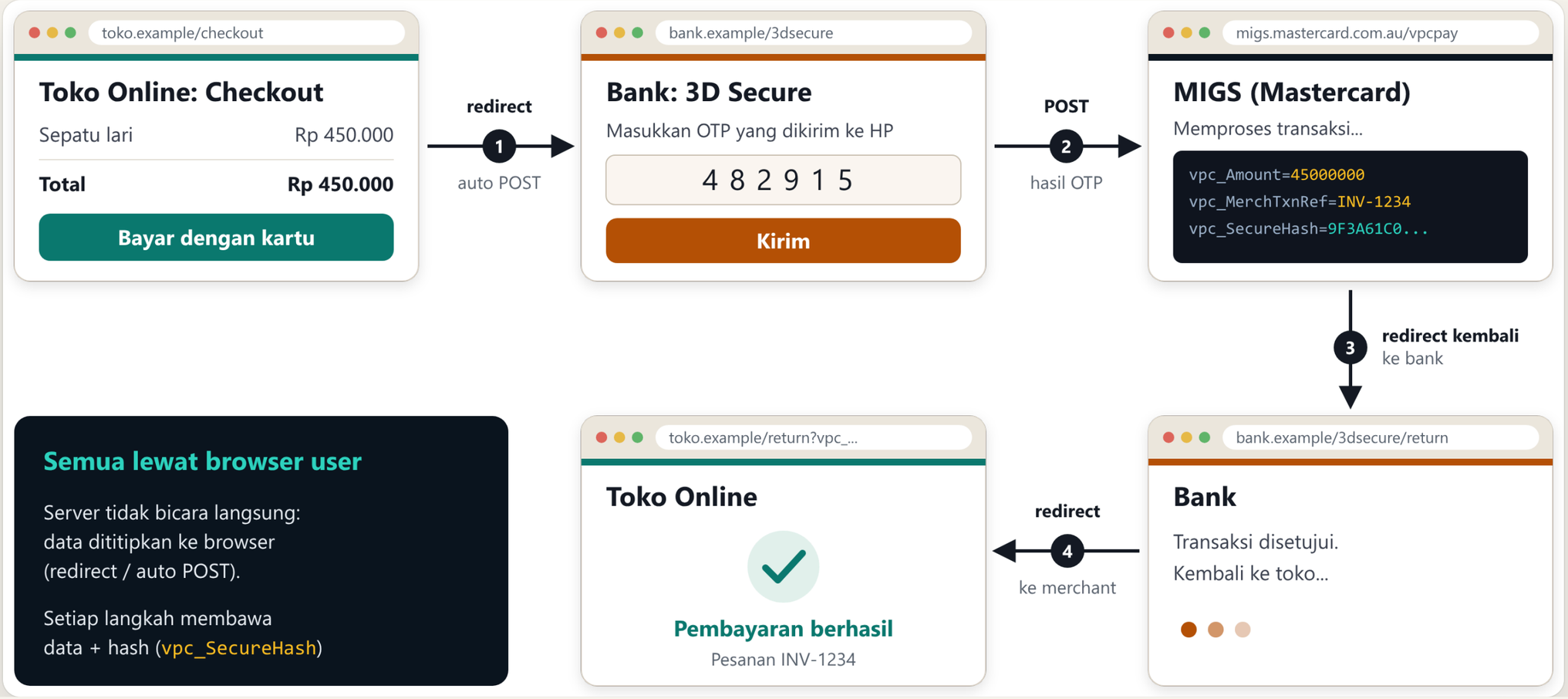
2016/2017


thenextweb.com/news/mastercard
2 FOR 1 TICKETS ARE LIVE This offer won't last forever 03d 00h 57m 24s
Reg
Latest
Deep tech
Sustainability
Ecosystems
Data and security
Fintech and ecommerce
Fut
mastercard
STORY BY
Mix
Vendors relying on Mastercard's Internet Gateway Service (MIGS) for processing online payments ought to double-check every transaction before they send out items to customers: There is a critical flaw in the system's validation protocol and it appears the company is completely ignoring it.
Independent security researcher Yohanes Nugroho has stumbled upon a glaring flaw in the MIGS protocol that allows hackers to spoof the payment system and trick merchants into accepting invalid transactions as successful – without even knowing.

Alur Pembayaran MIGS

- Merchant -> payment gateway perantara -> MIGS -> bank (3D Secure)
- Komunikasi antar server dilakukan **lewat browser user** (auto POST)
- Semua data di-"sign" dengan hash + secret
- Jika signing dan verifikasinya benar, user tidak bisa mengubah data

Alur Pembayaran MIGS: Lewat Browser



Konstruksi Hash MIGS (Versi MD5)

- `MD5(secret + data)`
- Data: ambil semua parameter `vpc_*`, urutkan berdasarkan nama
- Gabungkan **nilainya saja**, tanpa nama dan tanpa delimiter

OUTPUT

```
vpc_Amount=10000  vpc_Card=1234567890123456  vpc_Name=Joe
data = "10000" + "1234567890123456" + "Joe"
     = "10000123456789012345 6Joe"
```

Flaw-nya: Ambigu

OUTPUT

```
vpc_Amount=1  vpc_B=0000  vpc_Card=1234567890123456  vpc_Name=Joe
data = "1" + "0000" + "1234567890123456" + "Joe"
     = "10000123456789012345 6Joe"      ← SAMA
```

- Parameter baru `vpc_B` disortir setelah `vpc_Amount`, "memakan" digit
- Hash sama, amount berubah dari 10000 menjadi 1
- MacBook 2000 USD dibayar 2 USD

Data berbeda, byte yang sama

Lihat digit berpindah field sementara deretan byte yang ditandatangani tetap sama.

vpc_Amount	vpc_B	vpc_Card	vpc_Name
1	0000	1234567890123456	Joe

↓ encode nama + panjang + nilai

```
Amount:5:10000 | Card:16:1234567890123456 | Name:3:Joe
Amount:1:1 | B:4:0000 | Card:16:1234567890123456 | Name:3:Joe
```

Representasi berbeda → MAC dapat membedakannya

Warna menunjukkan kepemilikan digit, bukan perubahan digit.

05 **Beri batas yang tidak ambigu**

Contoh length-prefix membedakan kedua request. Nama, panjang, dan nilai semuanya harus ikut diautentikasi.

Versi Baru: HMAC-SHA256

- Sekarang: nama field + delimiter `&` + HMAC-SHA256
- Tapi dokumentasi: nilai **tidak di-URL-encode** sebelum hashing
- Nilai yang mengandung `&` dan `=` bisa "menyamar" jadi field lain

OUTPUT

```
Amount=100&Card=1234   CVV=555   →  HMAC("Amount=100&Card=1234&CVV=555")
Amount=100   Card=1234   CVV=555 →  HMAC("Amount=100&Card=1234&CVV=555")
```

Bonus: Signature yang Tidak Dicek

- Payment gateway perantara tidak memverifikasi hash dari MIGS
- Ubah satu karakter response: `F` (gagal) menjadi `0` (sukses)
- Kartu apa saja -> transaksi sukses
- Bug yang sama saya temukan di payment gateway lain saat pentest

Pelajaran: Canonicalization

- Yang di-hash/sign harus punya representasi **unik**
- Pakai format yang jelas: length-prefix, encoding konsisten, atau JSON canonical
- Lebih baik lagi: sign seluruh body apa adanya
- Selalu verifikasi di sisi server, jangan percaya "pihak lain akan mengecek"

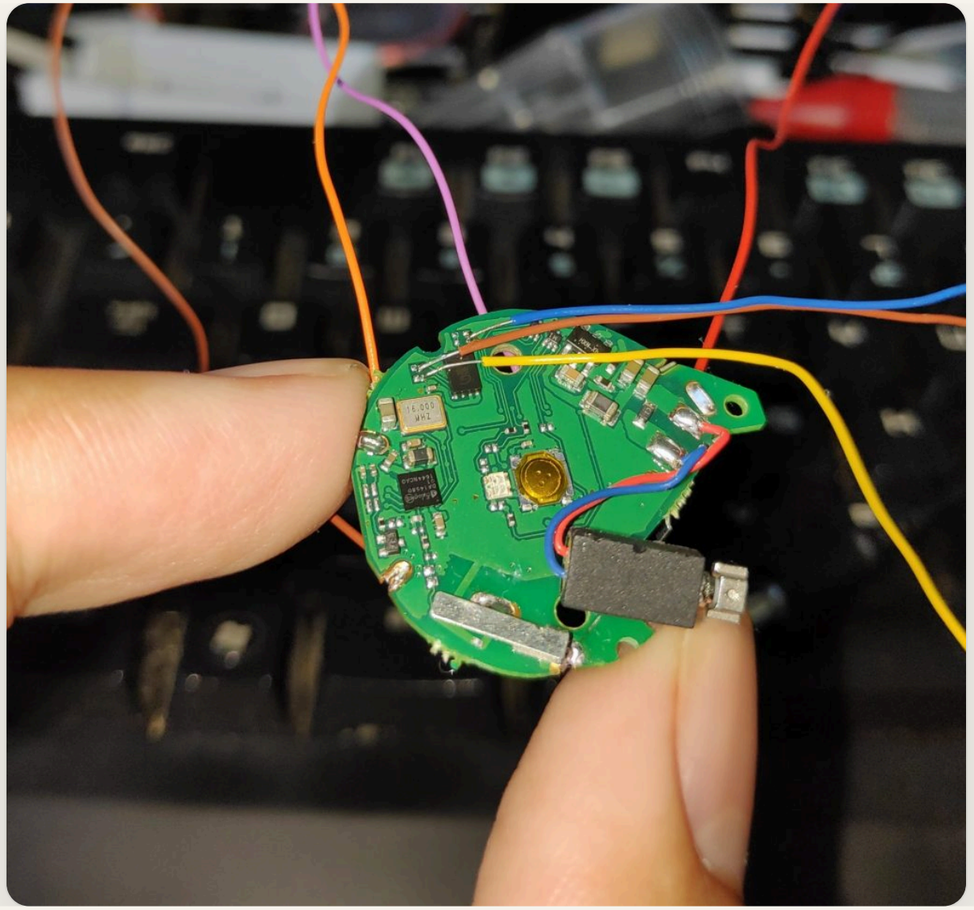
CERITA 3

Pokemon GO Plus (2018)

Enkripsi di hardware

Pokemon GO Plus

- Gelang Bluetooth Low Energy (BLE) untuk game Pokemon GO
- Sebelum bisa dipakai: proses "certification" dengan app
- Analisis traffic BLE saja tidak cukup: AES dengan key yang tidak pernah dikirim



Langkah Reverse Engineering Hardware

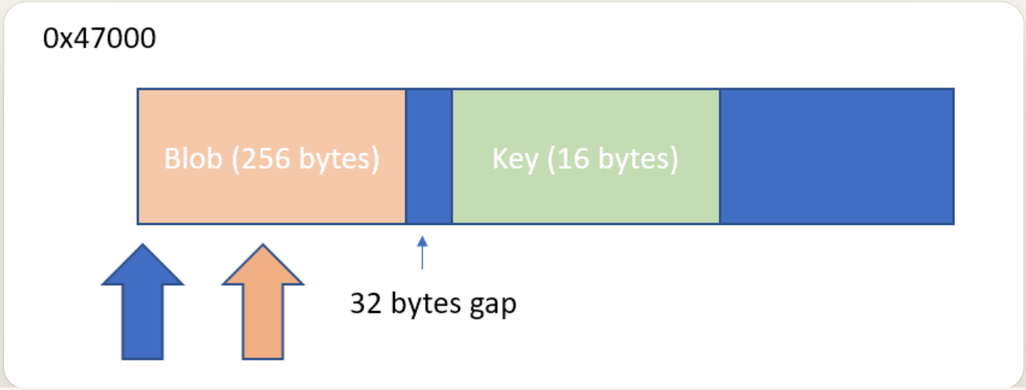
- Solder kabel ke SPI flash, baca dengan `flashrom` dari Raspberry Pi
- Firmware terenkripsi, tapi key-nya sudah diketahui dari riset sebelumnya
- Cortex-M0 (Thumb), tanpa simbol
- Trik: compile contoh dari SDK DA14580 -> dapat ELF dengan debug symbol untuk perbandingan

BASH

```
flashrom -p linux_spi:dev=/dev/spidev0.0,spispeed=1000 -r pgp.bin
```

Mencari Key

- Device key tidak ada di SPI flash, tapi di area OTP (one time programmable)
- Firmware mengirim blob 256 byte dari OTP saat challenge
- Patch **1 byte**: geser offset, key ikut terkirim lewat Bluetooth



Protokol Certification (Disederhanakan)

- Device: buat challenge A, session key Sk, nonce
- Paket 80 byte (berisi Sk, A terenkripsi, MAC Bluetooth) dienkripsi dengan device key
- App (tahu device key) membuka paket, mengirim balik A
- Beberapa ronde lagi, final: string `PokemonGooooooooo`
- Reconnect: `AES(Sk, A) xor B` dan sebaliknya

Tiga Fungsi "Custom" dari Firmware

```c
// aes_ctr: counter block = [01][13 byte nonce][00 00], increment byte terakhir
// aes_hash:
nonce_hash[0] = 57;                          // ???
memcpy(nonce_hash + 1, nonce, 13);
nonce_hash[14] = (datalen >> 8) & 0xff;
nonce_hash[15] = datalen & 0xff;
aes_encrypt(nonce_hash) → tmp; for tiap blok: tmp = AES(tmp ^ blok)
// encrypt_block: hasil aes_hash di-XOR dengan AES(counter block 0)
```

- Waktu itu saya dokumentasikan sebagai tiga fungsi terpisah
- Pertanyaan: ini benar-benar custom?

Ternyata: AES-CCM Standard

PYTHON

```python
pgp = aes_ctr(key, nonce, data) + encrypt_block(key, nonce, aes_hash(key, nonce, data))
ccm = AESCCM(key, tag_length=16).encrypt(nonce, data, None)
print("identik?", pgp == ccm)
```

OUTPUT

```
identik?  True
```

- `57 = 0x39` = flags CCM: `8*M' + L'` dengan tag 16 byte (`M'=7`), counter 2 byte (`L'=1`)
- CCM memang dipakai di Bluetooth LE

Sampingan: Autentikasi Tanpa Public Key

- Pokemon GO Plus: challenge-response dengan AES saja, tanpa RSA / ECC
- Key ditanam di hardware saat produksi, "didistribusikan" secara fisik
- Contoh lain: SIM card (key di SIM, operator punya salinannya)
- Kelemahan: key bisa diekstrak (seperti tadi), pihak pusat tahu semua key

Kalau Komputer Kuantum Datang

- Algoritma Shor memecahkan RSA, DH, ECC; AES cukup pakai key 256 bit
- Opsi 1: key simetris didistribusikan fisik (cara lama), tidak scalable
- Opsi 2: PQC (ML-KEM, ML-DSA); belum teruji waktu, kandidat lain (Rainbow, SIKE) dipecahkan dengan laptop
- Praktiknya sekarang: hybrid X25519 + ML-KEM (Chrome, OpenSSH, Signal)

CERITA 4

Ransomware PDN, LockBit dan Babuk (2024)

Seharusnya tidak perlu panik

Konteks: Dua Ransomware

- Juni 2024: Pusat Data Nasional (PDN) terkena ransomware, group Brain Cipher
- Tebusan yang diminta: 8 juta USD
- Awalnya yang diumumkan hanya LockBit
- Ternyata ada **dua** ransomware, keduanya dari builder / source yang bocor
- Babuk baru ketahuan setelah penyerang merilis dekriptor gratis

2024

Duh, Ternyata Ada Dua Ransomware yang Serang PDNS 2

Anggoro Suryo - detikInet

Kamis, 04 Jul 2024 16:18 WIB



Ilustrasi ransomware. Foto: Microsoft Designer

Jakarta - Brain Cipher sudah merilis kunci untuk membuka enkripsi ransomware yang menyerang Pusat Data Nasional Sementara (PDNS) 2. Namun ternyata ada dua ransomware yang menyerang PDNS 2.

Hal tersebut diungkap oleh Yohanes Nugroho, seorang programer yang

Dua Ransomware, Dua Jenis Kunci Publik

Target	Ransomware	Data	Master key
Windows	LockBit 3	Salsa20	RSA-1024
ESXi (hypervisor)	Babuk	Sosemanuk	Curve25519 (ECC)

LockBit 3 (Windows)

- Builder bocor, siapa saja bisa membuat varian
- Builder menghasilkan `locker.exe` (public key) dan `unlocker.exe` (private key) per target
- Perilaku tergantung `config.json`, jadi butuh sampel **eksak**

Hierarki Key

- Setiap file: `KEY_PER_FILE` (Salsa20) untuk isi file
- `KEY_PER_FILE` + nama file disimpan di footer
- Footer dienkripsi dengan `KEY_PER_1000_FILE`
- `KEY_PER_1000_FILE` dienkripsi dengan RSA public key penyerang
- Tanpa RSA private key: secara teori tidak bisa dibuka

Bug: Keystream Reuse

- Footer untuk ~1000 file dienkripsi dengan **keystream yang sama**
- Ini adalah two-time pad (materi One-Time Pad)
- `C1 = P1 xor K`, `C2 = P2 xor K`
- Tahu `P2` (nama file panjang) -> dapat `K` -> buka `C1` (nama + key file pendek)

Satu keystream, dua pesan

Known plaintext membuka keystream di posisi yang sama. Ikuti pemulihan setiap byte.

C_1	79	26	52	9b	42	05
C_2	61	2d	5e	9e	49	70
P_2 diketahui	4b	4e	4f	57	4e	21
$K = C_2 \oplus P_2$	2a	63	11	c9	07	51
$P_1 = C_1 \oplus K$	53	45	43	52	45	54

S E C R E T

ZOOM BYTE 6 · XOR per bit

0 1 1 1 0 0 0 0 C_2

0 0 1 0 0 0 0 1 $\oplus$ P_2

0 1 0 1 0 0 0 1 = K

5 ⊕ 51 = 54 → T

09 **Plaintext “SECRET” berhasil dibuka**

Mengulang pasangan key dan nonce menghasilkan keystream yang sama. K yang unik memutus relasi ini.



A	Footer File 1	A	.	T	X	T	/	K	E	Y	1						
B	Footer File2	Y	O	H	A	N	E	S	.	T	X	T	/	K	E	Y	2
C	KEY_PER_1000_FILE	0x11	0x22	0x33	0x34	0x36	0x88	0x78	0x99	0xAA	0xAB	0xCD	0x1F	0xAA	0xBB	0xCC	0xDD
D	Enc(Footer File 1)	0x50	0x0c	0x67	0x6c	0x62	0xa7	0x33	0xdc	0xf3	0x9a						
E	Enc(Footer File 2)	0x48	0x6d	0x7b	0x75	0x78	0xcd	0x2b	0xb7	0xfe	0xf3	0x99	0x30	0xe1	0xfe	0x95	0xef
F	Enc(Footer File 2) XOR YOHANES.TXT/	0x11	0x22	0x33	0x34	0x36	0x88	0x78	0x99	0xAA	0xAB	0xCD	0x1F				
G	Enc(Footer File 1) XOR F	A	.	T	X	T	/	K	E	Y	1						

Pelajaran dari LockBit

- Library-nya (Salsa20) aman, penggunaannya yang salah
- Nonce / key stream cipher tidak boleh dipakai ulang
- Versi LockBit lama: key dari `RtlRandomEx` (LCG) -> bisa diprediksi
- RE tidak bisa "menemukan key", tapi bisa menemukan **bug** di ransomware

Babuk (ESXi): Dekriptor "Gratis"

- Brain Cipher merilis dekriptor gratis, ternyata untuk Babuk
- Pertanyaan: dekriptornya benar? ada backdoor?
- Source Babuk sudah bocor -> bandingkan binary dengan source
- Bedanya hanya ekstensi (`.encrptd`, aslinya `.babyk`) dan key
- Private key master ada di binary dekriptor (`0x0060c360`)

Skema Key Babuk: ECDH per File

```c
// enkriptor (hanya tahu m_publ)
csprng(u_priv, 32);                              // key pair ephemeral per file
curve25519_donna(u_publ, u_priv, basepoint);
curve25519_donna(u_secr, u_priv, m_publ);        // ECDH
sha256(u_secr) → sm_key → Sosemanuk
// ... enkripsi, lalu u_publ ditempel di 32 byte terakhir file

// dekriptor (tahu m_priv)
curve25519_donna(u_secr, m_priv, u_publ);        // shared secret yang sama
```

- Ini Diffie-Hellman di kurva eliptik, persis materi kuliah

Babuk Tanpa Key?

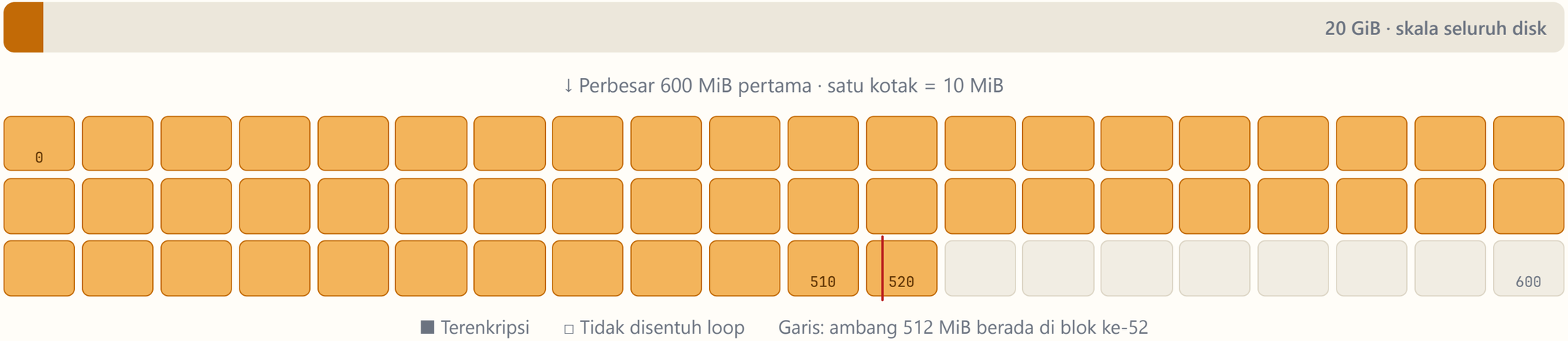
- Kriptografinya tidak ada bug: Curve25519 + SHA-256 + Sosemanuk
- Tapi lihat loop enkripsinya:

```c
do {
    wholeReaded += readed = fread(f_data, 1, CONST_BLOCK, fp);  // 10MB
    sosemanuk_encrypt(&rc, f_data, f_data, readed);
    ...
} while (wholeReaded < 0x20000000 && wholeReaded < fstat.st_size);
```

- Hanya **520MB pertama** yang dienkripsi, berapa pun ukuran VMDK

Enkripsi berhenti; data berlanjut

Animasi loop enkripsi Babuk: blok 10 MiB, ambang 512 MiB. Batas loop bukan tepat batas penulisan.



`520 < 512 → berhenti`

52 × 10 MiB = 520 MiB ≈ 2,54% dari 20 GiB. Persentase byte bukan persentase keberhasilan recovery.

06 **Recovery bergantung pada isi bagian awal**

Partisi dan metadata bisa rusak meski bagian besar disk tidak dienkripsi. Ini bukan jaminan persentase file yang pulih.

520MB Pertama Isinya Apa?

- VMDK thick provisioned (default ESXi) = disk mentah
- 520MB pertama biasanya: partition table, boot, sistem operasi
- Ubuntu / RHEL: partisi boot ~1.7GB di depan -> data utuh
- Debian (satu partisi): OS ~1.4GB di awal, data user di belakang
- Windows Server 2016 / 2019: partisi utama mulai di ~566MB -> NTFS utuh

Recovery Tanpa Key

```bash
# cari dan tulis ulang partition table yang tertimpa
testdisk debian-flat.vmdk.encrptd
# ext4 punya backup superblock
sudo kpartx -av debian-flat.vmdk.encrptd
sudo fsck.ext4 -y -b 163840 -B 4096 /dev/mapper/loop1p1
```

- Sulit: Windows Server 2022 (recovery partition dipindah ke belakang), thin provisioned, disk kedua dst, filesystem non-standard

Bonus: Bug di Dekriptor

- Dekriptor memotong 32 byte terakhir (`u_publ`) **sebelum** dekripsi selesai
- Dekripsi in-place, tidak ada catatan progress
- Terinterupsi (Ctrl-C, ssh putus, mati listrik) -> public key hilang, file setengah terdekrip
- Backup dulu, satu instance per direktori

Pelajaran dari PDN

- LockBit: kriptografi benar, **penggunaannya** salah (keystream reuse)
- Babuk: kriptografi benar, tapi sebagian besar data **tidak dienkripsi**
- Dekriptor dari penyerang: verifikasi, jangan langsung percaya
- Mengumumkan "data tidak bisa dipulihkan" terlalu cepat

CERITA 5

Akira Ransomware dan GPU (2025)

Sebelum AI cukup pintar

Konteks

- Sebuah perusahaan terkena Akira (varian Linux/ESXi), semua VM terenkripsi
- Pengecekan awal: key dibangkitkan dari **waktu saat itu (nanodetik)**
- Pikiran pertama: "Gampang, brute force saja dari timestamp file"
- Kenyataannya: hampir tiga minggu

2025

scworld.com 3 weeks ago

Akira ransomware decryption method uses GPUs to brute force keys

Source code for the decryption method was published by programmer Yohanes Nugroho.

bleepingcomputer.com 4 weeks ago

New Akira ransomware decryptor cracks encryptions keys using GPUs

Security researcher Yohanes Nugroho has released a decryptor for the Linux variant of Akira ransomware, which utilizes GPU power to retrieve...

mashable.com 3 weeks ago

Devastating Akira ransomware cracked in hours using cloud computing

Akira, one of the most dangerous ransomware strains floating around the internet, just met its match — an Indonesian programmer armed with...

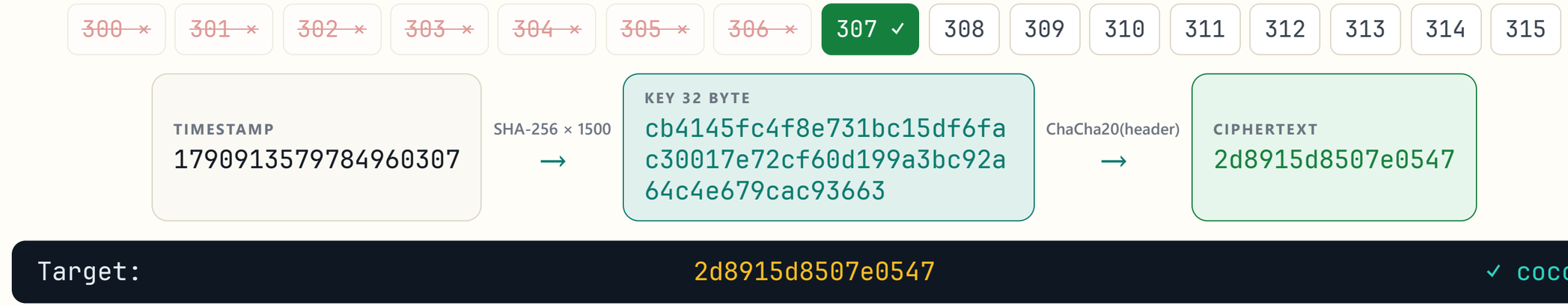
Bagaimana Key Dibangkitkan

```c
void generate_random(char *buffer, int size) {
    uint64_t t = get_current_time_nanosecond();
    char seed[32];
    snprintf(seed, sizeof(seed), "%lld", t);
    struct yarrow256_ctx ctx;
    yarrow256_init(&ctx, 0, NULL);
    yarrow256_seed(&ctx, strlen(seed), seed);   // 1500 ronde SHA-256
    yarrow256_random(&ctx, size, buffer);
}
```

- Dipanggil 4 kali: key + nonce ChaCha8, key + IV KCipher2
- Entropi efektif: hanya sebanyak ketidakpastian **waktu**

Key panjang tidak menambah entropi seed

Key 256 bit, tapi dihitung dari timestamp. Jumlah key yang mungkin = jumlah timestamp yang mungkin: 1 detik = 10^9 nanodetik ≈ 2^{30}, bukan 2^{256}. Trace disederhanakan dari demo Python, bukan Yarrow/KCipher2 Akira asli.



$$\log_2(10^9) \approx 29{,}9 \text{ bit}$$

Biaya tiap tebakan dan hubungan empat timestamp membuat serangan asli jauh lebih mahal.

07 **Panjang key tidak memperbesar ruang seed**

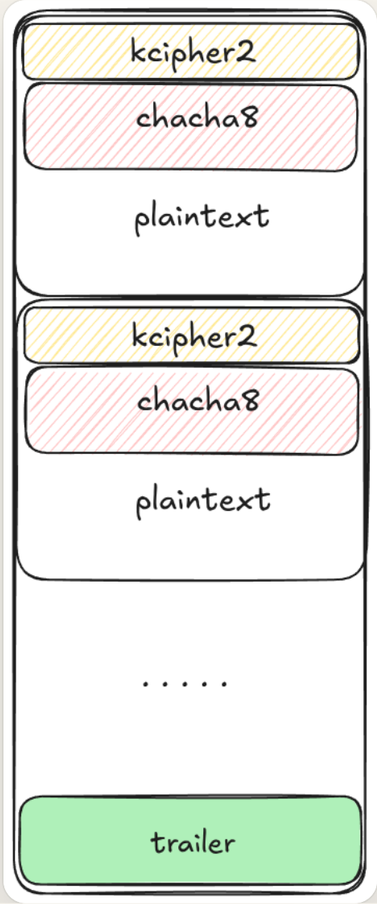
Jendela 1 detik memberi 10^9 kandidat nanodetik ≈ 2^{30}, meskipun key keluaran 256 bit.

Kenapa Tidak Semudah Itu

- 4 timestamp, bukan 1, dan saling berkaitan
- Tiap kandidat: 1500 ronde SHA-256
- Filesystem VMFS hanya menyimpan waktu dengan resolusi **detik**
- Ransomware multithread, modification time = waktu file **ditutup**
- Jarak antar timestamp bervariasi 1.5 - 5 juta nanodetik

Known Plaintext

- Brute force butuh plaintext yang diketahui
- File `sesparse` VMware selalu diawali `00 00 00 00 ca fe ba be`
- Flat VMDK: install OS yang sama, ambil 8 byte pertama
- 8 byte pertama dienkrip KCipher2 -> cukup untuk mengecek kandidat



Butuh ketelitian

- Implementasi KCipher2 publik **tidak cocok** dengan hasil malware
- Ternyata malware memodifikasi sedikit IV dan endianness
- ChaCha8, bukan ChaCha20 seperti yang ditulis banyak laporan
- KCipher2 mengenkrip 65535 byte (bukan 65536)

GPU

- Kode CUDA awal: dibantu ChatGPT (o1) -> compile, tapi **hasilnya salah**
- Penyebab: AI mengubah angka di tabel konstanta
- Optimasi manual: shared memory, matching di GPU, batch sync
- RTX 3090: 60 juta -> 1.5 milyar enkripsi KCipher2 per detik



Hasilnya

- RTX 4090 sewaan (Runpod, ~0.69 USD/jam): 2.3x lebih cepat dari 3090
- Satu detik waktu enkripsi, 16 GPU: sekitar 10 jam
- Total biaya eksperimen: sekitar 1200 USD
- Seluruh VM berhasil dipulihkan tanpa membayar tebusan
- Kode: github.com/yohanes/akira-bruteforce

Demo: Brute Force Seed Waktu

PYTHON

```python
def key_from_time(t_ns):
    seed = str(t_ns).encode()
    v0 = d = hashlib.sha256(seed).digest()
    for i in range(1, 1500):
        d = hashlib.sha256(d + v0 + i.to_bytes(4, "big")).digest()
    return d

for t in range(start, start + window):
    if chacha(key_from_time(t), known) == ciphertext[:8]:
        print("ketemu!", t)
```

OUTPUT

```
ketemu! t=17909135797849603O7 (asli 17909135797849603O7), 6899 percobaan, 4.8 detik
```


Here to make friends
@InternetHippo
Congratulations on graduating and welcome to the workforce. Choose your job:
Evil
Boring
8:51 PM · 9/22/22 · Twitter Web App
2,446 Retweets 194 Quote Tweets 33.2K Likes

Rangkuman Cerita

Kasus	Kesalahan	Materi kuliah
Bank 2015	`hashCode()` sebagai MAC	Fungsi hash, MAC
MIGS 2017	Concat ambigu, tanpa encoding	Hash, MAC, signature
PGP 2018	(aman) AES-CCM, key di OTP	Mode operasi block cipher
PDN 2024: LockBit	Keystream reuse	One-Time Pad, stream cipher
PDN 2024: Babuk	(aman) tapi hanya 520MB dienkripsi	ECDH, Curve25519
Akira 2025	Seed dari waktu	PRNG, known plaintext attack

BAGIAN 7

Disrupsi AI

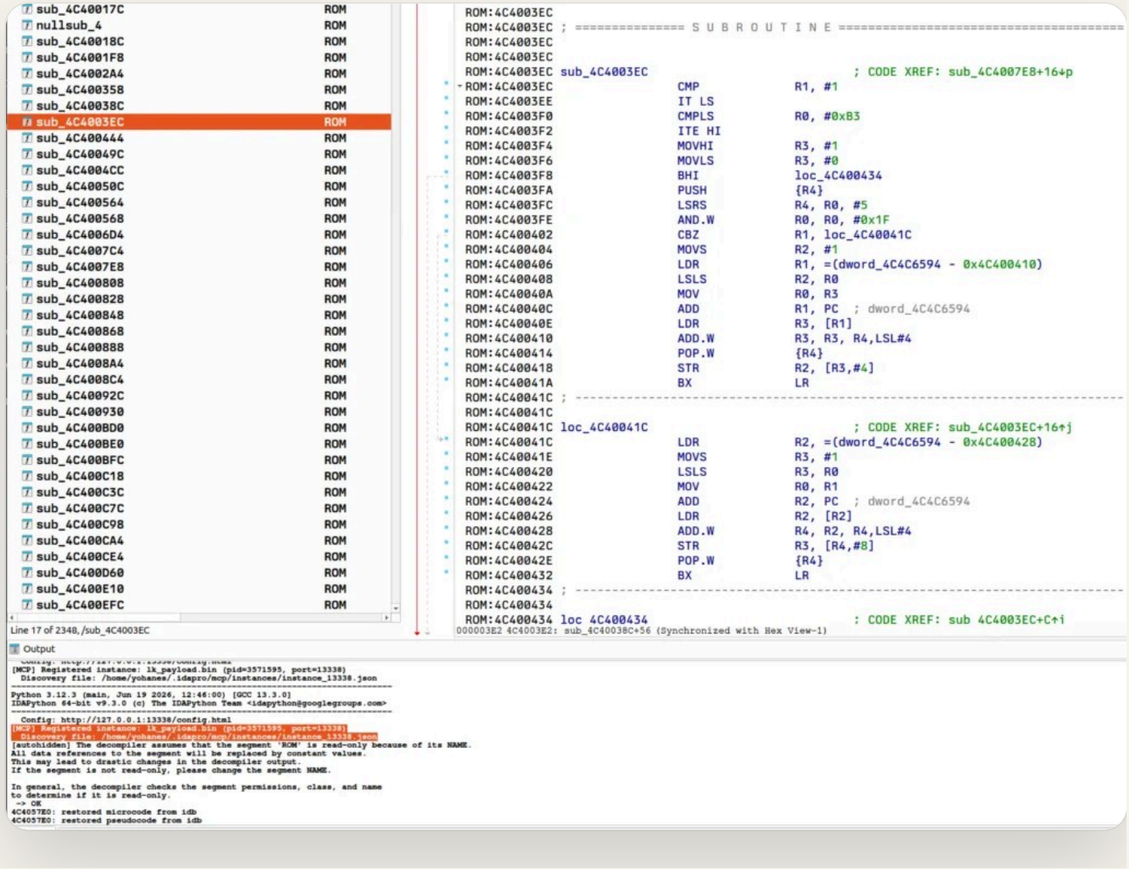
AI semakin pintar, apa peran kita?

Yang Saya Lakukan Tadi, Sekarang Bisa Dilakukan AI

- Membaca kode dan langsung melihat `hashCode()` sebagai MAC
- Mengenali konstruksi standard (CTR + CBC-MAC = CCM)
- Mengendalikan IDA, Ghidra, radare2 lewat MCP
- Live debugging dengan debugger, Frida, adb, ssh
- Menulis decryptor dan brute forcer

Contoh: IDA Pro dengan MCP

- Agent membuka binary, rename fungsi, dekompilasi, menjelaskan algoritma
- Manusia: memberi tujuan dan memeriksa hasil



2-3 Tahun Lalu vs Sekarang

2-3 tahun lalu	Sekarang
Menjawab pertanyaan dasar RE	RE dan reimplementasi game lama
Dekompilasi fungsi kecil	Hampir semua soal CTF RE
Membantu menulis script Frida	RE app mobile dan binary tanpa obfuscation
	Debugging dan tracing langsung di device

Yang Masih Sulit untuk AI

- **RASP** (Runtime Application Self-Protection), misalnya di app banking
- **Obfuscation berat**: masalah graph yang kompleks, butuh tool khusus
- Kode yang diload saat runtime
- Trik anti-RE yang mengecoh
- Bisa saja dengan token dan waktu yang cukup, tapi **mahal**

Guardrail

- Model akan menolak permintaan yang terlihat ilegal
- Contoh: "bypass RASP aplikasi bank ini" -> sering ditolak
- Akun bisa di-ban; untuk pekerjaan cyber perlu verifikasi
- Pekerjaan legal pun kadang ditolak, perlu dipecah langkah demi langkah

BAGIAN 8

Peran Manusia di Zaman AI

Beberapa tahun ke depan, manusia masih dibutuhkan

AI Perlu Diarahkan

- Tentukan **tujuan** RE dengan jelas
- Siapkan tool dan konteks: device yang sudah di-root, hasil `jadx` / `apktool`
- Bantu di titik yang AI tidak bisa: KYC, 2FA, PIN, layar yang diblok screenshot
- Saat AI "menyerah", dorong dengan pertanyaan yang tepat

OUTPUT

```
reverse engineer this app, look at the "check balance" function,
create a python script so i can do it via CLI
```

Contoh: AI Menyerah Terlalu Cepat

- AI: "data dari browser capture tidak bisa didekrip, *ephemeral session key* sudah hilang"
- Saya: "jelaskan bagaimana browser membuat key ini? key agreement? bisa inject JS untuk menangkapnya?"
- AI menyelidiki lagi: bukan Diffie-Hellman, tapi RSA ephemeral dari WASM + session AES
- Hanya bisa diarahkan jika kita paham istilahnya

Hasil AI Tetap Perlu Dicek

- "Enkripsinya AES" -> benar AES? atau varian (seperti KCipher2 di Akira)?
- "Key-nya statik" -> coba besok, ternyata berdasarkan waktu / hari
- "Semua test sudah lolos" -> dijalankan, ternyata tidak
- Script AI tidak mengirim header yang dianggap "tidak penting" -> akun di-ban
- AI mengubah angka di tabel konstanta (CUDA Akira)

Cara Memverifikasi

- Reimplementasi, lalu bandingkan dengan **test vector** atau output asli
- Bandingkan dengan implementasi standard (seperti demo CCM tadi)
- Ulangi di waktu / device / akun berbeda
- Minta AI menjelaskan **kenapa**, lalu cek klaimnya ke kode
- Jalankan sendiri perintah yang AI bilang "sudah berhasil"


From comics community on Reddit
I'M IN... I HACKED INTO THE SERVER...
HACKER
MOMENTS EARLIER
SOME GUY OVER AT I.T. NEEDS MY PASSWORD...
SOUNDS A LITTLE FISHY...
WAIT A SECOND... ....I DON'T GIVE A SHIT ABOUT THIS COMPANY
SENT

BAGIAN 9

Pekerjaan Saya Sekarang

Adaptasi pekerjaan di era AI

Training RE dengan AI

- RE dengan AI terlihat mudah, tapi kebanyakan orang belum bisa
- Saya memberikan training RE dengan AI ke beberapa bank
- Isi: setup tool, cara bertanya, kapan harus intervensi, cara memverifikasi
- Target kompleks tetap butuh pemahaman dasar yang dalam

Testing KYC

- KYC: face recognition, liveness detection, foto KTP
- Butuh **manusia**: wajah asli, gerakan, device fisik
- Pertanyaan yang dites:
  - Bisakah gambar kamera diganti / diinjeksi?
  - Apakah payload dari SDK dienkripsi dan di-sign? Bisa di-replay?
  - Apakah server memverifikasi, atau percaya hasil dari client?

Testing AI itu sendiri

- Attack terhadap pipeline AI
- Attack terhadap embedding (bagaimana supaya dokumen yang sangat menyesatkan bisa dianggap paling relevan)
- Attack terhadap model yang disertakan aplikasi (misalnya model face recognition)

Testing Proteksi Mobile

- Tujuan: membuat RE "murah" dengan AI menjadi **mahal**
- RASP: deteksi root, hook, Frida, app yang dimodifikasi
- Obfuscation dan proteksi anti-AI mulai dikembangkan
- Selalu bisa ditembus, pertanyaannya: berapa biayanya untuk attacker?

Contoh: RASP dan `seccomp`

- RASP membaca APK sendiri untuk mengecek signature
- Memakai **syscall langsung**, bukan libc, supaya tidak mudah di-hook
- Malware (seperti SnowBlind) memakai `seccomp` untuk mengintersepsi syscall itu
- Syscall `openat("base.apk")` dialihkan ke APK asli

`seccomp` Filter (BPF)

```c
static struct sock_filter filter[] = {
  BPF_STMT(BPF_LD | BPF_W | BPF_ABS, offsetof(struct seccomp_data, arch)),
  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, AUDIT_ARCH_AARCH64, 1, 0),
  BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL),
  BPF_STMT(BPF_LD | BPF_W | BPF_ABS, offsetof(struct seccomp_data, nr)),
  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, __NR_openat, 0, 2),
  BPF_STMT(BPF_LD | BPF_W | BPF_ABS, offsetof(struct seccomp_data, args[4])),
  BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, SECMAGIC, 0, 1),
  BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
  BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_TRAP),   // → SIGSYS handler
};
```

BAGIAN 10

Penutup

Sekedar tips dari saya

Tips Belajar: Genetic Method

- Otto Toeplitz (1927): ajarkan konsep dengan mengikuti **masalah** yang melahirkannya
- Jangan hanya hafal solusi, pahami kenapa solusinya begitu
- Contoh:
  - Kenapa HMAC, bukan `hash(key + data)`? -> length extension
  - Kenapa AEAD (CCM, GCM)? -> enkripsi tanpa autentikasi bisa diubah
  - Kenapa nonce tidak boleh diulang? -> two-time pad

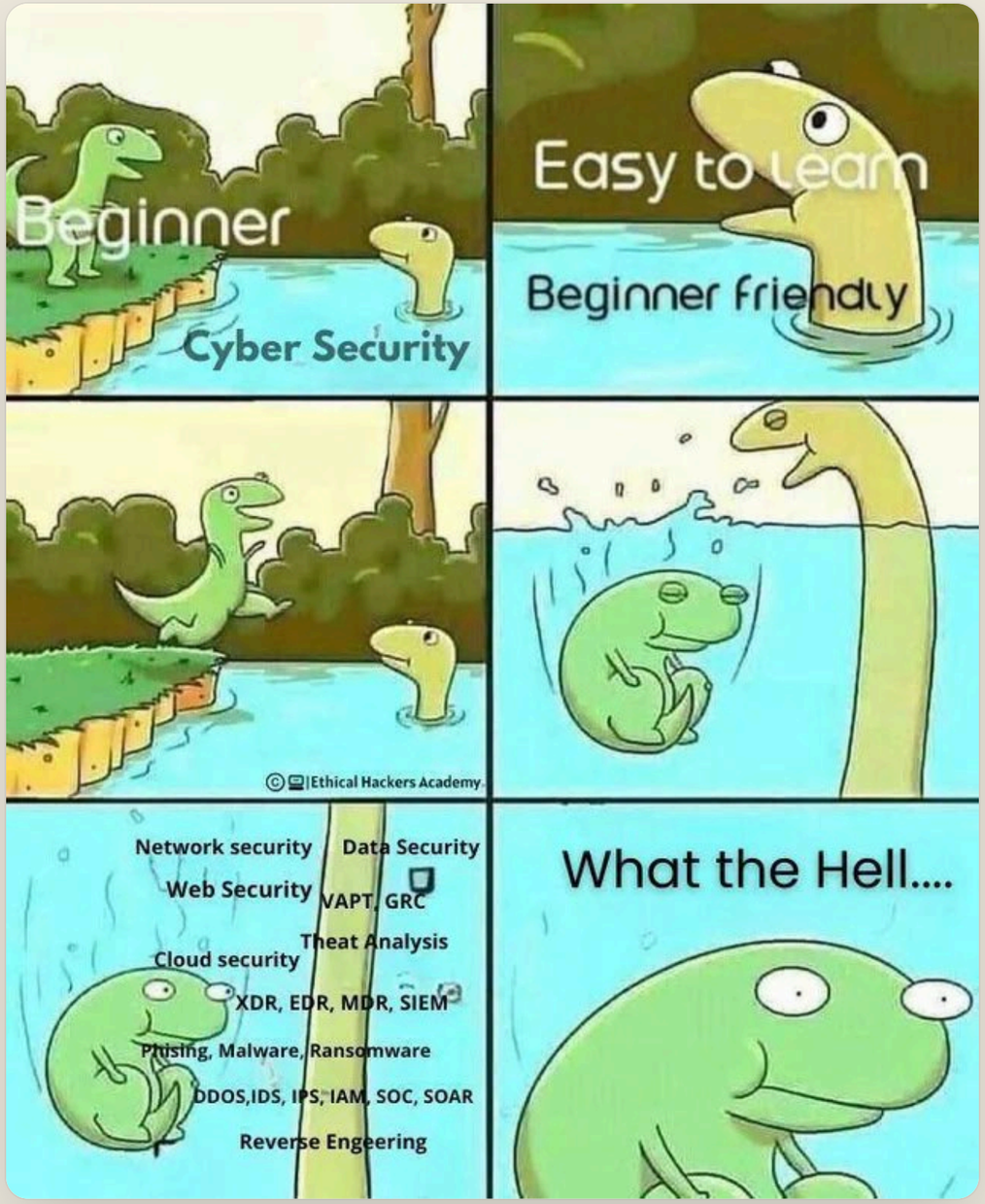

Sudah kubilang
aku ga pinter
Tapi kamu kan itb

Contoh: Kerberos

- Kerberos terlihat rumit: ticket, TGT, KDC, authenticator, timestamp
- Baca: *Designing an Authentication System: a Dialogue in Four Scenes*
- Athena dan Euripides merancang ulang Kerberos dari nol
- Setiap komponen muncul karena serangan pada desain sebelumnya

web.mit.edu/kerberos/dialogue.html


SwiftOnSecurity
@SwiftOnSecurity
Subscribe
One time I tried to explain Kerberos to someone.
Then we both didn't understand it.
1:00 PM · Nov 21, 2014
26
445
932
17


Beginner
Cyber Security
Easy to Learn
Beginner friendly
©|Ethical Hackers Academy.
Network security
Data Security
Web Security
VAPT, GRC
Theat Analysis
Cloud security
XDR, EDR, MDR, SIEM
Phising, Malware, Ransomware
DDOS,IDS, IPS, IAM, SOC, SOAR
Reverse Engeering
What the Hell....


"If you only do what you can do,
you'll never be better than
what you are." - Shifu

Rangkuman

- Bug kriptografi di dunia nyata hampir selalu di **cara pakai**, bukan algoritmanya
- Kriptografi dan RE berjalan bersama
- AI sudah bisa mengerjakan sebagian besar RE, kecuali proteksi berat
- Manusia: mengarahkan AI dan memverifikasi hasilnya
- Untuk itu, pemahaman dasar kriptografi justru makin penting

Referensi

- Mastercard Internet Gateway Service: Hashing Design Flaw
- Reverse Engineering Pokemon GO Plus
- Ransomware PDN: LockBit dan Babuk (tag pdn)
- eMudhra: Private key compromise of Intermediate Certificates (Mozilla Bugzilla)
- Compromised eMudhra intermediate certificates (Cisco Talos)
- Decrypting Akira Ransomware (Linux/ESXi) Using a Bunch of GPUs
- Reverse Engineering dengan AI
- Seccomp untuk Intersepsi Syscall di Linux dan Android
- Designing an Authentication System: a Dialogue in Four Scenes

TERIMA KASIH

Pertanyaan?

↗ tinyhack.com

↗ blog.compactbyte.com