Project No.7
ElGemal

INSTRUCTIONS FOR USE

ElGemal

A program that encrypts and then decrypts a string between two entities using the ELGemal algorithm.

Jadx

Analysis of an Android program by means of its reverse engineering

OWASP

Research on OWASP Top 10 2021

Made by:

Hoorieh Sabzevari - 98412004

Elnaz Rezaee - 98411387

1

ElGemal

Let’s start with the first set of slides

In this section, we have written a program that plays the role of a user.

Steps

- Key Generation
- Request for Other Party's Public Key
- Communication Establishment
- Message Encryption
- Message Decryption

First part

main function

Main function

Main

```
def main():
    q = random.randint(pow(10, 20), pow(10, 50))
    g = random.randint(2, q)
    key = gen_key(q)
    h = power(g, key, q)
    print("user g : ", g)
    print("user h : ", h)
    print("user q : ", q)
    print("_____")
    other_g=int(input("enter other side g:"))
    other_h=int(input("enter other side h:"))
    other_q=int(input("enter other side q:"))
    a="initial"
    while(a !="exit"):
        a = input("*******************************\nThis Is User A \nsend message : 1 \ndecode recived message : 2 \nexit : exit \n")
        if(a=='1'):
            message=input("enter message:")
            encrypted_message, p = encrypt(message, other_q, other_h, other_g)
            print("encrypted message :"+str(encrypted_message)+"\n"+"p:"+str(p)+"\n")
        elif(a=='2'):
            message=input("enter encrypted message:")
            message = message.strip('][').split(',')
            for i in range(len(message)):
                message[i]=int(message[i])
            p=int(input("enter p :"))
            decrypted_message = decrypt(message, p, key, q)
            dmsg = ''.join(decrypted_message)
            print("decrypted message :", dmsg);
        elif(a!="exit"):
            print("wrong input!")
```

Gcd and gen_key functions

Recursive GCD Calculation using Euclidean Algorithm

```
[3]  def gcd(a, b):
         if a < b:
             return gcd(b, a)
         elif a % b == 0:
             return b;
         else:
             return gcd(b, a % b)
```

Random Key Generation for Cryptographic Applications

```
[4]  def gen_key(q):
         key = random.randint(pow(10, 20), q)
         while gcd(q, key) != 1:
             key = random.randint(pow(10, 20), q)
         return key
```

Modular power function

Exponentiation using Binary Exponentiation Algorithm

```
def power(a, b, c):
    x = 1
    y = a
    while b > 0:
        if b % 2 == 0:
            x = (x * y) % c;
        y = (y * y) % c
        b = int(b / 2)
    return x % c
```

Encrypt and decrypt functions

Encryption Function for Secure Message Transmission

```
def encrypt(message, q, h, g):
    encrypted_message = []
    k = gen_key(q)
    s = power(h, k, q)
    p = power(g, k, q)
    for i in range(0, len(message)):
        encrypted_message.append(message[i])
    for i in range(0, len(encrypted_message)):
        encrypted_message[i] = s * ord(encrypted_message[i])
    return encrypted_message, p
```

Decryption Function for Secure Message Retrieval

```
[7]  def decrypt(encrypted_message, p, key, q):
        decrypted_message = []
        h = power(p, key, q)
        for i in range(0, len(encrypted_message)):
            decrypted_message.append(chr(int(encrypted_message[i]/h)))
        return decrypted_message
```

Testing

```
if __name__ == '__main__':
    main()
```

```
user g :  388195933700995998161677617504736849441526245879 2
user h :  263237683536270803464676077270943021339396785631 58
user q :  462915700056843266720131592503253558778699661675 35
_____
enter other side g:109446188314575224046196982155434284911552432512 04
enter other side h:116411933823313027392065220362022848424256657613 66
enter other side q:120343579691614703897269628901085759756284834170 77
**********************************
This Is User 1
send message : 1
decode recived message : 2
exit : exit
2
enter encrypted message:[361071810552606457876451606720914112018811255608693 7, 439376540551966894524356774443521991733734178511783 9, 430676014996482401563478422474343338432076075966996
enter p :308133805320343393817525203483588493879403292351 89
decrypted message : Security 1402
**********************************
This Is User 1
send message : 1
decode recived message : 2
exit : exit
exit
```

```
if __name__ == '__main__':
    main()
```

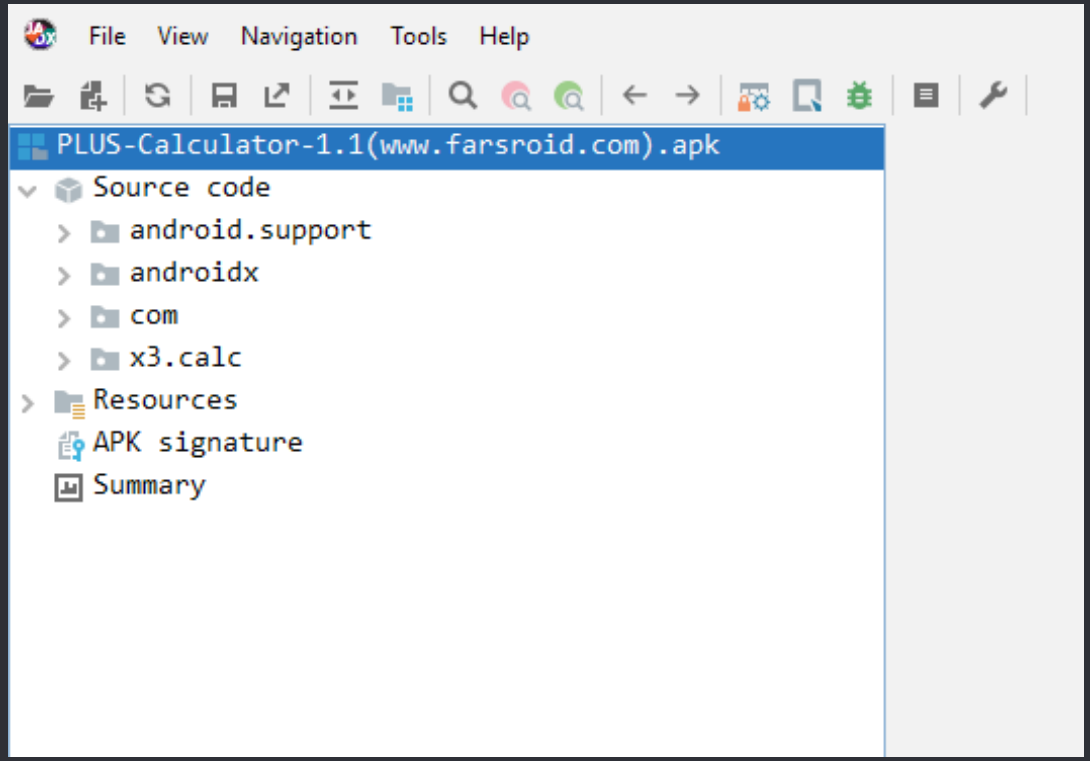
```
user g :  109446188314575224046196982155434284911552432512 04
user h :  116411933823313027392065220362022848424256657613 66
user q :  120343579691614703897269628901085759756284834170 77
_____
enter other side g:388195933700995998161677617504736849441526245879 2
enter other side h:263237683536270803464676077270943021339396785631 58
enter other side q:462915700056843266720131592503253558778699661675 35
**********************************
This Is User 2
send message : 1
decode recived message : 2
exit : exit
1
enter message:Security 1402
encrypted message :[361071810552606457876451606720914112018811255608693 7, 439376540551966894524356774443521991733734178511783 9, 430676014996482401563478422474343338432076075966996 1, 50
p:308133805320343393817525203483588493879403292351 89

**********************************
This Is User 2
send message : 1
decode recived message : 2
exit : exit
exit
```

Jadx

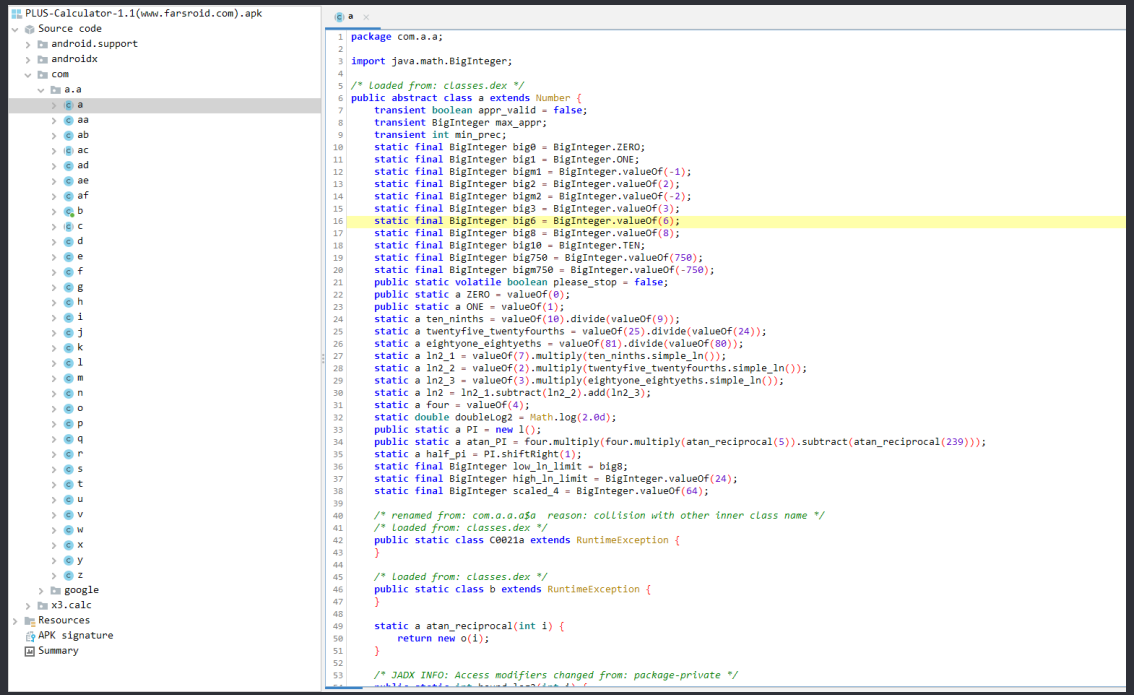
In this part we have to reverse engineer an Android app and report our analysis.

Our chosen program is an Android calculator called PLUS Calculator.

General environment



The main code of the program



Check the main code of the program

```
/* JADX INFO: Access modifiers changed from: package-private */
public static int bound_log2(int i) {
    return (int) Math.ceil(Math.log(Math.abs(i) + 1) / Math.log(2.0d));
}
```

```
public a cos() {
    BigInteger bigInteger = divide(PI).get_appr(-1);
    if (bigInteger.abs().compareTo(big2) >= 0) {
        BigInteger scale = scale(bigInteger, -1);
        a multiply = PI.multiply(valueOf(scale));
        return scale.and(big1).signum() != 0 ? subtract(multiply).cos().negate() : subtract(multiply).cos();
    } else if (get_appr(-1).abs().compareTo(big2) >= 0) {
        a cos = shiftRight(1).cos();
        return cos.multiply(cos).shiftLeft(1).subtract(ONE);
    } else {
        return new w(this);
    }
}
```

```
public a exp() {
    BigInteger bigInteger = get_appr(-10);
    if (bigInteger.compareTo(big2) > 0 || bigInteger.compareTo(bigm2) < 0) {
        a exp = shiftRight(1).exp();
        return exp.multiply(exp);
    }
    return new x(this);
}
```

```
public a sqrt() {
    return new ad(this);
}
```

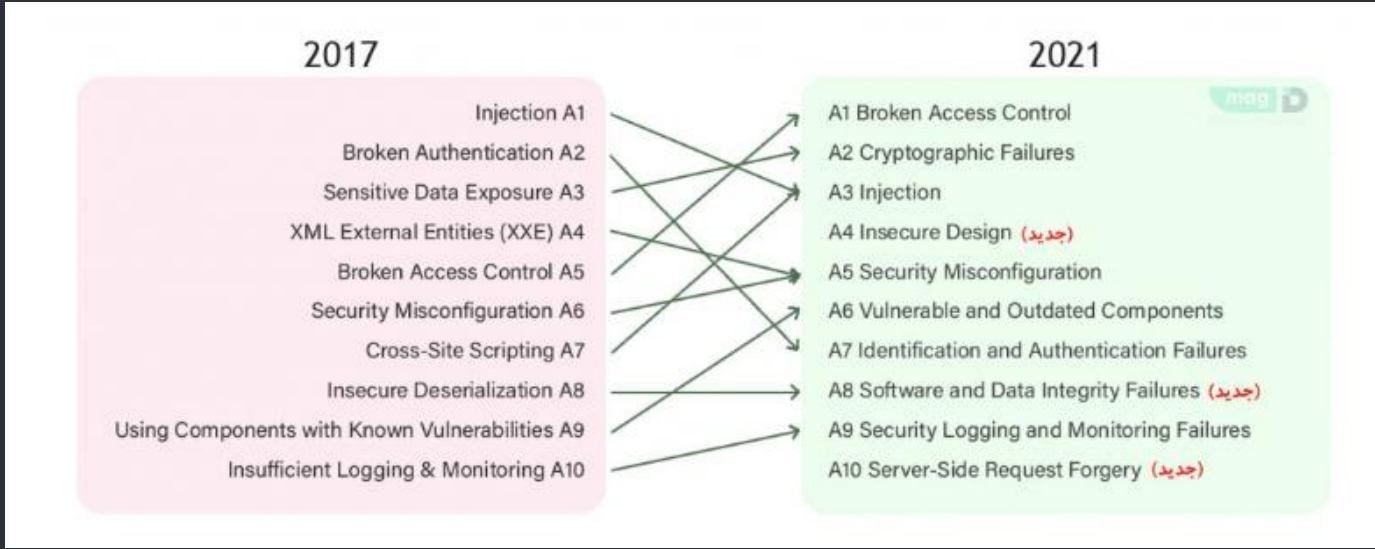
OWASP Top 10 2021

OWASP Top 10 is actually a report compiled by a team of security experts around the world, and its data is an analysis of reports obtained from a number of organizations.

List of vulnerabilities

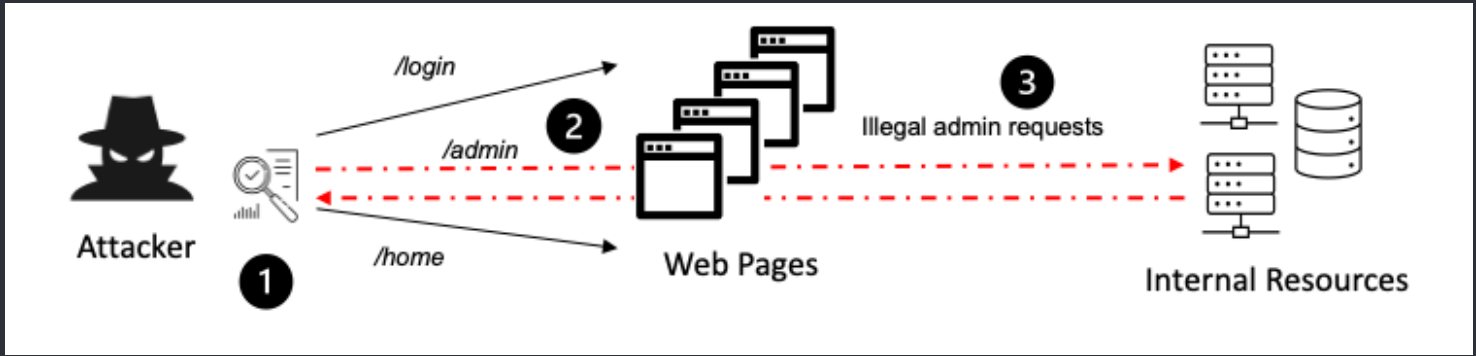
- A01:2021-Broken Access Control
- A02:2021-Cryptographic Failures
- A03:2021-Injection
- A04:2021-Insecure Design
- A05:2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- A08:2021-Software and Data Integrity Failures
- A09:2021-Security Logging and Monitoring Failures
- A10:2021-Server-Side Request Forgery

Changes from 2017 to 2021



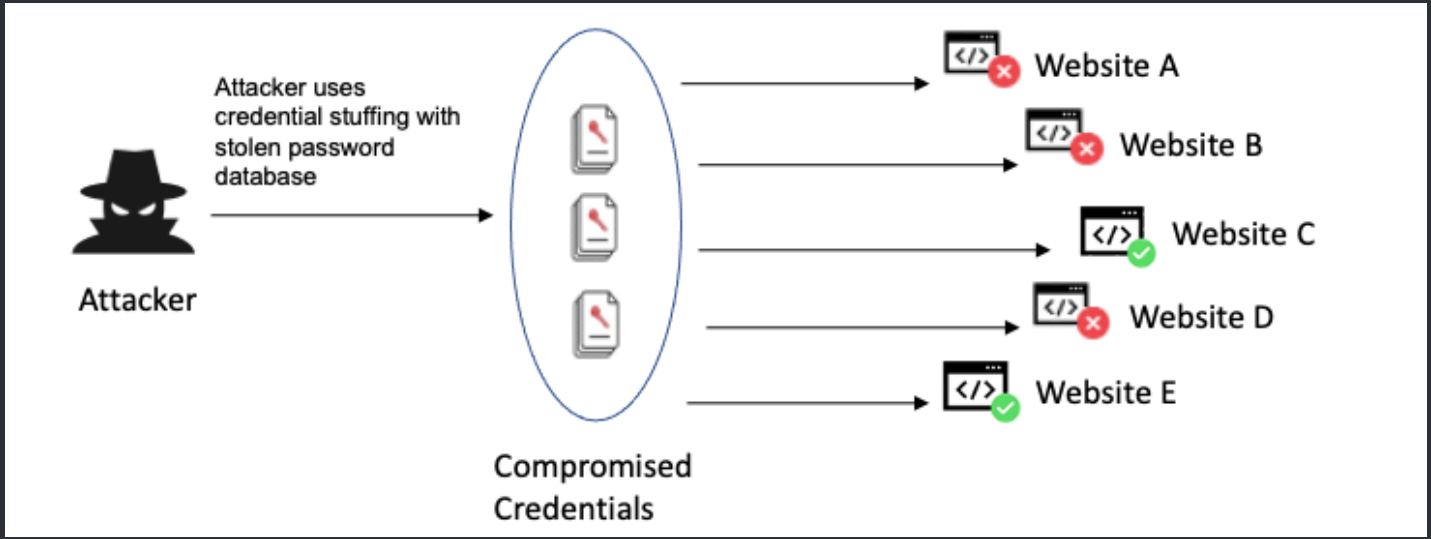
A01:2021-Broken Access Control

This type of risk includes a wide variety of risks, mainly risks that allow attackers to break the restrictions around access.



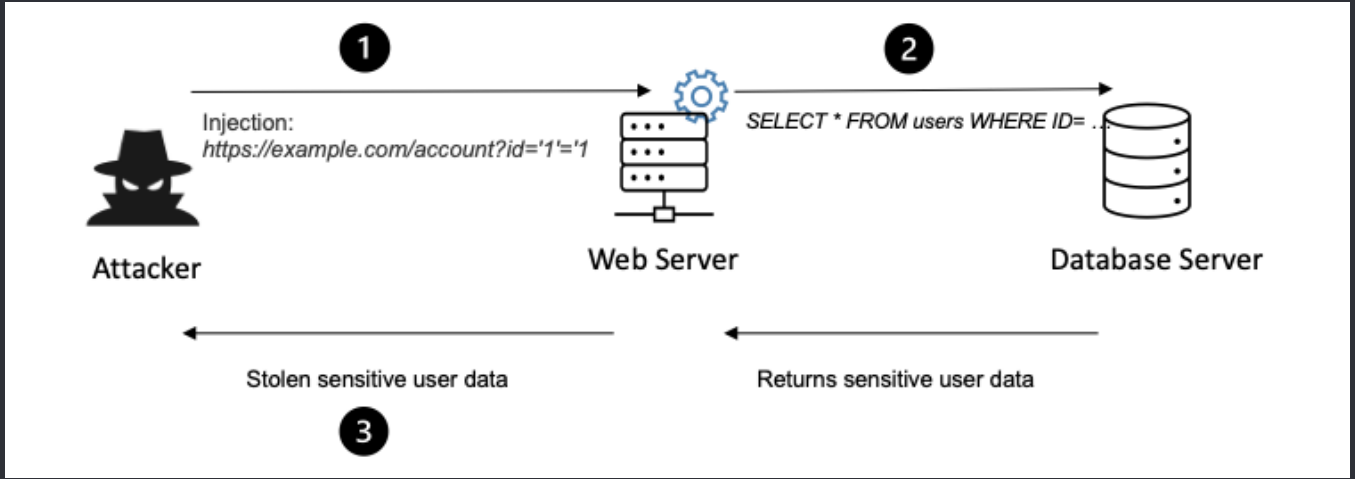
A02:2021-Cryptographic Failures

These types of errors occur when sensitive data and passwords are placed insecurely.



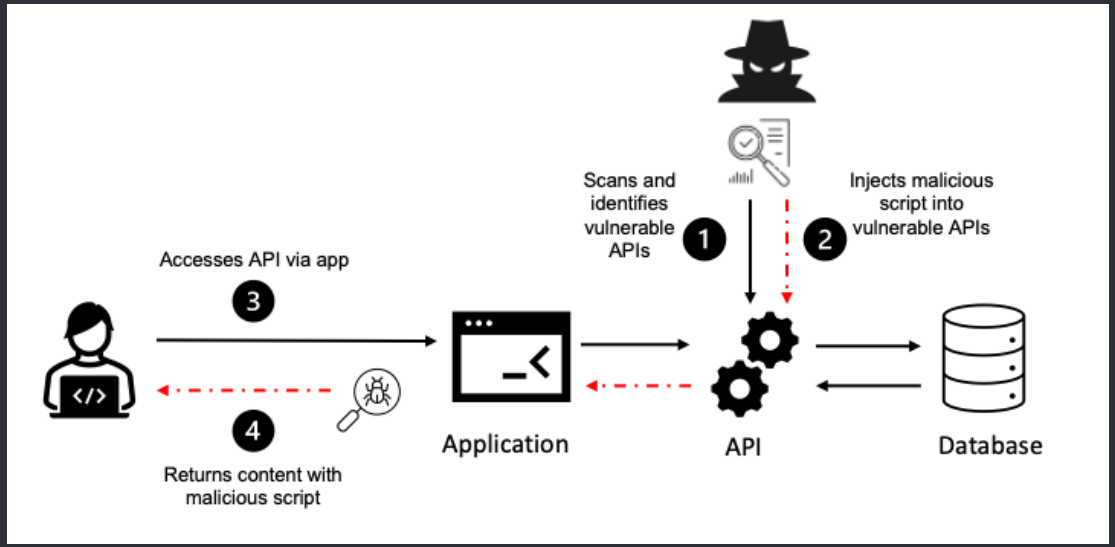
A03:2021-Injection

Risks in this domain include entering unreliable data to an interpreter that is sent as a request or command.



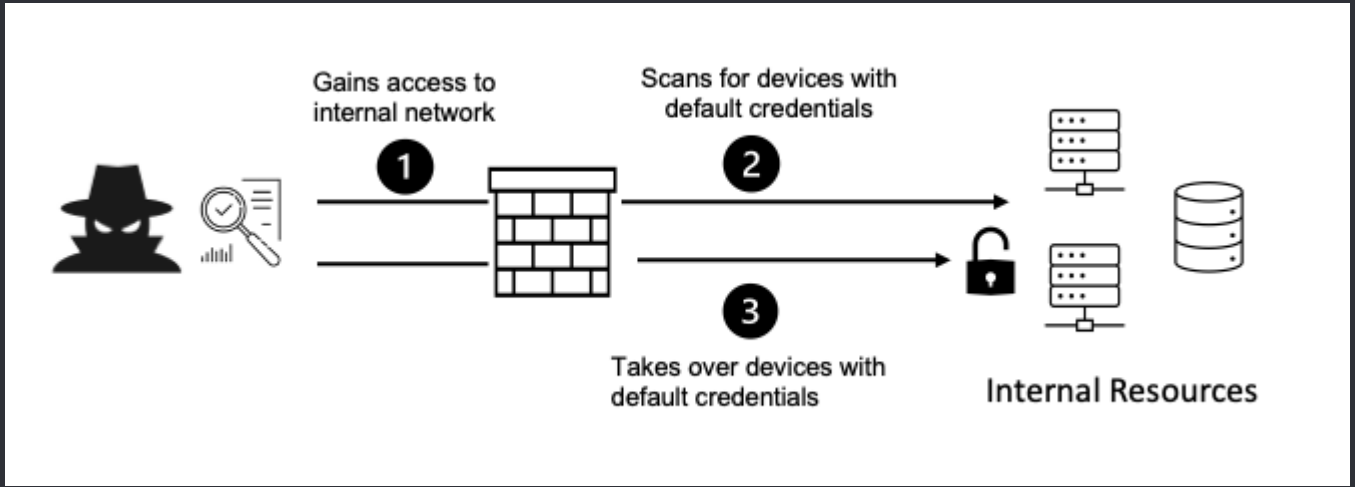
A04:2021-Insecure Design

This is a new issue that involves supporting risks due to "defective control design".



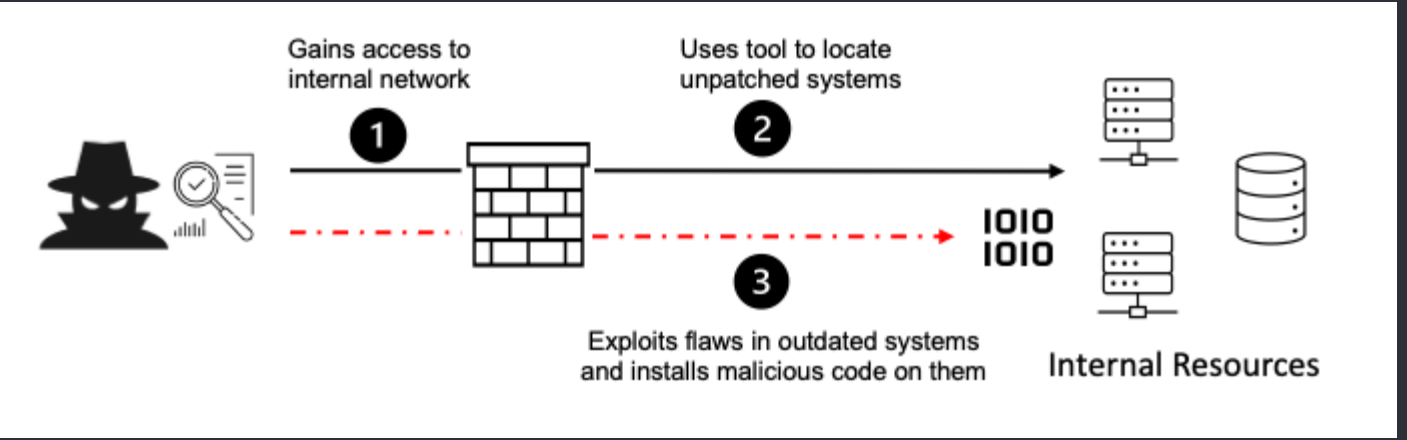
A05:2021-Security Misconfiguration

These types of risks include various types of security risks that include the web environment.



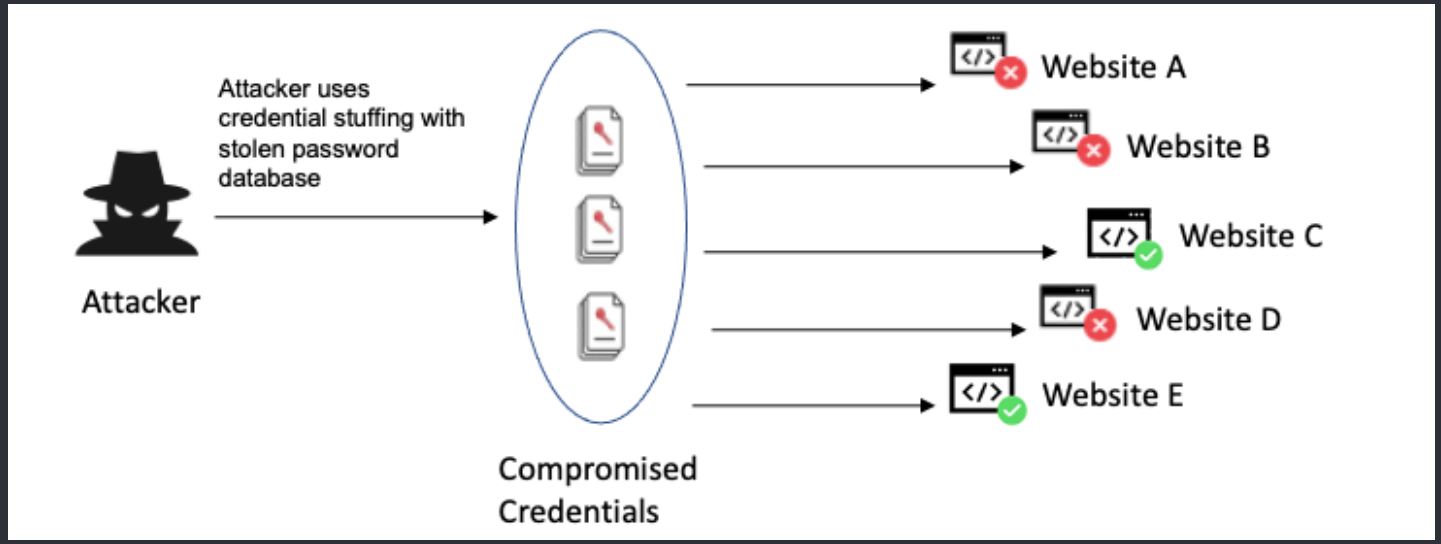
A06:2021-Vulnerable and Outdated Components

Using components and tools that are not patched or updated or have errors in a program causes its security to fail and exposes it to soft attacks.



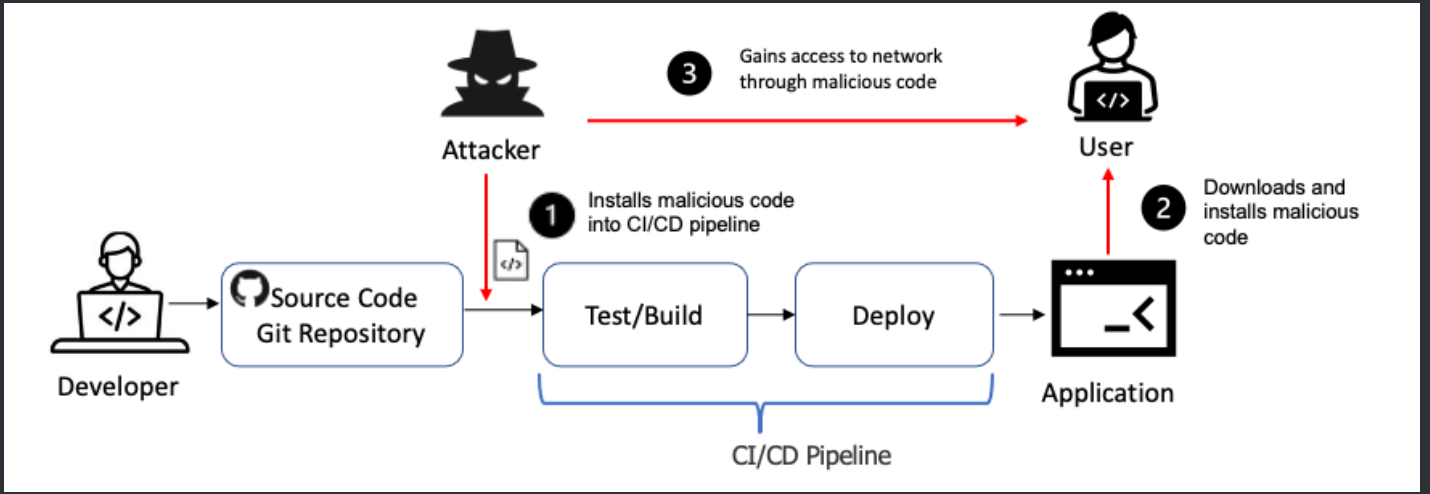
A07:2021-Identification and Authentication Failures

Verification and authentication as well as session management is a vital action to deal with authentication attacks.



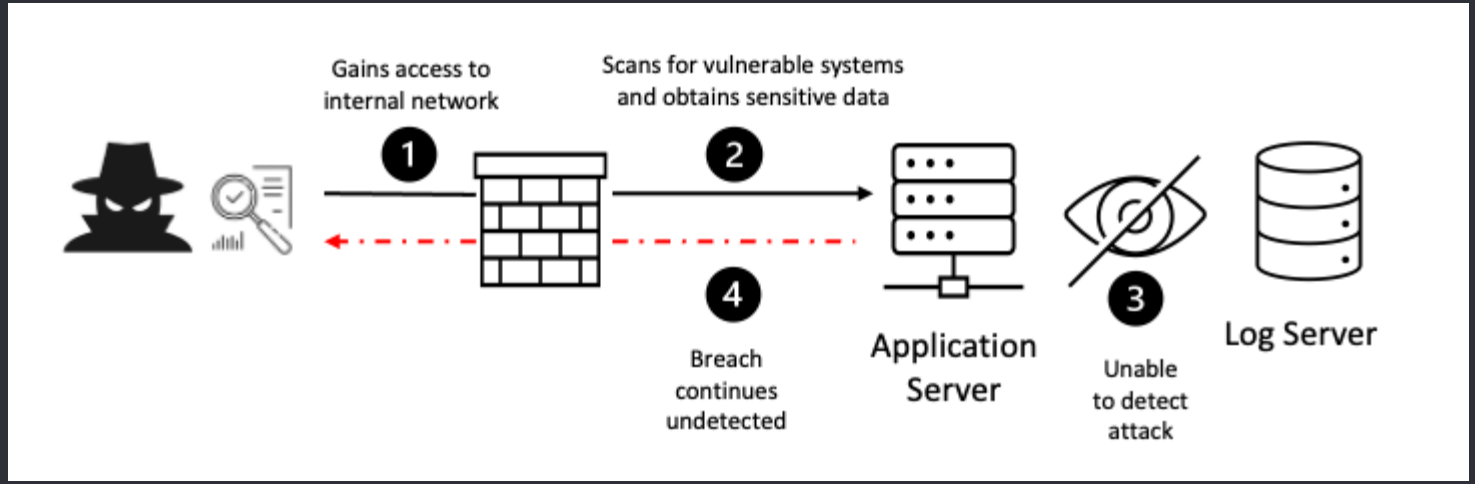
A08:2021-Software and Data Integrity Failures

Attacks and subsets of this category are related to data integrity protection violations in the application code or infrastructure.



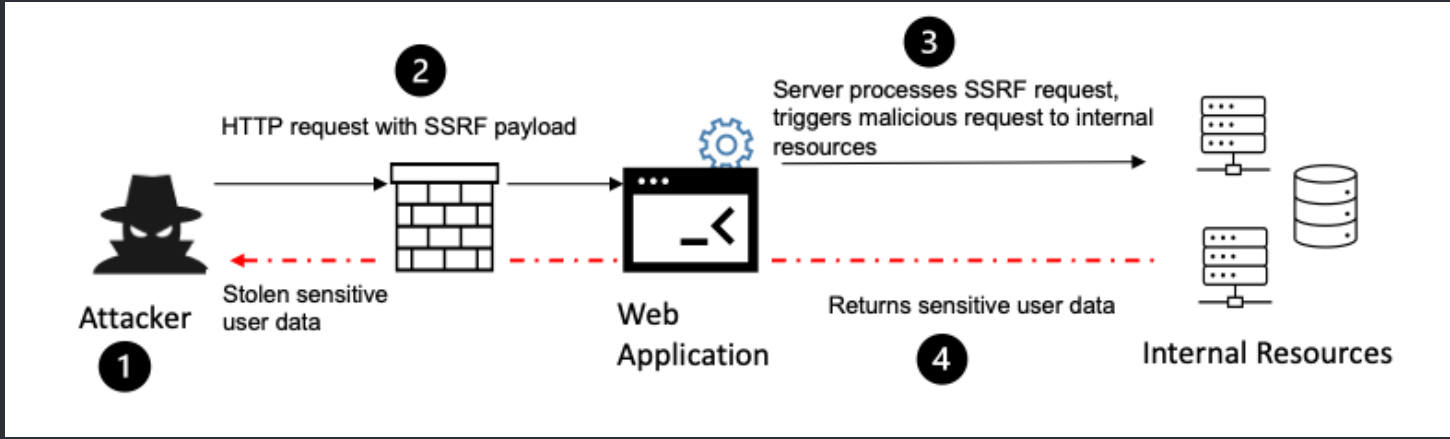
A09:2021-Security Logging and Monitoring Failures

According to the OWASP Top 10 2021, this category is important to help identify, mitigate, and respond to active application defects.



A10:2021-Server-Side Request Forgery

SSRF flaws occur when an application is retrieving data from an external source and there is no validation on the URL provided by the user.



Resources

https://www.irandnn.ir/
https://virgool.io/
https://www.cloudflare.com/
https://owasp.org/
https://github.com/skylot/jadx
https://www.codespeedy.com/
https://my.f5.com/manage/s/article/K44094284

Thanks!

ANY QUESTIONS?

You can find us at:

hoorieh95@gmail.com

elnazrezaee80@gmail.com