SYNOPSYS®

Safeway

SAST - Custom

August 31, 2018

This work performed under contract to:

Daimler Greater China Ltd. (DGRC)
IT/CD
Ines Janssen
8 Wangjing Street
Chaoyang District
100102 Beijing
China

For more information contact:

Amanvir Sangha
Security Consultant

Alexios Fakos
Managing Consultant

James Spooner
Managing Director

Proprietary Statement

The information in this documentation is subject to change without notice and should not be construed as a commitment by Synopsys, Inc. Synopsys, Inc. makes no representations or warranties, express or implied, with respect to the documentation and shall not be liable for any damages, including any indirect, incidental, consequential damages (such as loss of profit, loss of use of assets, loss of business opportunity, loss of data or claims for or on behalf of user’s customers), that may be suffered by the user.

Synopsys, Inc.
185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

U.S. Sales: 800.873.8193
International Sales: +1 415.321.5237
Email: sig-info@synopsys.com
www.synopsys.com/software.

Table of Contents

Document Revision History

Version	Modification	Date	Author
1.0	Initial Draft Document	28/08/2018	Amanvir Sangha
1.1	Final	31/08/2018	Amanvir Sangha

Contacts

Contact	Title	Organization	Phone Number	Email Address
James Spooner	Managing Director	Synopsys	+44 207 510 9022	jspooner@synopsys.com
Alexios Fakos	Managing Consultant	Synopsys	+49 89 9932 0279	alexios.fakos@synopsys.com
Amanvir Sangha	Security Consultant	Synopsys	+4474663340744	amanvir@synopsys.com

Executive Summary

Daimler AG has engaged Synopsys to perform a SAST - Custom assessment on Safeway which is written in a mixture of programming languages and frameworks such as Java, JavaScript, Vue and react-native.

The objective of this assessment was to assess the overall security posture of the application from a white-box perspective. This includes determining the application's ability to resist common attack patterns. All code and modules received was considered in scope for the code review.

During the assessment, Synopsys identified 4 findings characterized as follows:

- 1 "high",
- 2 "medium",
- 1 "low",

priority findings in this application.

High Priority Findings:

- **HTTPS Not Enabled (CWE-319)**

  The application does not use HTTPS to encrypt traffic over the network. TLS and SSL are common protocols used in protecting the confidentiality and integrity of network communications over HTTPS. At a high level, this is accomplished by first verifying the server's identity through the use of trusted certificate authorities. Then, the browser and server generate a shared secret, which is used to encrypt all network communication.

Moderate Priority Findings:

- **Unrestricted HTML5 Cross-Domain Resource Sharing (CWE-284)**

  An unrestricted CORS policy allows an attacker to access sensitive data or perform unauthorized user actions without user knowledge. Malicious Javascript can perform these actions even if the server uses Cross Site Request Forgery tokens.

- **Insecure Pseudo-Random Number Generator Used (CWE-330)**

  The application uses a cryptographically insecure pseudo-random number generator (PRNG). The outputs it generates are potentially predictable. A PRNG is a deterministic algorithm that takes an unpredictable seed as input and generates a stream of output that appears random. There are two popular types of PRNGs: PRNGs that produce outputs that are statistically random, and Cryptographically Secure PRNGs (CSPRNGs) that produce outputs that are suitable for cryptographic purposes.

There is 1 other low risk finding described in the remainder of this document.

Summary of Findings

Finding	CVSS Score	CWE	Instances	Status
HTTPS Not Enabled	8.0	319	2	Open
Unrestricted HTML5 Cross-Domain Resource Sharing	5.4	284	1	Open
Insecure Pseudo-Random Number Generator Used	4.8	330	1	Open
Hard-Coded Secret Tokens Present in Application Code	3.8	259	1	Open

1 Methodology

1.1 Assessment Type

Synopsys received source code for the Safeway application and assessed the code to identify security vulnerabilities. This assessment involved automated code scanning using AppScan Source and Secure Assist. The assessment also included a risk-based manual code review. The assessment methodology was based on the application's architecture for efficiency and effectiveness.

1.2 Risk Assessment Methodology

The severity assigned to each vulnerability is calculated using the Common Vulnerability Scoring System (CVSS v3) standard. CVSS scoring methodology is based on 3 groups: Base, Temporal and Environmental. The Base group determines the risk score specific to the vulnerability. The Temporal group determines the temporary vulnerability score subject to change over time. The Environmental score is based on user/application environment.

More information on the CVSS v3 standard for risk assessment can be found at the link below:

https://www.first.org/cvss/specification-document

Risk Assessment Table

Risk Level	Symbol	Definition
Critical	C	**CVSS v3.0 Score: 9.0 to 10.0** This finding will compromise Confidentiality and/or Integrity and/or Availability. These findings represent an important risk to the application's security, therefore it is a top priority and must be remediated in an immediate manner (or risk accepted).
High	H	**CVSS v3.0 Score: 7.0 to 8.9** This Finding on its own will compromise Confidentiality and/or Integrity and/or Availability of a significant data element. These findings represent an elevated and important risk to the application's security, hence this must be considered a top priority to remediate and must be remediated (or risk accepted).
Medium	M	**CVSS v3.0 Score: 4.0 to 6.9** This Finding will compromise Confidentiality and/or Integrity and/or Availability of a significant data element but requires one or more "pre-conditions" to exist. These findings represent a significant but less important risk to the application's security in that should the pre-conditions be introduced to the environment, so too would the significant risk. These findings should be addressed quickly and must be remediated (or risk accepted).
Low	L	**CVSS v3.0 Score: 0.1 to 3.9** A less important risk to the application's security – this should be addressed within a reasonable time unless there is business justification not to.
Informational	I	**CVSS v3.0 Score: N/A** Potential for some risk to the application's security – this is used to identify discussion items in order to determine whether remediation is necessary.

2 Findings

2.1 Finding Details

2.1.1 High Priority Findings

2.1.1.1 HTTPS Not Enabled

Description:

The application does not use HTTPS to encrypt traffic over the network. TLS and SSL are common protocols used in protecting the confidentiality and integrity of network communications over HTTPS. At a high level, this is accomplished by first verifying the server's identity through the use of trusted certificate authorities. Then, the browser and server generate a shared secret, which is used to encrypt all network communication.

HTTPS is not enabled on the application server resulting in application traffic being delivered in plaintext. An attacker listening on any network between the victim and the application server may view and modify application traffic.

Instances:

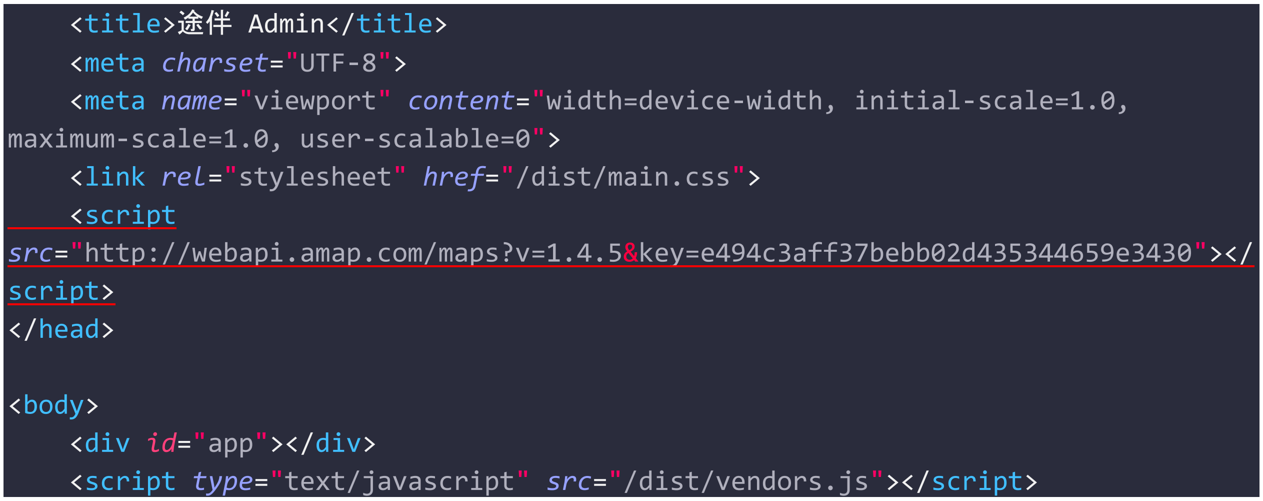
1. \safeway\adminpanel\index.html, 9
2. \safeway_frontend\ios\manager\AppStateManager\AppStateManager.m, 69

Context Description:

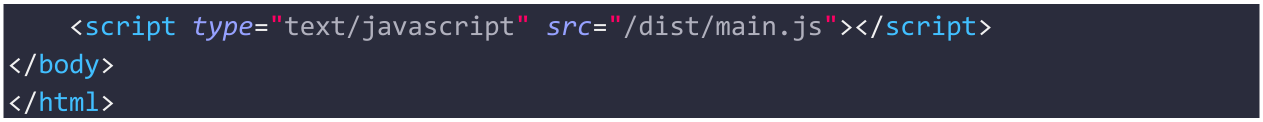
The application has hardcoded “http” URLs for including scripts which implies that the admin panel is not served over HTTPS. An attacker listening on any network between the victim and the application server may view and modify application traffic.

Code Snippet:

Instance 1:

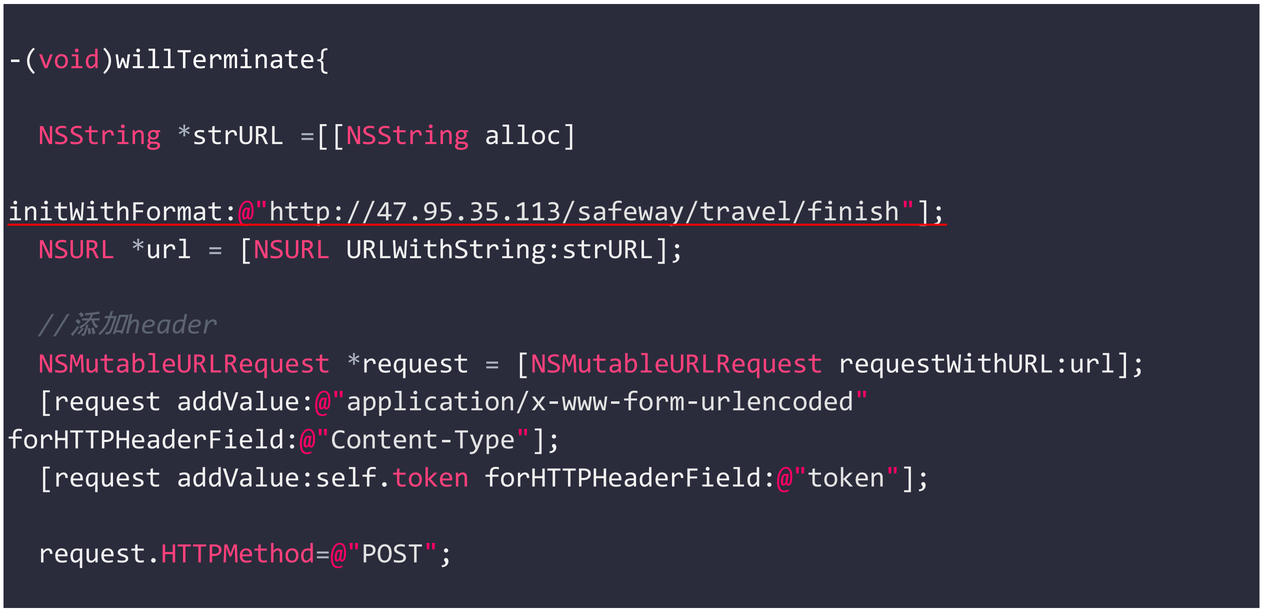


```
    <title>途伴 Admin</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">
    <link rel="stylesheet" href="/dist/main.css">
    <script src="http://webapi.amap.com/maps?v=1.4.5&key=e494c3aff37bebb02d435344659e3430"></script>
</head>

<body>
    <div id="app"></div>
    <script type="text/javascript" src="/dist/vendors.js"></script>
```



```
    <script type="text/javascript" src="/dist/main.js"></script>
</body>
</html>
```

Instance 2:



```

-(void)willTerminate{

  NSString *strURL =[[NSString alloc]

initWithFormat:@"http://47.95.35.113/safeway/travel/finish"];
  NSURL *url = [NSURL URLWithString:strURL];

  //添加header
  NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
  [request addValue:@"application/x-www-form-urlencoded"
forHTTPHeaderField:@"Content-Type"];
  [request addValue:self.token forHTTPHeaderField:@"token"];

  request.HTTPMethod=@"POST";

```

CVSS Score: 8.0
(AV:A/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H)

CWE ID: 319

Remediation:

HTTPS should be enabled on the application server. The exact process of enabling HTTPS depends on the technology stack, but there are common considerations for all HTTPS implementations. The first consideration is to only use TLS v1.1 or v1.2; if possible, SSL v2, SSL v3 and TLS v1.0 should be avoided due to known protocol security vulnerabilities. The second consideration is to only support strong cryptographic ciphers. The properties of strong cryptographic ciphers are ciphers that contain authentication and have a key size of 128 bits or higher.

In addition to the previous configuration changes, these best practices should be followed to prevent known attacks against TLS/SSL:

- Disable TLS compression
- Disable client-initiated renegotiation
- Generate a certificate per host, avoid wildcard certificates

Once HTTPS has been enabled on the server, there are several application-level tasks which should be performed. The first task is to ensure that only HTTPS connections are accepted. When a user attempts to navigate to any part of the application over HTTP, the application should redirect the user to the HTTPS version of the application. The second task is to always set the secure cookie attribute when setting session identifiers. The secure attribute tells the user's browser that the cookie should only be delivered over an encrypted channel. If the application fails to set this cookie attribute and a user accidentally navigates to the HTTP version of the application, an attacker listening on the network would still be able to obtain the user's session identifier.

For more information on protecting transport layer security, see the Transport Layer Protection Cheat Sheet created by OWASP at https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

2.1.2 Medium Priority Findings

2.1.2.1 Unrestricted HTML5 Cross-Domain Resource Sharing

Description:

The server is configured with an unrestricted HTML5 Cross-Origin Resource Sharing (CORS) policy. CORS defines whether resources on other domains can interact with this server. An attacker can place malicious Javascript on his domain that can exploit the unrestrictive CORS policy to access sensitive data on this server or perform sensitive operations without the user's knowledge. Additionally, an attacker could exploit security vulnerabilities on other domains to compromise services on this server. The CORS policy relaxes the Same Origin Policy, an important security control that isolates potentially malicious resources to its respective domain name.

If a script attempts to violate the Same Origin Policy by interacting with another domain, modern browsers will check a server's CORS policy by issuing a "pre-flight request". The browser allows the interaction only if the server responds with an Access-Control-Allow-Origin header that lists the script's domain or a wildcard match (*). A wildcard match allows interaction from any other domain, which allows any malicious content to retrieve content from this server or perform user actions.

An unrestricted CORS policy allows an attacker to access sensitive data or perform unauthorized user actions without user knowledge. Malicious Javascript can perform these actions even if the server uses Cross Site Request Forgery tokens.

Instances:

1. \safeway_backend\src\main\java\com\bamboonetworks\safeway\common\security\handler\JwtAuthenticationEntryPoint.java, 31, 32, 33

Code Context:

In the application source code Access-Control-Allow-Origin is set to “*”. An attacker can place malicious Javascript on his domain that can exploit the unrestrictive CORS policy to access sensitive data on this server or perform sensitive operations without the user's knowledge.

Code Snippet:

Instance 1:

```
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

    @Autowired
    private JSONUtils jsonUtils;

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse
response,
            AuthenticationException authException) throws IOException,
ServletException {
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json; charset=utf-8");

        JSONObject result =
jsonUtils.buildResponse(SafewayStatusCode.AUTHORIZATION_FAILED, null);
        response.setStatus(HttpStatus.UNAUTHORIZED.value());
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Methods", "*");
        response.setHeader("Access-Control-Allow-Headers", "*");
        response.getWriter().write(result.toJSONString());

    }

}
```

CVSS Score: 5.4
(AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N)

CWE ID: 284

Remediation:

The Access-Control-Allow-Origin header should not be set to a wildcard match. In most cases, this header can be safely removed. However, if the application requires a relaxation of the Same Origin Policy, the Access-Control-Allow-Origin header should whitelist only domains that are trusted by this server.

2.1.2.2 Insecure Pseudo-Random Number Generator Used

Description:

The application uses a cryptographically insecure pseudo-random number generator (PRNG). The outputs it generates are potentially predictable. A PRNG is a deterministic algorithm that takes an unpredictable seed as input and generates a stream of output that appears random. There are two popular types of PRNGs: PRNGs that produce outputs that are statistically random, and Cryptographically Secure PRNGs (CSPRNGs) that produce outputs that are suitable for cryptographic purposes. Loosely speaking, PRNGs generate outputs that do not contain any recognizable patterns; however, given a small amount of output from the PRNG, the past and/or future outputs may be computable. CSPRNGs on the other hand remain unpredictable as long as the seed used to initialize the CSPRNG is unknown.

An attacker who sees a small amount of output from a PRNG that is not cryptographically secure may be able to predict all past and/or future outputs from the PRNG. Using the PRNG outputs to generate values that are meant to be unpredictable (e.g. cryptographic keys, passwords, etc.) is insecure because the attacker will be able to guess these values and potentially gain unauthorized access to data or functionality.

Instances:

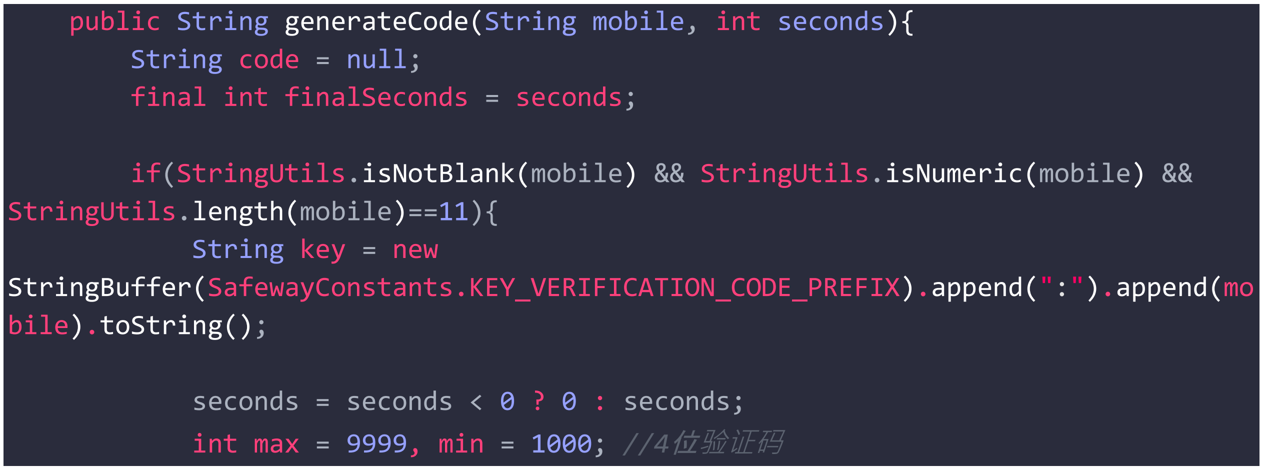
1. \safeway_backend\src\main\java\com\bamboonetworks\safeway\utils\VerificationCodeUtils.java, 29, 58

Context Description:

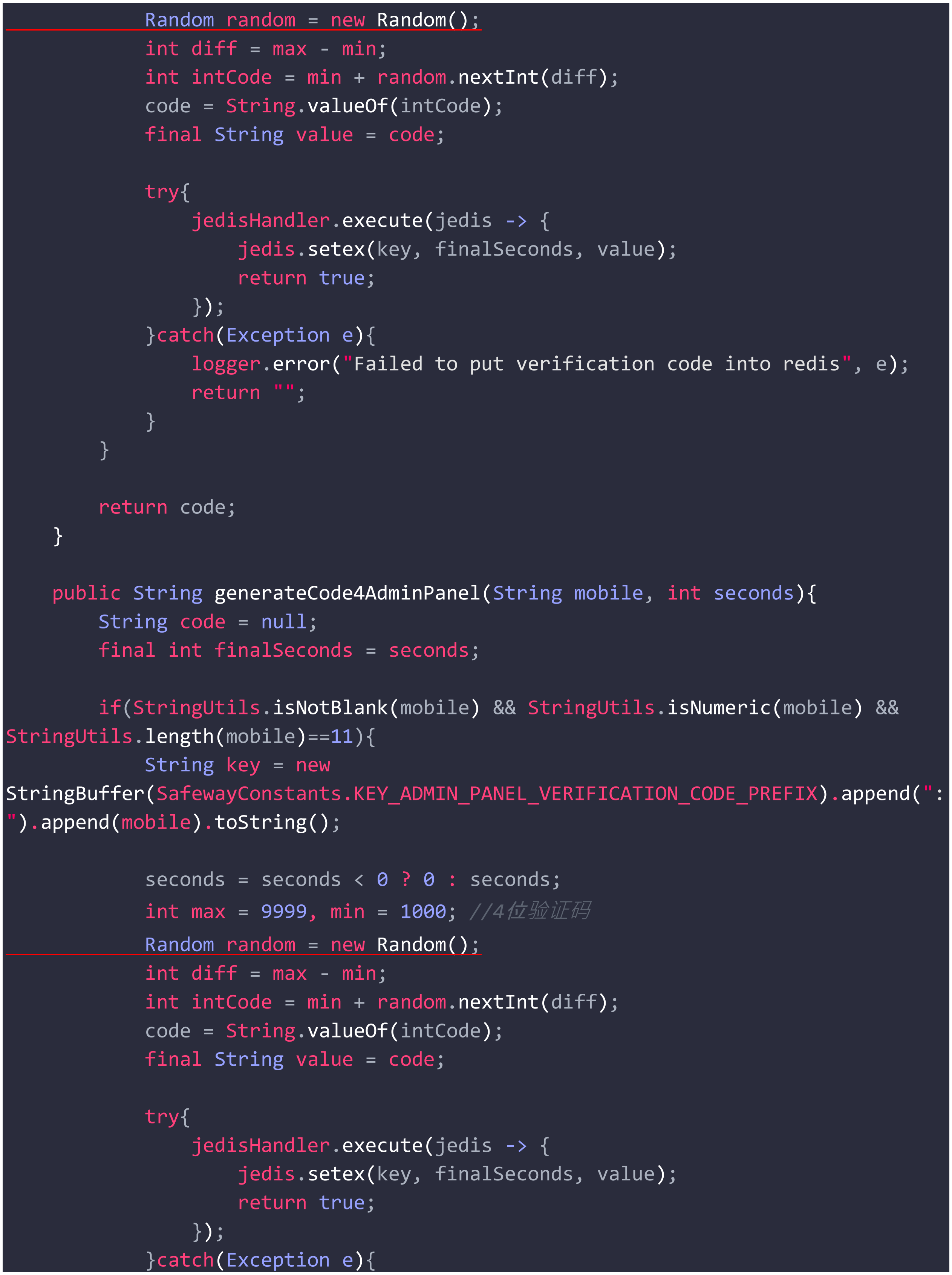
The application uses insecure new Random() method to generate random numbers and these numbers are sent as a captcha to the users. An attacker who sees a small amount of output from a RNG that is not cryptographically secure may be able to predict all past and/or future outputs from the RNG.

Code Snippet:

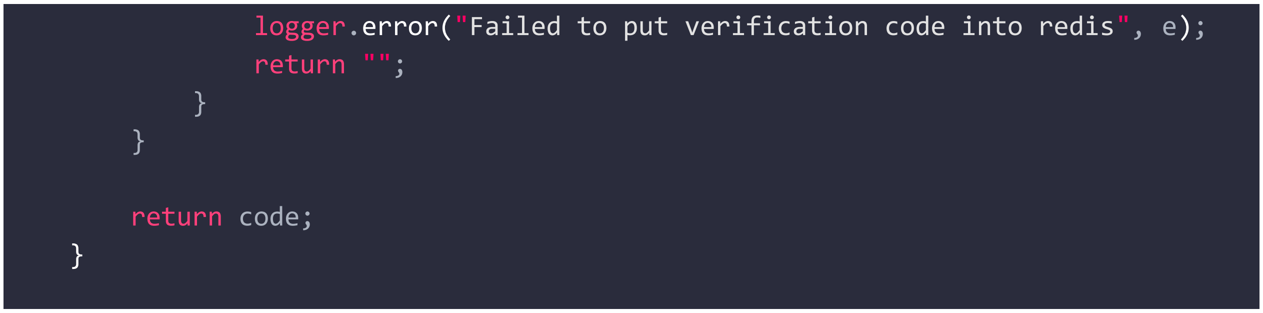
Instance 1:



```
    public String generateCode(String mobile, int seconds){
        String code = null;
        final int finalSeconds = seconds;

        if(StringUtils.isNotBlank(mobile) && StringUtils.isNumeric(mobile) &&
StringUtils.length(mobile)==11){
            String key = new
StringBuffer(SafewayConstants.KEY_VERIFICATION_CODE_PREFIX).append(":").append(mobile).toString();

            seconds = seconds < 0 ? 0 : seconds;
            int max = 9999, min = 1000; //4位验证码
```



```
            Random random = new Random();
            int diff = max - min;
            int intCode = min + random.nextInt(diff);
            code = String.valueOf(intCode);
            final String value = code;

            try{
                jedisHandler.execute(jedis -> {
                    jedis.setex(key, finalSeconds, value);
                    return true;
                });
            }catch(Exception e){
                logger.error("Failed to put verification code into redis", e);
                return "";
            }
        }

        return code;
    }

    public String generateCode4AdminPanel(String mobile, int seconds){
        String code = null;
        final int finalSeconds = seconds;

        if(StringUtils.isNotBlank(mobile) && StringUtils.isNumeric(mobile) &&
StringUtils.length(mobile)==11){
            String key = new
StringBuffer(SafewayConstants.KEY_ADMIN_PANEL_VERIFICATION_CODE_PREFIX).append(":
").append(mobile).toString();

            seconds = seconds < 0 ? 0 : seconds;
            int max = 9999, min = 1000; //4位验证码
            Random random = new Random();
            int diff = max - min;
            int intCode = min + random.nextInt(diff);
            code = String.valueOf(intCode);
            final String value = code;

            try{
                jedisHandler.execute(jedis -> {
                    jedis.setex(key, finalSeconds, value);
                    return true;
                });
            }catch(Exception e){
```



```
                logger.error("Failed to put verification code into redis", e);
                return "";
            }
        }

        return code;
    }
```

CVSS Score: 4.8
(AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N)

CWE ID: 330

Remediation:

The Java standard library function SecureRandom() should be used instead.

Use a CSPRNG with an unpredictable seed to ensure that the random values needed by the application cannot be predicted by an attacker. Some standardized CSPRNGs are described in NIST SP 800-90A, and ANSI X8.82 Part 3. See NIST SP 800-90B for guidance on entropy sources for seed generation.

2.1.3 Low Priority Findings

2.1.3.1 Hard-Coded Secret Tokens Present in Application Code

Description:

The application stores sensitive information such as encryption keys insecurely. When the application communicates with external and internal entities, it may store, encryption keys in clear text within a properties file, or as hard-coded text within the source code. Even if access to properties files and source code is restricted to a group of internal users, it is still important to consider unauthorized malicious employees that have access to the data center and web server as a potential threat.

Instances:

1. *\safeway_backend\src\main\resources\app-prod.properties, 3, 39, 55, 60*

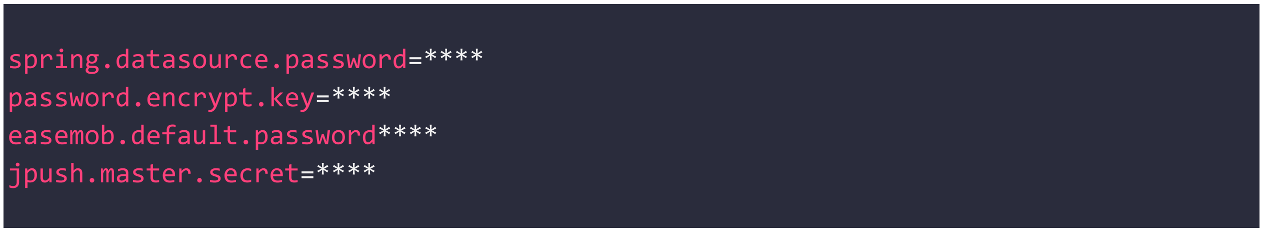
*Note: In the below code snippets, the actual keys are replaced with "****"*

Code Context:

In the application source code encryption keys and secret keys are hard-coded. If any malicious insider or attacker gets access to the source code, the attacker can use the key to break confidentiality in the application.

Code Snippet:

Instance 1:



```
spring.datasource.password=****
password.encrypt.key=****
easemob.default.password****
jpush.master.secret=****
```

CVSS Score: 3.8
(AV:L/AC:L/PR:L/UI:N/S:C/C:N/I:N/A:L)

CWE ID: 259

Remediation:

It is recommended to store encryption key outside the source code. While it is a good practice to store credentials in properties files, it's a best practice to encrypt them within the properties file and store the key in a safe location. This is especially necessary if people other than the administrators have access to the properties file. Upon encryption, if different people need access to the properties file, access to the encryption key should be restricted.

About Synopsys

Synopsys offers the most comprehensive solution for building integrity—security and quality—into your SDLC and supply chain. We've united leading testing technologies, automated analysis, and experts to create a robust portfolio of products and services. This portfolio enables companies to develop customized programs for detecting and remediating defects and vulnerabilities early in the development process, minimizing risk and maximizing productivity. Synopsys, a recognized leader in application security testing, is uniquely positioned to adapt and apply best practices to new technologies and trends such as IoT, DevOps, CI/CD, and the Cloud. We don't stop when the test is over. We offer onboarding and deployment assistance, targeted remediation guidance, and a variety of training solutions that empower you to optimize your investment. Whether you're just starting your journey or well on your way, our platform will help ensure the integrity of the applications that power your business.

For more information, go to www.synopsys.com/software.