YesWeHack

#YWH-PGM2123-748 NEW

/ Yeswehack: Dojo #29 : Writeup by alt3kx (2023) □□

YESWEHACK DOJO

SUBMITTED BY XK3TLA ON 2023-12-26

REPORT DETAILS

BUG TYPE	Use of Hard-coded Cryptographic Key (CWE-321)
SCOPE	https://dojo-yeswehack.com/Playground
ENDPOINT	DOJO #29
SEVERITY	Critical
VULNERABLE PART	others
PART NAME	source code / java script / har coded keys
PAYLOAD	$ openssl enc -d -base64 -aes-256-cbc -md md5 -pass pass:'[key_exposed_here]'
TECHNICAL ENVIRONMENT	DOJO #29
APPLICATION FINGERPRINT	Christmas XMAS Webserver v 25.DEC.23
IP USED	127.0.0.1

CVSS SCORE	SEVERITY
9.8	CRITICAL

VECTOR STRING

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

DOCUMENTS

/0.png

/2.png

/3.png

/4.png

/5.png

BUG DESCRIPTION

/ Yeswehack: Dojo #29 : Writeup by alt3kx (2023) □□

/ Hard-Coded Cryptographic Keys (JavaCode|MD5) | CVSS:3.1 9.8 | CWE-321 | A02:2021-Cryptographic Failures

Description

Hard-coded cryptographic keys refer to encryption keys that are embedded directly into a software , web application or device. These keys are often used to secure sensitive information or communications, but their hard-coded nature makes them vulnerable to exploitation by attackers who can easily access and use them to decrypt data.

Impact

An attacker who gains access to an application or device that uses hard-coded keys can easily use those keys to decrypt any sensitive data that is stored or transmitted using the application or device. For example, if a mobile app uses a hard-coded encryption key to protect user data, an attacker who reverse-engineers the app can extract the key and use it to steal the user's personal information.

Mostly, the result is highly devastating for the target such as:

- / Deprecated hash functions such as MD5 or SHA1 in use
- / Non-cryptographic hash functions used when cryptographic hash functions are needed
- / Stealing data
- / Man-in-the-middle attacks
- / Malicious software

Steps to Reproduce

(1) Start a recon trough the source code and observe the following code snippet: Ref: 2.png

```
var output = document.getElementById("output_vault")

// Create a new vault (backup vault) to store our data in.
// When we have our vault, set a secret key that will lock the vault (I have to use the MD5 hash of the key because we f
var vault = new Vault()
vault.setKey("7f16e4fc3d6c9bd92de59e9369891dba")
vault.lock()

// We only got the MD5 hash value of the key.
// I'm going to move this AES encrypted data from my other vault that uses the same key, so I'll add it as its AES encry
vault.setData("U2FsdGVkX1/SpGvO7gL9H5GgEjLvEUhKcB9yK4sEC/JTVqUwyPZJrD9g84JkXuJurQY6naZ9KOFKXsll1jqWO10i3CVyZOcyDCi+3anUu

// Let's take a guess!
const key = "$KEY"

// Check if we were able to unlock the vault with our guessed key:
error = vault.unlock(key)
if ( error == null ) {
    let data = vault.open()
    output.innerHTML = `<code>${data}</code>`
} else {
    output.innerHTML = `<b style="color:red"> ${error} </b>`
}
</script>
```

(2) According to the logic of code provided:

- a) $const key = "$KEY" | It's a input value provide by end users and store it into $KEY variable and it will validated to unlock the vault!
- b) vault.setKey | The string contain a MD5 exposed trough source code (The key to decrypt data).
- c) vault.setData | The string contain the cipher data.

(3) Extract those important values (Hard-coded-keys) and decrypt the data using OpenSSL as follow: Ref: 3.png

```
$ echo
"U2FsdGVkX1/SpGvO7gL9H5GgEjLvEUhKcB9yK4sEC/JTVqUwyPZJrD9g84JkXuJurQY6naZ9KOFKXsll1jqWO10i3CVyZOcyDCi+3anUuqawcsErq+k5SC5ExWd7S59ANl/kcMDQf6cHI7s+hG3Z6als7+whFD+2fHHo7FJOjSu7KCTfBZlOhfT2
d1kjV/gIEN4EFFOAU2BuRwEOYbjSQ7G0kFX3Bws4oupyFLZiiTE5y2YFVgt8KExnSk2hQOCZbNRmOQsvlY+Z9Jbtp2XtduIyRbJRPhDrG9Bb8UFwdAmroaa0aaLfobHbRrvo2fd3b2VA+AGIdyBYPYgpCL0DL8b/kbL9Usleseb+A2rFr0lvFEisfHI
me4N4T3zwg+QhDAOrCBxhoOO2yeL8FaEGxT2DGJnyDB63Dl6zDiFzydTI4jxZ8E7rOA3iwXsccCI5lZE6Ljid+bPCvoTva2SGexkm8Yn1E6LTEG87zqb+xTawhotU+YzmUOHvnEg9YrRg0VVtx8GX0EI+lzSV04ughDa+lqiCNcdkePOlMsM9Z
nXvwjxOOUTrJsQ+00DXRZuX8FK/SaeTJSJNRAYDnrpSscMxUZ5S3gg0SoMN0+Ngb3yMxI+AOFOU/+kJ2egI9hbJnh7aDvX8RXd/0DDcj5YbQgolfPRNK7vR0/gLHw728uJfy3dsUhiDYNemui5BP/9Tl3uG5xTWP7ncPaPnymGYGUiW61yilIR4iF
qN2feSFpsoJNFQClu/EJU+bPvuehy1onA8UR+zW9idOJCd5VzSY4e7PjsNduWWMVI45SGrJzxKZj7avedoAOFKwcdZqQ4BNUBRmW0UzEjhBiXx3lZZ7OjzXjp5dp13pDrn5MRVA+bLD95EK0JH8wJD32E22KNjXw4SfWx8+0RF5UKNXqTJAQT
67bVCJS11Z3fbc=" | openssl enc -d -base64 -aes-256-cbc -md md5 -pass pass:'7f16e4fc3d6c9bd92de59e9369891dba'
```

```
~/Downloads/CTF_yeswehack/chall29                                    at 05:28:36 AM
echo "U2FsdGVkX1/SpGvO7gL9H5GgEjLvEUhKcB9yK4sEC/JTVqUwyPZJrD9g84JkXuJurQY6naZ9KOFKXsll1jqWO10i3CVyZOcyDCi+3anUuqa
wcsErq+k5SC5ExWd7S59ANl/kcMDQf6cHI7s+hG3Z6als7+whFD+2fHHo7FJOjSu7KCTfBZlOhfT2nd1kjV/gIEN4EFFOAU2BuRwEOYbjSQ7G0kFX3Bw
s4oupyFLZiiTE5y2YFVgt8KExnSk2hQOCZbNRmOQsvlY+Z9Jbtp2XtduIyRbJRPhDrG9Bb8UFwdAmroaa0aaLfobHbRrvo2fd3b2VA+AGIdyBYPYgpCL
0DL8b/kbL9Usleseb+A2rFr0lvFEisfHRvme4N4T3zwg+QhDAOrCBxhoOO2yeL8FaEGxT2DGJnyDB63Dl6zDiFzydTI4jxZ8E7rOA3iwXsccCI5lZE6L
Jid+bPCvoTva2SGexkm8Yn1E6LTEG87zqb+xTawhotU+YzmUOHvnEg9YrRg0VVtx8GX0EI+lzSV04ughDa+lqiCNcdkePOlMsM9ZqnXvwjxOOUTrJsQ+
00DXRZuX8FK/SaeTJSJNRAYDnrpSscMxUZ5S3gg0SoMN0+Ngb3yMxI+AOFOU/+kJ2egI9hbJnh7aDvX8RXd/0DDcj5YbQgolfPRNK7vR0/gLHw728uJf
y3dsUhiDYNemui5BP/9Tl3uG5xTWP7ncPaPnymGYGUiW61yilIR4iPFmqN2feSFpsoJNFQClu/EJU+bPvuehy1onA8UR+zW9idOJCd5VzSY4e7PjsNdu
WWMVI45SGrJzxKZj7avedoAOFKwcdZqQ4BNUBRmW0UzEjhBiXx3lZZ7OjzXjp5dp13pDrn5MRVA+bLD95EK0JH8wJD32E22KNjXw4SfWx8+0RF5UKNXq
TJAQTe67bVCJS11Z3fbc=" | openssl enc -d -base64 -aes-256-cbc -md md5 -pass pass:'7f16e4fc3d6c9bd92de59e9369891dba'
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
function chiperXOR(v, pincode) {
    if ( pincode.match(/^[0-9]{4}$/) === null ) {
        return Error("The PIN code must be exactly 4 digits long and may only contain digits from 0-9.");
    }
    /* Perform an XOR operation from they provided key for each character in the given input (value) */
    v = v.split("");
    for (let i = 0; i < v.length; i++) {
        v[i] = (String.fromCharCode((v[i].charCodeAt(0)) ^ pincode.charCodeAt(0)));
    }
    /* Combine all the characters to a string and base64 encode the new encrypted value. Then return the value: */
    return btoa( v.join("") )
}
/* Decrypt me (format: "FLAG{...}") => dX9ydEhnWwBsfQBEbHBBSkNHA2xeBxdHAEFO */%
```

(4) Notice the decrypted data now is a new code snippet revealed as follow:

```
function chiperXOR(v, pincode) {
if ( pincode.match(/^[0-9]{4}$/) === null ) {
return Error("The PIN code must be exactly 4 digits long and may only contain digits from 0-9.");
}
/* Perform an XOR operation from they provided key for each character in the given input (value) */
v = v.split("");
for (let i = 0; i < v.length; i++) {
v[i] = (String.fromCharCode((v[i].charCodeAt(0)) ^ pincode.charCodeAt(0)));
}
/* Combine all the characters to a string and base64 encode the new encrypted value. Then return the value: */
return btoa( v.join("") )
}
/* Decrypt me (format: "FLAG{...}") => dX9ydEhnWwBsfQBEbHBBSkNHA2xeBxdHAEFO */%
```

(5) According to the logic of code provided:

- a) To decrypt the data must be provided a PIN code 4 digits 0000.
- b) The XOR operation to cipher the string will be using the PIN for for each character provided.
- c) The final part once cipher the data will be encoded the string with with base64.

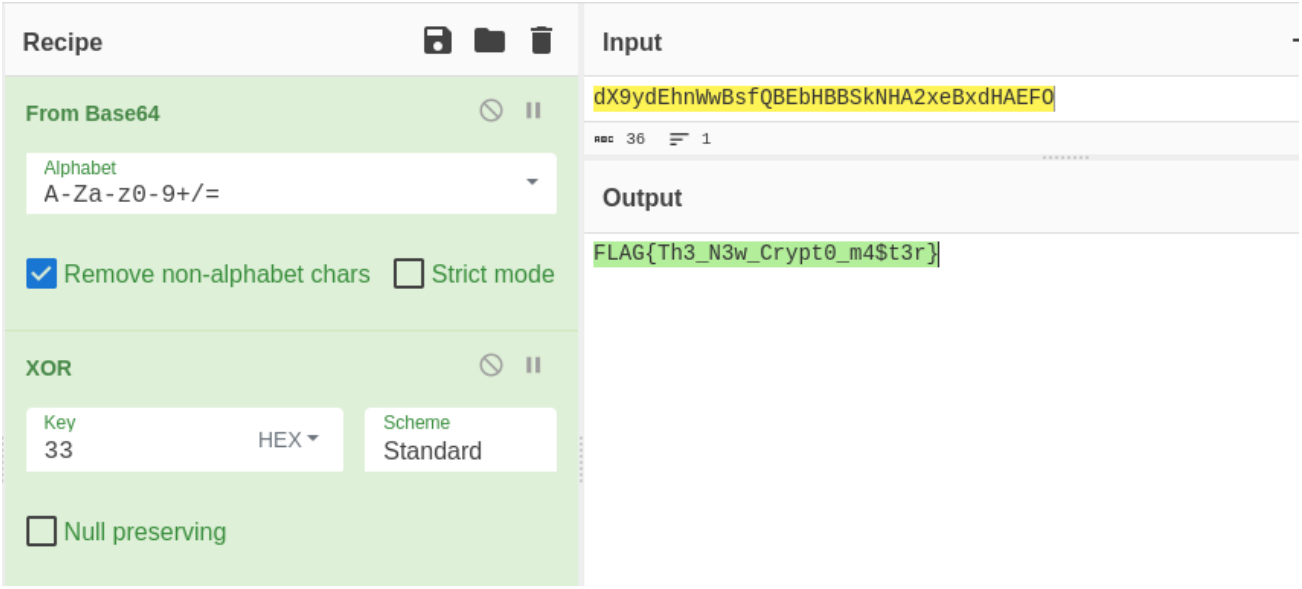
(6) Extract those important values and decrypt the data using xortool/on-liners tools as follow: Ref: 4.png

```
$ cat msg.enc
dX9ydEhnWwBsfQBEbHBBSkNHA2xeBxdHAEFO

$ for i in {3330..3340}; do cat msg.enc | base64 -d | xortool-xor -r "$i" --no-cycle -f - ; echo "\033[0;36m [+] YesWehack Chall29 2023 by alt3kx \033[m | PIN: \033[33m$i\033[m" ; done

[+] YesWehack Chall29 2023 by alt3kx | PIN: 3332 <--HERE! string FLAG obtained cipher key = 33
FLAGHg[l}DlpAJCGl^GAN

[../snip]

$ cat msg.enc | base64 -d | xortool-xor --no-cycle -r 3333 -f -
FLAGHg[l}DlpAJCGl^GAN
```

```
~/Downloads/CTF_yeswehack/chall29 ✔ at 04:24:19 AM
cat msg.enc
dX9ydEhnWwBsfQBEbHBBSkNHA2xeBxdHAEFO

~/Downloads/CTF_yeswehack/chall29 ✔ at 04:24:24 AM
for i in {3330..3340}; do cat msg.enc | base64 -d | xortool-xor -r "$i" --no-cycle -f - ;  echo "\033[0;36m [+] YesWehack Chall29 2023 by alt3kx \033[m | PIN: \033[33m$i\033[m"  ; done
FLADHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3330
FLAEHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3331
FLAFHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3332
FLAGHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3333
FLA@Hg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3334
FLAAHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3335
FLABHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3336
FLACHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3337
FLALHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3338
FLAMHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3339
FLFDHg[l}DlpAJCGl^GAN

 [+] YesWehack Chall29 2023 by alt3kx  | PIN: 3340

~/Downloads/CTF_yeswehack/chall29 ✔ took 4s ⌛ at 04:24:45 AM
cat msg.enc | base64 -d | xortool-xor --no-cycle -r 3333 -f -
FLAGHg[l}DlpAJCGl^GAN

~/Downloads/CTF_yeswehack/chall29 ✔ at 04:24:53 AM
```

(7) Enhancement using a Cyberchef recipe as follow:: Ref: 5.png

https://gchq.github.io/CyberChef/#recipe=From_Base64('A-Za-z0-9%2B/%3D',true,false)XOR(%7B'option':'Hex','string':'33'%7D,'Standard',false)&input=ZFg5eWRFaG5Xd0JzZlFCRWJIQkJTa05lQTJ4ZUJ4ZEhBRUZP



Flag

FLAG{Th3_N3w_Crypt0_m4$t3r}

Recommendations

Issue Domain : Developers, Software Architects & Administrators

To avoid the attack

- Use secure coding practices: Developers should avoid hard-coding cryptographic keys in their code whenever possible. Instead, keys should be generated dynamically and stored securely.
- Follow the principle of least privilege: Users and applications should only be given the minimum level of access necessary to perform their tasks. This can help prevent unauthorized access to sensitive information.
- Regularly update software and firmware: Updates can often include security patches that address vulnerabilities, including those related to hard-coded cryptographic keys.
- Use encryption and secure communication protocols: Encryption can help protect sensitive data in transit and at rest. Secure communication protocols, such as HTTPS, can help prevent man-in-the-middle attacks.
- Conduct regular security assessments and penetration testing: Regular testing can help identify vulnerabilities, including hard-coded cryptographic keys, so that they can be addressed before they are exploited.
- Use strong and unique passwords: Passwords should be complex and unique for each account. This can help prevent unauthorized access even if a hard-coded cryptographic key is compromised.
- Implement multi-factor authentication: Multi-factor authentication adds an extra layer of security to the authentication process, making it more difficult for attackers to gain unauthorized access to accounts or systems.

References

https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
https://cqr.company/web-vulnerabilities/hard-coded-cryptographic-keys/

Contact

https://twitter.com/*alt3kx*
https://alt3kx.github.io/
https://infosec.exchange/@alt3kx

COMMENTS

XK3TLA ON 2023-12-26 07:53:08

NEW