360° Product Security

Boost Your Product Security Program

secureIO

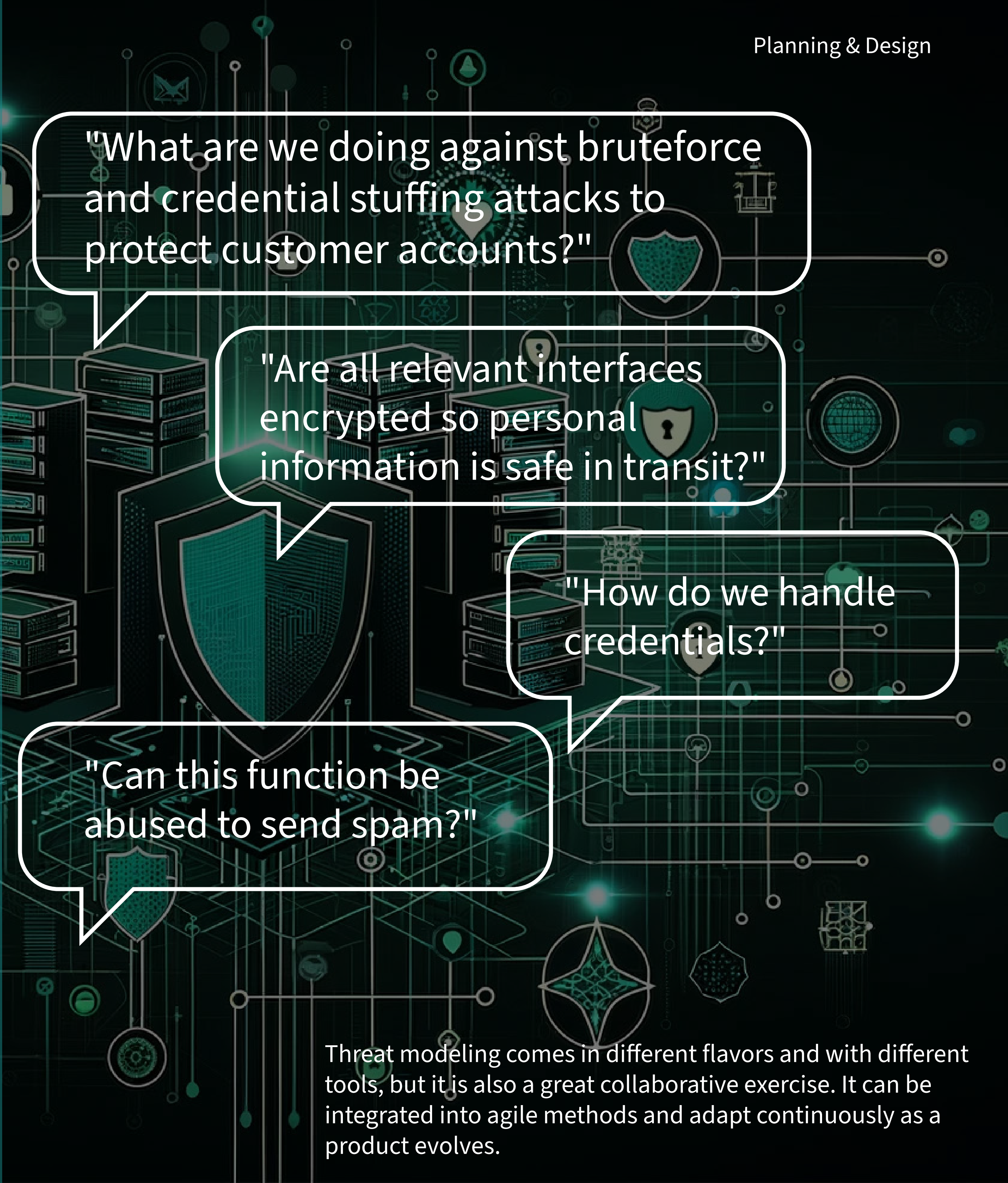
1. Threat Modeling

"Plans are nothing. Planning is everything."

Threat Modeling is the activity of analysing application designs, features and business logic flows for security issues that might affect the confidentiality, integrity or availability of systems or data. Threat modeling systems early and often ensure there is less likelihood of design related security flaws. It reduces the risk of costly security issues in production, supports compliance to standards as ISO 27001:2022 and is also a great team activity.

A common process is to consider 4 questions when designing a system:
1. What are we talking about
2. What can go wrong
3. What are we going to do about it?
4. Did we do a good job?

Think about it: Would you drive a car that has not been designed with safety in mind? Would you use a medical device where no one checked whether it is safe to use? Would you feel well putting your credit card details, health data or personal information in a software that has not considered security in its planning stage?



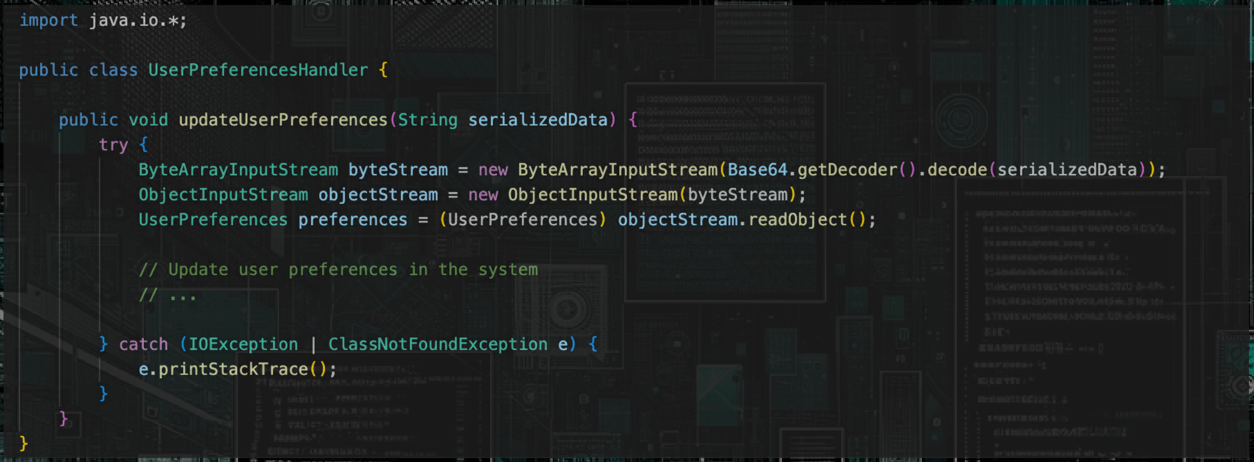
"What are we doing against bruteforce and credential stuffing attacks to protect customer accounts?"

"Are all relevant interfaces encrypted so personal information is safe in transit?"

"How do we handle credentials?"

"Can this function be abused to send spam?"

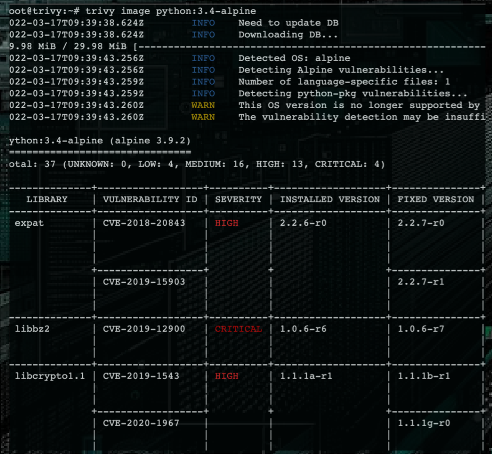
Threat modeling comes in different flavors and with different tools, but it is also a great collaborative exercise. It can be integrated into agile methods and adapt continuously as a product evolves.



```
import java.io.*;

public class UserPreferencesHandler {

    public void updateUserPreferences(String serializedData) {
        try {
            ByteArrayInputStream byteStream = new ByteArrayInputStream(Base64.getDecoder().decode(serializedData));
            ObjectInputStream objectStream = new ObjectInputStream(byteStream);
            UserPreferences preferences = (UserPreferences) objectStream.readObject();

            // Update user preferences in the system
            // ...

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

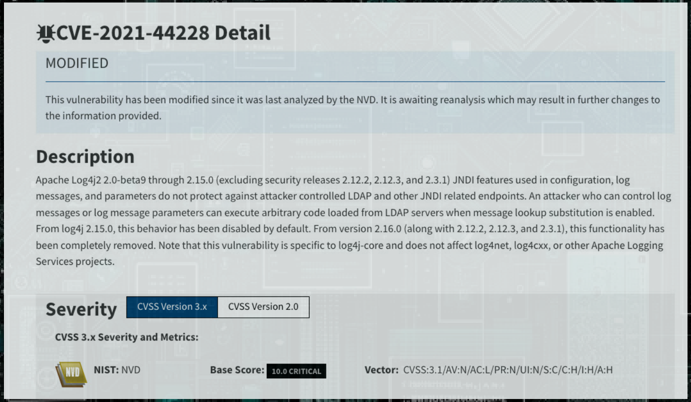
Vulnerable code example that contains multiple potential issues

Static scanning tools can be highly automated and integrated in CI/CD pipelines to ensure fast but secure delivieries. They are essential to provide a secure-by-default environment and turn DevOps into DevSecOps where teams can deploy a secure product using automated security checks in the pipeline. They usually detects unvalidated user input, outdated cryptographic functions, plaintext passwords, insecure API calls and a lot of other types of vulnerabilities.



```
oot@trivy:~# trivy image python:3.4-alpine
022-03-17T09:39:38.624Z        INFO    Need to update DB
022-03-17T09:39:38.624Z        INFO    Downloading DB...
9.98 MiB / 29.98 MiB [-------------------------------------------------------
022-03-17T09:39:43.256Z        INFO    Detected OS: alpine
022-03-17T09:39:43.256Z        INFO    Detecting Alpine vulnerabilities...
022-03-17T09:39:43.259Z        INFO    Number of language-specific files: 1
022-03-17T09:39:43.259Z        INFO    Detecting python-pkg vulnerabilities...
022-03-17T09:39:43.260Z        WARN    This OS version is no longer supported by
022-03-17T09:39:43.260Z        WARN    The vulnerability detection may be insuffi

ython:3.4-alpine (alpine 3.9.2)
===============================
otal: 37 (UNKNOWN: 0, LOW: 4, MEDIUM: 16, HIGH: 13, CRITICAL: 4)

--------------+------------------+----------+-------------------+---------------+
  LIBRARY     | VULNERABILITY ID | SEVERITY | INSTALLED VERSION | FIXED VERSION |
--------------+------------------+----------+-------------------+---------------+
 expat        | CVE-2018-20843   | HIGH     | 2.2.6-r0          | 2.2.7-r0      |
              |                  |          |                   |               |
              |                  |          |                   |               |
              +------------------+          +                   +---------------+
              | CVE-2019-15903   |          |                   | 2.2.7-r1      |
              |                  |          |                   |               |
--------------+------------------+----------+-------------------+---------------+
 libbz2       | CVE-2019-12900   | CRITICAL | 1.0.6-r6          | 1.0.6-r7      |
              |                  |          |                   |               |
--------------+------------------+----------+-------------------+---------------+
 libcrypto1.1 | CVE-2019-1543    | HIGH     | 1.1.1a-r1         | 1.1.1b-r1     |
              |                  |          |                   |               |
              +------------------+          +                   +---------------+
              | CVE-2020-1967    |          |                   | 1.1.1g-r0     |
              |                  |          |                   |               |
```

Container scanning result example



CVE-2021-44228 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Description

Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.

Severity CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 10.0 CRITICAL **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

CVE entry for Log4j

Scanning of container images, infrastructure as code descriptions and libraries (dependencies) significantly contributes to a better security posture of the final product and is essential to manage 3rd party risk. While only a low percentage of vulnerabilities might be exploitable by attackers and further analysis is often needed to develop a reasonable remediation plan, scanning is the first step to achieve transparency of the security status of the product and product stack. Instead of SAST and depending on your development setup you might chose to look into IAST (interactive application security testing) as a way to detect vulnerabilities during testing by instrumenting the application.

2. Static Scanning

"The truth is in the code"

A good plan needs even better implementation. Static scanning helps to ensure the implementation does not contain security flaws. Static scanning can be done

- on code level (SAST)
- on software dependencies (SCA)
- on container descriptions
- on infrastructure as code (IAC)

or even on binary artifacts.

Codescanning is an essential building block in ensuring security in depth throughout the software development lifecycle. There is a variety of code scanning tools on the market with different strength and integration capabilities that can help your developers to code more securely without losing too much time in manual code reviews.

Codescanning detects violation from secure coding best practices, identifies vulnerabilities in dependencies, insecure framework configurations, ensures container images are free from vulnerabilities and infrastructures stay secure and compliant.

3. Security Testing

"The Attacker's view"

A final product needs to undergo thorough security testing before going to production. But to ensure that it stays secure, it continuously needs to be tested while in production as new vulnerabilities and attack vectors are discovered and features and infrastructures change.

Vulnerability scanning and dynamic scanning tools treat applications (mostly) like black boxes, mimicking an attacker's view and can run continuously and automated. A manual penetration testing activity adds an additional layer of security. Skilled pentesters can discover logic flaws, permission issues or hidden injection vectors that automated tools might miss. They also sort out false positives.

While penetration testing takes time, it is an additional layer that should be part of every secure development lifecycle and heps to ensures compliance to existing or emerging requirements.

secureIO

Plan

Discover

Attack

Report

While dynamic scanning and penetration testing are only pieces of the puzzle in application lifecycle security they can also be a good starting point to uncover your current exposure and point to weaknesses in your development processes.

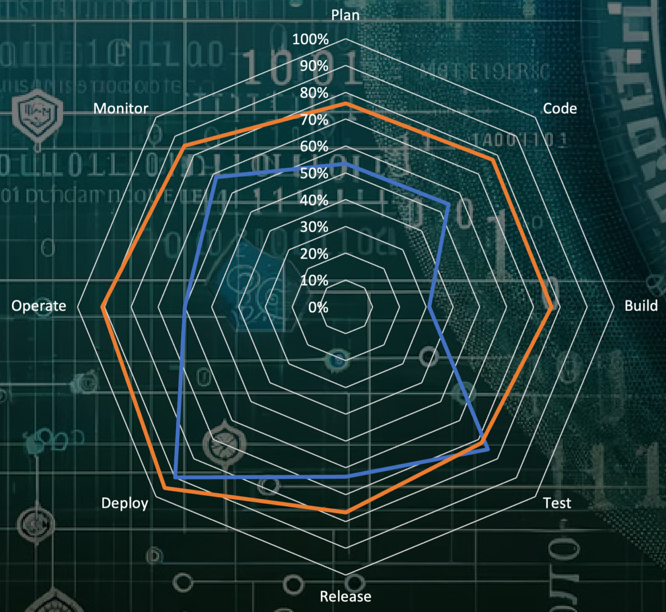
Ready for a global challenge?

Security has become a global challenge and there is no way not to be part of it. Are you prepared or still unsure where you stand and what are the next steps in your journey?

Assess your product security program maturity using industry standards like OWASP SAMM, BSIMM, DSOMM or tailored assessments for quick results. Create tactical and strategic roadmaps to mature your product security program based on your organization's business needs.

We strongly believe that Security needs to collaboratively support your teams and your business mission. To create a well anchored security program, you need to bring your development teams and product owners on board. Establishing a security culture is an essential building stone for your long-term business success.

To help bridge the gaps and respond to immediate requirements we can help out with our 360° application assessment incorporating security architecture reviews, code assessments and penetration testing to help you meet your application security requirements.



Example: Security maturity(blue) and planning (orange) depicted as spider diagram per DevOps phase

Test yourself

- ☐ Our security culture is well established and includes security champions, knowledge sharing and community of practices.
- ☐ Our application and infrastructure is designed according to best practices. We have considered all relevant potential attack vectors by continuously threat modeling our architecture.
- ☐ We have clear perspective how compliance requirements and product security align and support each other.
- ☐ We have automated code and implementation checks in our product development lifecycle and CI/CD pipelines with clear visibility and remediation processes for development teams and reporting to stakeholders.
- ☐ We continuously check our exposure, our applications and our infrastructure from an attacker's perspective, monitor for attacks and have established well working incident response processes
- ☐ We learn from past incidents, keep track of new attack vectors and security best practices for Cloud, AI, DevSecOps and are continuously measuring and optimizing our tools and processes.

Our Services

Strategic Planning & Maturity Assessment

360° Application Security Assessment

Threat Modeling

Code Reviews & Analysis

Tool Selection and Integration

Security Testing & Vulnerability Analysis

Training & Education

Culture & Community Building

Contact

secureIO GmbH
Ainmillerstr. 22
80801 Munich
GERMANY

www.secure-io.de
cybersecurity@secure-io.de

https://www.linkedin.com/company/secureio-gmbh/

secureIO